2017

Optimized terasort algorithm for data analytics: case of climate data analysis

Fiona Mugure Matu
Fuculty of Information Technology (FIT)
Strathmore University

Optimized Terasort Algorithm for Data Analytics: Case of Climate Data Analysis

Matu Fiona Mugure

Master of Science in Information Technology

2017

Optimized Terasort Algorithm for Data Analytics: Case of Climate Data Analysis

Matu, Fiona Mugure

089476

A research thesis submitted in partial fulfilment of the requirements of the Degree of Master of Science in Information Technology at Strathmore University

Faculty of Information Technology
Strathmore University
Nairobi, Kenya

June, 2017

Declaration

I declare that this work has not been previously submitted and approved for the award of a degree by this or any other University. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made in the thesis itself.

Matu Fiona Mugure

.....

June 2017

Approval

The thesis of Matu Fiona Mugure was reviewed and approved by the following:

Professor Ismail Ateya Lukandu, D. Sc.,
Associate Professor, Faculty of Information Technology,
Strathmore University

Dr. Joseph Orero (PhD),
Dean, Faculty of Information Technology,
Strathmore University

Professor Ruth Kiraka,
Dean, School of Graduate Studies,
Strathmore University

Abstract

Weather forecasting has proven valuable in unravelling the causes of the occurrence of natural phenomena and predicting of future climatic conditions. Subsequently, better preparation and policy making regarding these occurrences can be done using resultant information from techniques employed in weather forecasting. Analysis of vast amounts of data are characteristic of climatology hence require computing intensive techniques such as numerical weather prediction (NWP). This has made climate modelling a preserve of high performance computing (HPC) until the recent entrance of big data analytics. It is therefore necessary to optimize the algorithms used in the big data environment so as to give comparable performance to that offered by HPC environments. The study aimed at improving the big data MapReduce framework of analysis by optimizing the TeraSort benchmark algorithm. The algorithm proposed employed classical sort techniques and incorporated quantum computing mechanisms. Historical weather data collected at weather stations across the world was gathered and converted into organised, human readable format to suffice as input to the program. The proposed algorithm constituting of a map, sort and reduction phase transformed the bulky observational data into a compact summary of monthly temperature averages in linear complexity. This is a significant improvement in performance in comparison to the TeraSort algorithm on a single node. The study concludes by suggesting areas that may be explored for further optimization with emphasis on quantum computing capabilities.

Table of Contents

List of Figures

List of Tables

List of Equations

List of Abbreviations

ASCII	-	American Standard Code for Information Interchange
AFCCC	-	US Air Force Combat Climatology Center
CPU	-	Central Processing Unit
FLOPS	-	Floating Point Operations per second
FNMOD	-	US Navy's Fleet Numerical Meteorological and Oceanographical Command Detachment
HPC	-	High Performance Computing
IBM	-	International Business Machines
IDE	-	Integrated Development Environment
ISD	-	Integrated Surface Data
HDFS	-	Hadoop Distributed File System
NASA	-	National Aeronautics and Space Administration (USA)
NCDC	-	National Climatic Data Center
NERA	-	National Energy Resources Australia
NOAA	-	National Oceanic and Atmospheric Administration
NWP	-	Numerical Weather Prediction

Definition of Terms

Algorithmic Complexity - the quantity of computational resources (time, storage, program, communication) used up by an algorithm (Stephens, 2013)

Big Data - is the "data" principally characterized four "V"s. They are Volume, Variety, Velocity and Value (Iafrate, 2015)

Numerical Weather Prediction - consists of automatically performing meteorological forecasts and implementing a series of clearly identified processes, determination of the initial state of the atmosphere, computation of the final state at a given range, computation of the weather parameters at the local scale and tailoring and dissemination of results (Coiffier, 2011)

Quantum Computing - quantum computing as the theory of information processing with the traditional (classical) information replaced with quantum information (Hirvensalo, 2013).

Acknowledgements

I would like to acknowledge the assistance of my research supervisor, Prof. Ismail Ateya for his continuous guidance as well as my classmates and family for their passionate support.

Dedication

This research is dedicated to my beloved parents for the support they have tirelessly accorded me through many hardships to achieve academic excellence. I thank God Almighty for His abundant grace throughout this academic journey.

Chapter One: Introduction

1.1 Background of the Study

Weather can be described as the way atmosphere is behaving, mainly with respect to its effects upon life and human activities. Weather focuses on short periods of change in the atmosphere while climate is the average weather pattern of a geographical area over long periods of time (National Aeronautics and Space Administration, 2015). Climate can also be defined as the distribution of weather and other variables that are part of the climate system according to Guttorp (2014).

Studying climate changes has proven to be essential because of its direct impact to people around the world. For instance, a rise in global temperature levels results in a rise in sea levels therefore changing precipitation and other climate conditions of an area. Crops, water sources and forest cover are all affected by climate variations. Consequently, human and animal health are in turn affected as well as other ecosystems present.

The practice of predicting the weather begun thousands of years ago largely based on intuition, but evolved into a fully-fledged science that climatology has become. Indeed, climatology has become so complex in our present world that climate modelling simulations gobble up large amounts of computing power. Furthermore, as McGuffie and Henderson-Sellers (2014) noted, the largest single use of computers in the world is climate research and still is the case upto date, Cray supercomputers being a good example of high performance machines that are used for climate data analytics.

Big data analytics, utilizing clusters of commodity hardware, is quickly taking centre stage when it comes to climate research. Giving similar processing capability to high performance computing (HPC), projects on climate change are able to make sophisticated analysis using big data analytics.

Climate model data, data from satellites and observational networks is viewed as big data having different levels of the three characteristics of big data which are volume, variety and velocity (Lautenschlager, Adamidis, & Kuhn, 2015). The rate at which data is currently being produced is higher than in the recent past, making traditional storage and processing difficult. Sophisticated data collection and

transmitting methods have made this possible. Climate data however does not have much variety since it operates within a limited number of well-defined data formats.

1.2 Problem Statement

In as much as the quality of weather forecasts and studies to do with climate variations have improved, Balaji (2015) notes that a major disruption in computational code design is taking place. Climate science has organized itself on a global scale, whereby information analysis and sharing between entities around the world is necessary. A great interest in the use for climate data in diverse domains has emerged, resulting in a dire need for scientific scalability of systems involved.

Considering factors such as increased resolution, an increasing number of experiments and a greater area over which they are carried out in various climate services, computing facilities need to adapt accordingly (Joussaume, 2012). Big data analytics has recently started being used for scientific analysis of climate data as opposed to the predominant use of HPC since it is more affordable in monetary terms. There therefore is a need for rapid processing capability while conciously conserving computing resources using a cost effective architecture.

1.3 Research Objectives

i. To identify weather data commonly required in both climate analysis and modelling
ii. To identify the challenges associated with analysis of large weather datasets
iii. To review existing sort algorithms used for computational analysis of large datasets
iv. To develop an algorithm that may be used to optimally sort large datasets
v. To test the algorithm using climate model output data

1.4 Research Questions

i. Which weather data is used in analysis and modelling of climate?
ii. What challenges are presented by analysis of large weather datasets?

iii. Which algorithms have been used in the analysis of large weather datasets?
iv. How can an optimal sorting algorithm be developed based on the reviewed algorithms?
v. How can the algorithm be validated using climate model output data?

1.5 Justification

As stated in the background statement, changes in climate conditions affects food production, water availability, wildlife and human health. Weather conditions such as storms, damage infrastructure like roads, rail networks and buildings (National Energy Resources Australia, n.d.). This is why Climates Science is such an important study that any improvement in its analysis is welcome. Platforms that perform analysis at a quicker rate than they could before simply means that a life is saved, an organization is better prepared, and an economy flourishes.

Furthermore, data analytics is not limited to climatology but various other fields that present data with similar characteristics as climate data. Fields such as genetics, engineering, finance are examples of domains that churn out data which requires high analytical power. All these domains will benefit from an enhancement of a programming model such as MapReduce which is relevant in their analysis.

Domains described above have been known to utilize high performance computing (HPC) for specialized problems because of the computing power required. This is slowly evolving to the use of commodity hardware in clusters to harness similar amounts of processing power. Optimization of algorithms running on such architectures presents an efficient use of processing power and in effect conservation of resources. The focus will shift to better utilization of available processing power rather than an infinite addition of resources to support power-hungry algorithms.

1.6 Scope

Of particular interest to this study is to optimize climate data analytics within the big data environment. Climate model output data, which is voluminous and requires big data tools for storage, analysis and mining, will be used. The TeraSort benchmark algorithm will optimized for this data considering that the MapReduce programming model, present in the big data environment, uses this benchmark to

deduce the model's performance. Assumption made are that performance solely relies on the algorithm design.

1.7 Limitations

Given the time allocated for this study, research was limited to only the sorting functionality within MapReduce without distributed processing. Citing that big data has both capabilities of handling structured and unstructured data, focus was be on structured weather data. Data used to design and test the algorithm was that which was accessible to the researcher. Quantum computing operations were simulated because of the inability by the researcher to access a quantum computer.

Chapter Two: Literature Review

2.1 Introduction

To provide an empirical framework, a review of data necessary in the analysis of climatic conditions, existing sorting algorithms, challenges faced and probable causes of these challenges will be delved into. To appreciate the purpose of this research, it is imperative to grasp computational algorithmic concepts that this study is anchored in.

Theory of computational complexity and quantum theory as expressed by various scholars will constitute a theoretical framework of interest in this discourse. Sorting algorithms of note will be reviewed, taking into account their performance based on the former theory. The two frameworks will guide in building a conceptual framework which puts together views from both approaches.

2.2 Data used in Climate Analysis and modelling

Climatology sprung from a much older study known as meteorology, which focused on processes occurring in the atmosphere with the intention of weather forecasting. Climatology, is concerned with long durations of changes in weather and or the climate of a region. It aims to analyse macro-level effects such as global warming, large-scale phenomena such as typhoons and tsunamis. It therefore assists in planning ahead for areas deemed as hot spot areas.

Meteorology was once an activity undertaken by a forecaster based on their experience but is now sophisticated due to the advent of computers according to (Harper, 2012). Climatology and meteorology have not only matured in complexity but also complement each other according to Edwards (2010). One of the techniques that have characterized this great growth in atmospheric sciences is climate modelling.

2.2.1 Climate modelling

A climate model is an idealised representation of the climate conditions of a region built with the goal of gaining a better understanding of an aspect in that climate (McGuffie & Henderson-Sellers, 2001). Models therefore assist in large scale analysis

which facilitates discovery of patterns and or phenomena that may be present and worth noting. It therefore utilizes simulation and vast amounts of data to emulate the closest possible resemblance to the actual climatic conditions present in a geographical area.

According to Edwards (2010), computer modelling of the climate brought about a "voracious appetite for data" whereby the geographical sizes of the models increased with the increase in computing power. Hemispheric and eventually global models were being developed as early as in the 1960s. McGuffie & Henderson-Sellers (2014) describes climate models as "collections of software" that are formulated based on statistical approaches. Subsequently, numerical weather prediction or otherwise referred to as NWP, was made possible by modelling capabilities. NWP involves forecasting weather conditions using computers (Harper, 2012). Forecast models are built and deductions made using numerical methods.

All models are dependent on observations as their cardinal source of data (Warner, 2011). Warner qualifies the use of models as being more cost effective in augmenting observations since data collection over vast geographical areas is expensive. The relationship between observations, models and theory the models are constructed upon is shown in figure 2-1

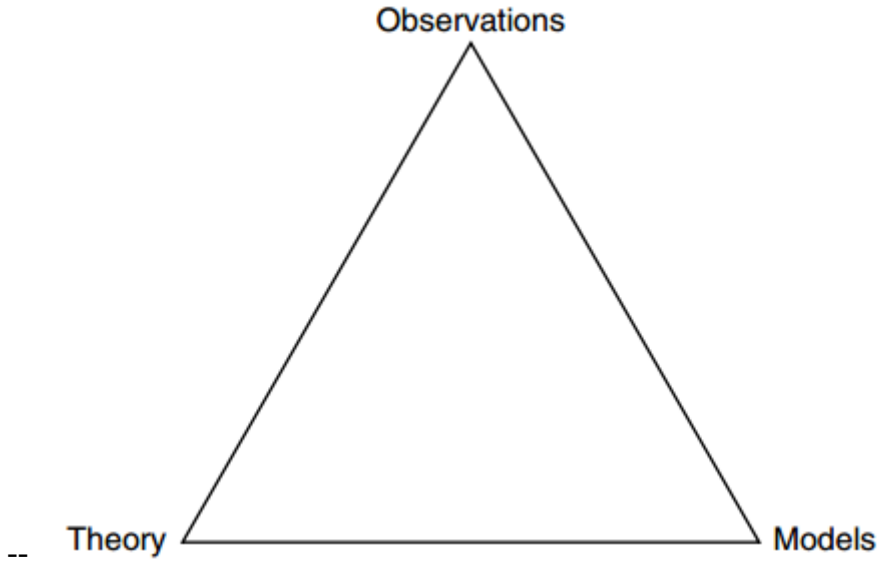


Figure 2-1: Relationship between Observations, Theory, and Models (Source: Warner, 2011, p. 3)

This relationship demonstrates that observational data collected at weather stations is the driving factor behind usable models and formulated theories.

Climate data fields are essential for verification of global climate models (GCM) and are obtained by means of interpolation of observations recorded at weather stations (Thomas & Herzfeld, 2004). Furthermore, to discover occurrence and cause of phenomena that takes place over a long duration such as global warming, historical data (in this case temperature data) for as far back as a hundred years or more is required for accurate analysis to be done according to Spellman (2012).

2.3 Approaches used in computational analysis of climate data.

2.3.1 High performance computing (HPC)

High Performance Computing generally refers to the practice of aggregating computing power by use of parallel processing techniques for solving complex computational problems. Consequently, parallel architectures and algorithms are necessary in this form of computing for efficient use of computing resources.

Climate models, as noted by Edwards (2010) in 2.2, required large amounts of data and consequently, became tremendously hungry for processing power. And as climatology continuously borrowed from the other sciences such as physics, extensive research into the physical laws governing the weather was being conducted. Increased complexity of models resulted in building of increasingly powerful computers and software that could deliver forecasts within acceptable execution times (Grandinetti, Joubert, & Kunze, 2015).

A hypothesis, in this case may be explained as an explanation of an observable phenomenon that can be measured using well-defined metrics. Hypothesis may also be used to predict the observation of one or two phenomena that may not be observable or measureable yet according to Grandinetti et.al (2015). Models are mathematical representations of these hypotheses.

Such phenomena can be simulated with the help of experiments and supporting observational data to construct a system similar to the real world physical system. Monte Carlo, neural networks, cellular automata are all techniques that may be applied alternatively according to Grandinetti et.al (2015).

Figure 2-2 illustrates the human interaction with all contributing aspects to climate modelling.

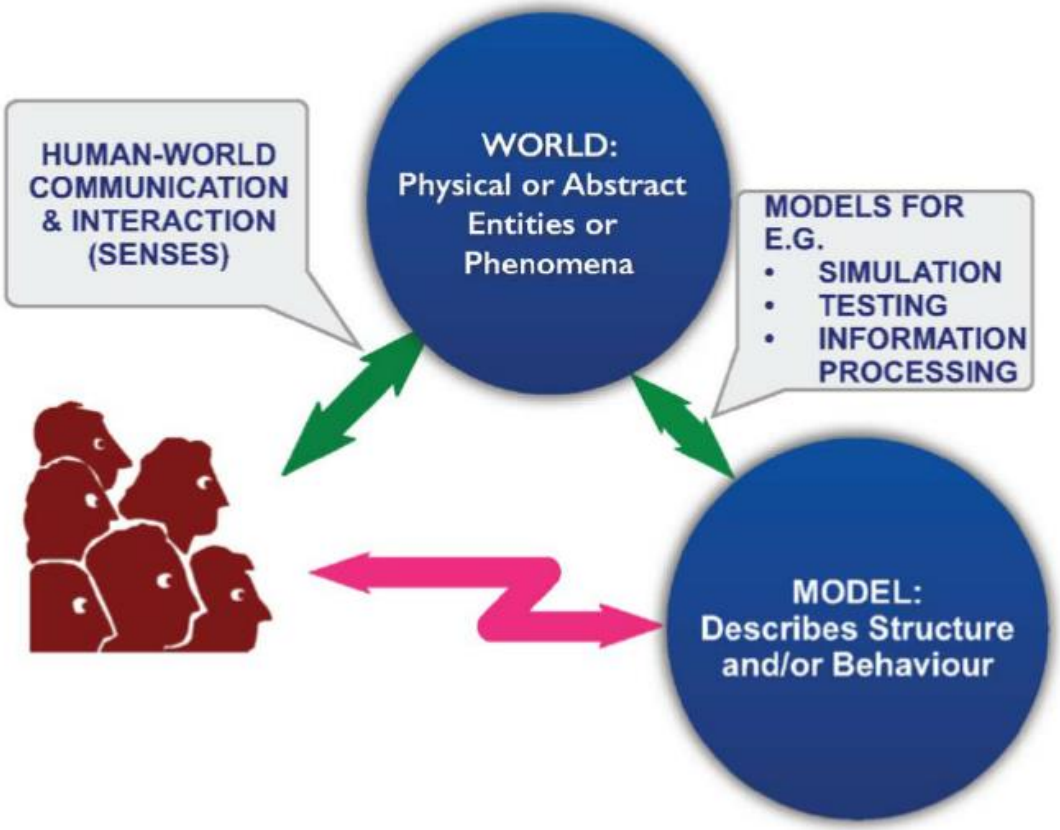


Figure 2-2: Human- World- Model Interaction (Source: Grandinetti, Joubert, & Kunze, 2015, p. 10)

Computer models aid in solving complex mathematical models that would otherwise be impractical to use analytically. A collaboration between the use of analytical models and numerical models allow for the construction of the computational model that should be carried out by a suitable computer. Software algorithms used should also be adapted to that underlying architecture for optimisation of effienciency.

Figure 2-3 illustrates how climate simulations are achieved using computation and numerical tools

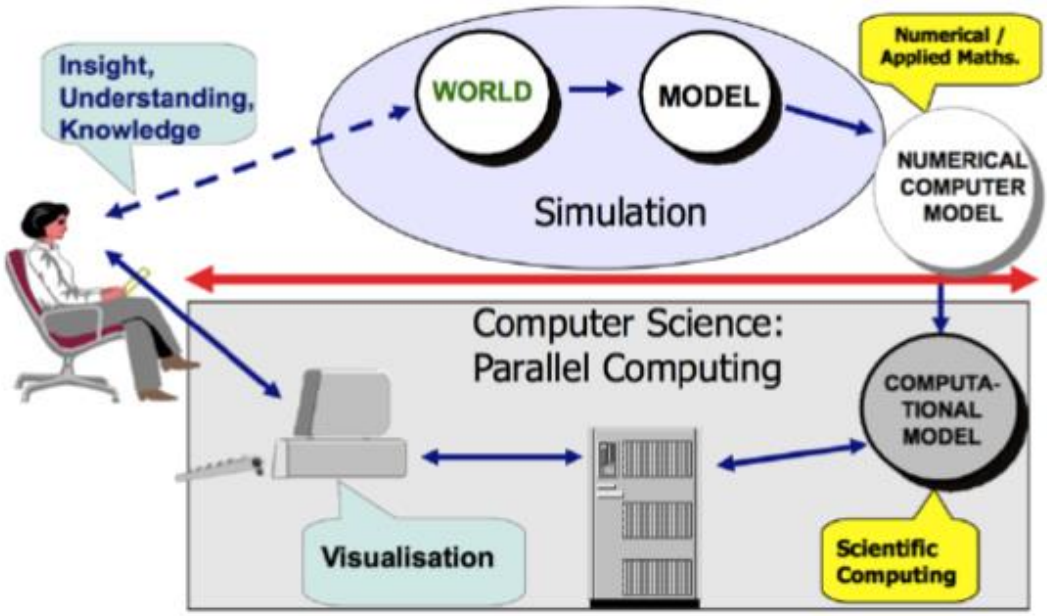


Figure 2-3: Computational models (Source: Grandinetti, Joubert, & Kunze, 2015, p. 11)

There are however challenges with the ever ambitious research in climate whereby experimental facilities and large scale simulation have exploding amounts of data generated. Grandinetti et al.(2015) describes the resultant difficulty faced by scientists and HPC centres is in storing, accessing, sharing, managing and analysing massive data sets.

A Traditional HPC approach that is still in use is supercomputing with clustering while newer trends are the use of networks of workstations (NOW) or network of networks (grid computing) according to Hanuliak (2014). Supercomputing aims at achieving the best achievable performance for demanding computational needs (Graham, Snir, & Patterson, 2005). Systems used in supercomputing are highly specific to their intended purpose and their performance measured in terms of Floating Point Operations per second (FLOPS).

Several components constitute a climate system making climate modelling computationally intensive. Thousands of floating point operations and high amounts of overhead due to internal communication within the computing unit occurs in processing climate models. This is the reason supercomputers are well suited to execute them. Another aspect that makes climate modelling suitable for high performance computing is the grid resolution of the geographical area being considered. The higher the vertical and horizontal resolution, the greater the

computational strain. This is because the physics involved increases in complexity while a larger amounts of data are required for assimilation purposes according to (Zwieflhofer & Mozdzynski, 2005) .

Most climate models however, as Graham (2005) explains, typically have grid sizes of hundreds of kilometres, and have few components and oversimplified parameterizations. Such models have rarely simulated future climate changes beyond a century. Models that can represent more detail are necessary to unearth meteorological patterns. A massive need for increased computational resources therefore exists.

2.3.2 Big data analytics

The more information is collected, the wider the area also extending into the atmosphere over which weather data is collected, the greater the accuracy of predictions made as noted in 2.3.1.1. Weather prediction may be sufficiently solved if complete global data can be processed effectively (Grandinetti, Joubert, & Kunze, 2015) . Such data can be categorized as big data taking into account that it is extremely voluminous and churned out at a high rate.

Demand for data systems and services is growing with the sheer massive size, resolution and complexity of large-scale simulations that produce petabytes of data per simulation. Big data analytics offers an alternative to the expensive and highly customized supercomputing model discovered in 2.3.1.1. A popular big data framework is the Hadoop MapReduce Framework.

A. Hadoop MapReduce model

MapReduce is a popular data-intensive distributed computing model especially for the use of large-scale data-parallel applications such as web indexing, data mining, and scientific simulation (Tannir, 2014). It was designed by Google Research in the early 2000s. The most popular implementation of MapReduce is by Apache Hadoop even though other big data processing platforms such as Spark, Disco and MARIANE (MapReduce Implementation Adapted for High Performance Environments) have their own implementations of it.

MapReduce is the processing part of Hadoop and consists of two main functions: a mapper and a reducer that can be executed in parallel on multiple machines. Input datasets provided to a MapReduce job are split into independent data chunks that undergo the map phase in parallel. The map phase produces key, value pairs for each word. These pairs are then passed to the reducer as input (Akthar, Ahamad, & Ahmad, 2016). The reducer then produces the final output values by grouping together values based on their keys.

A MapReduce program manages data using a hash table and contains in the following standard components according to Hashem and Ranc (2016):

i. Several mappers, which receive input data and generate keys from the data
ii. One or more reducers that perform data consolidation
iii. Optional combiners, which act as semi-reducers which intermediately consolidate data before final reduction
iv. Optional partioners, which control the partitioning of the keys from the map phase.
v. Optional shuffle and sort, recommended to yield better processing performance.

Figure 2-4 shows a diagrammatic representation of the MapReduce model with the components listed included and the flow of execution follows a top-down design.

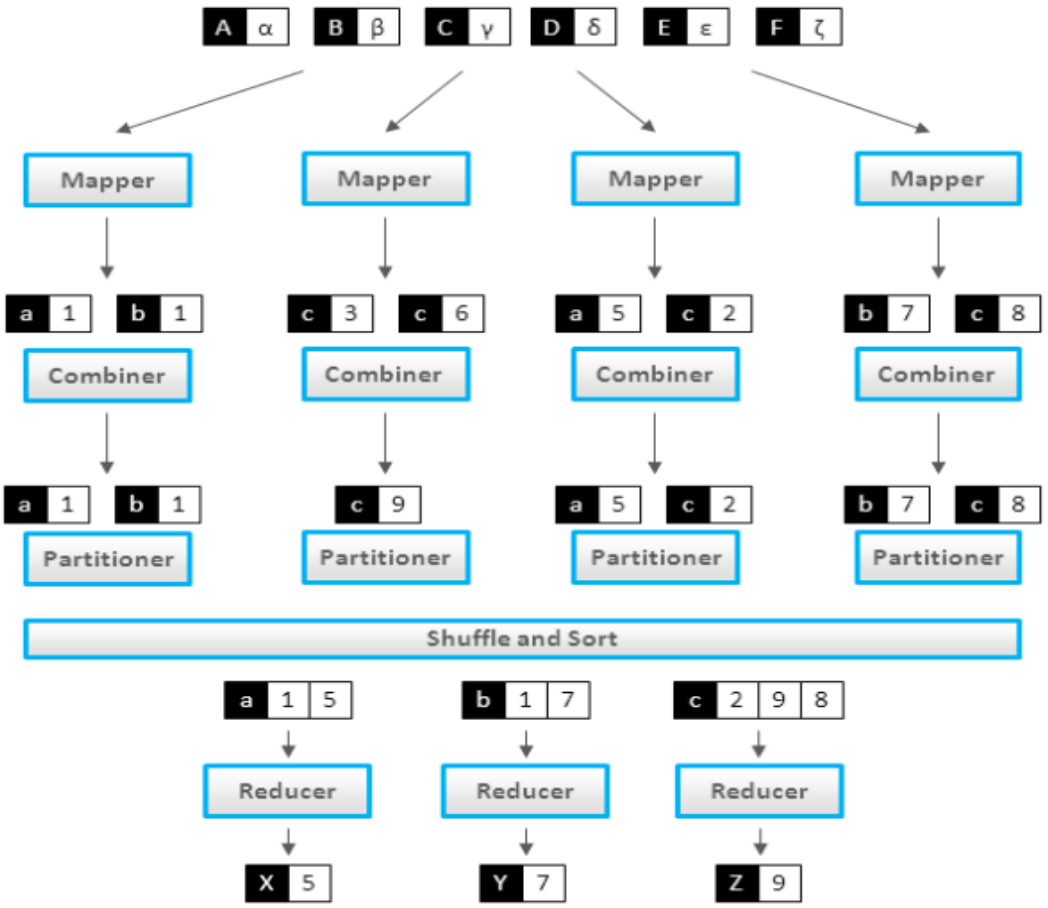


Figure 2-4: MapReduce Framework (Source: Hashem & Ranc, 2016,p. 160)

The processing time of input data with MapReduce may be affected by multiple factors. Algorithmic factors when implementing mapping and reducing functions as well as external factors may affect the performance of MapReduce job. Factors affecting the performance of MapReduce according to Tannir (2014) are :

i. Hardware (or resources) such as CPU clock, disk I/O, network bandwidth, and memory size.
ii. The underlying storage system.
iii. Input data size, shuffle data, and output data, which are all closely correlated with the runtime of a job.
iv. The nature of a job may not be well conceptualized using a MapReduce approach making them inefficient when converted to a MapReduce design

2.4 Classical Sorting Algorithms

Sorting algorithms are the most widely used type of algorithms explaining why they are well studied (Song, Xu, Zhang, Pahl, & Yu, 2015). Many sorting algorithms exist and use various sorting mechanisms that bring about significant differences in their efficiency regardless of the hardware and software used (Cormen, Leiserson, & Rivest, 2009).For the purpose of this study, we will review quicksort algorithm and TeraSort algorithm which are of particular relevance.

A. Quicksort

This is a comparison based sort with a best and average case complexity of $O(N \log N)$ and a worst case complexity of (N^2) . It uses a divide and conquer approach which involves dividing the problem into sub problems which are easier to solve in what is thought as conquering the problem.

The algorithm begins by arbitrarily selecting an item in the array containing the values to be sorted. This item is considered as a pivot and the algorithm rearranges elements to the left of the pivot are less than the pivot value and all elements to the right are greater in value than the pivot. The elements in either side of the pivot are not necessarily in order but are on the intended side of the pivot.

A new pivot is selected from other remaining values in the array and rearrangement of values relative to the pivot is repeated recursively until all items in the array are sorted. Taking figure 2-5 as an example of an unsorted array, the operation of quicksort is shown in figure 2-6 to arrive at a sorted array. The rightmost element is chosen in successive recursions as the pivot value. The letters p and r indicate the first and last indices of the array being sorted in each step while q represents the index used as the pivot.

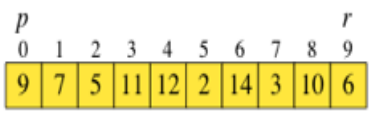


Figure 2-5: Unsorted Array (Source: Khan Academy, 5th June 2017)

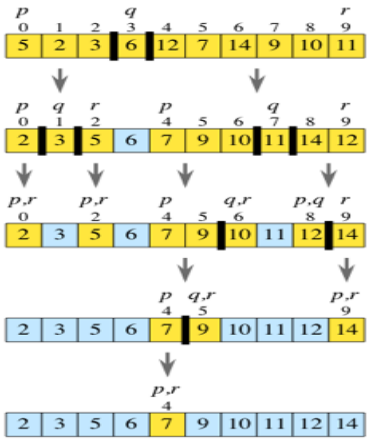


Figure 2-6: Quicksort Operation (Source: Khan Academy, 5th June 2017)

The algorithm iterates through all items in the array at least once while partitioning the array. This is done in O(N) time.

The multiplicative function is determined by the nature of the sub problems whereby the partitions result in an unbalanced tree in the worst case scenario, a well-balanced tree in the best case scenario and a partially balanced tree in the average scenario. It therefore follows that in the worst case scenario, where the array constitutes of values arranged in a descending fashion, all values are compared to each other resulting in $O(N)$. In the best case scenario where all values are arranged in the desired ascending order, comparisons are halved with each recursion resulting in $O(\log N)$ time.

Multiplying the complexities in both the divide and conquer step results in

$O(N \times N) = O(N^2)$ in the worst case and $O(N \times \log N) = O(N \log N)$ in the best case.

Quicksort does have the optimum complexity for a comparison sort in a best case and average case scenario but escalates to a much greater complexity in the worst case scenario which may be the case in a real life setting. Applying quicksort for sorting large weather datasets hence therefore becomes impractical in a big data setting.

B. TeraSort

TeraSort is a benchmark algorithm used to measure the performance of MapReduce framework on a Hadoop cluster. It is based on the standard MapReduce sort and sorts a terabyte of data generated by a different program known as TeraGen.

TeraSort begins by sampling input received to generate sorted keys and writing the list of keys to the Hadoop Distributed File System (HDFS). The input is then partitioned using the keys generated and sent to reducers based on the range of keys each reducer is assigned. Radix trees or simply referred to as tries are used to index the first two bytes of each key in an effort to improve performance (Bhardwaj, Singh, & Vanraj, 2015).

The underlying sort mechanism used to sort keys in TeraSort is quicksort despite the weakness pointed out in the 2.4.2 B. The algorithm structure of TeraSort is shown in figure 2-7.

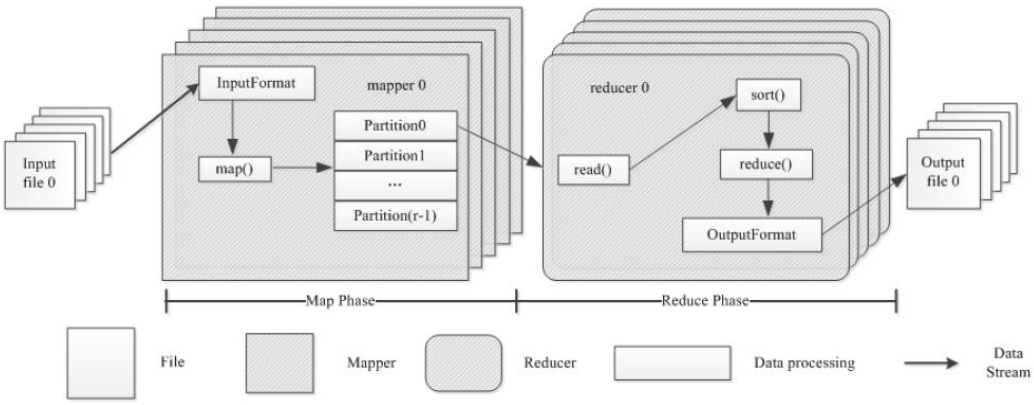


Figure 2-7: Structure of TeraSort Algorithm (Source: Song, Xu, Zhang, Pahl, & Yu, 2015, p. 3)

Figure 2-8 shows how TeraSort builds a trie for key prefixes of length 2.

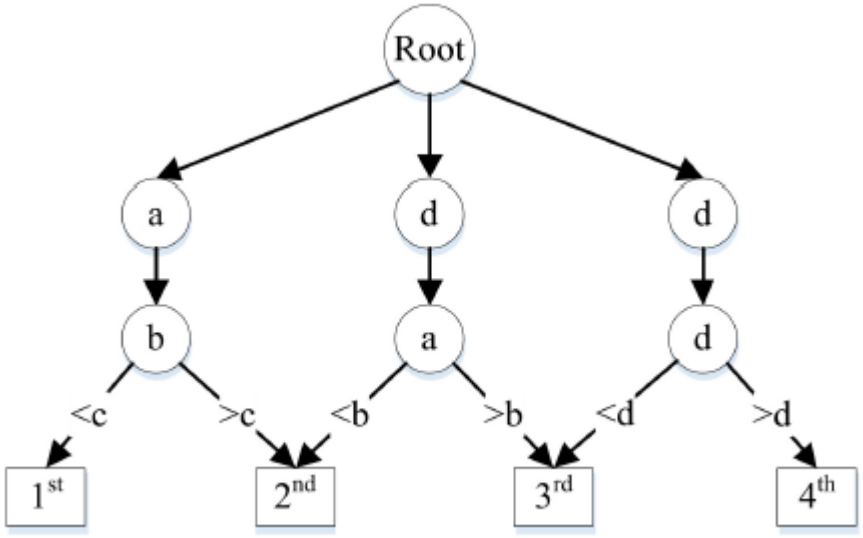


Figure 2-8: Simple trie in TeraSort (Source: Song, Xu, Zhang, Pahl, & Yu, 2015, p. 3)

There have been attempts in research to optimize TeraSort more specifically in the manner in which it distributes the reduction load across nodes in the network. Two techniques have been suggested and experimentally tested by Slagter, Hsu, Chung, and Zhang, (2013) which attempt to improve the performance of TeraSort namely Xtrie and Etrie.

i. Xtrie

A weakness identified in TeraSort was the use of quicksort algorithm making it necessary to store all keys generated in memory therefore limiting the possible key sample size. Another weakness pointed out was consideration of only 2 characters in each key while partitioning. The two characteristics were noted to reduce the ability to balance the load among nodes in the cluster.

Insertion and search for keys using a trie yields a complexity of O(k) where k is the length of the key which is better than that of quicksort. The memory footprint of a trie is fixed making it suitable in adding prefixes for a large sample of keys.

The trie is implemented using an array and the trie code equation is as described in equation 2.1

-1:

$$TrieCode = \sum_{n=1}^{TotalWord} W_n \times 256^{n-1} \quad (2.1)$$

ii. Etrie

Both TeraSort and Xtrie use an array to represent a trie which wastes space since arrays have a fixed size upon declaration and throughout the program. For applications using small sized keys, and considering characters that the mapper filters out when creating keys, the resultant waste of memory is significant.

Etrie technique proposes an algorithm that reduces the number of elements the trie considers by grouping ASCII characters into three groups. Alphanumeric characters is the first group which includes upper and lower case alphabet and numbers. Other is the second group that contains all other characters not included in the alphanumeric group. The null is the third group containing keys whose value is shorter than the trie levels.

The 256 characters included under ASCII are therefore reduced to 64 elements and consequently the memory footprint of the two level trie is reduced by a factor of $\frac{1}{16}$.

The EtrieCode equation as shown in equation 2.2:

2-2: Etrie Equation

$$ETrieCode = \sum_{n=1}^{TotalWord} W_n \times 64^{n-1} \quad (2.2)$$

Distributing the sample keys evenly is made possible since deeper tries can be built using Etrie. More character prefixes can be considered and the range of keys per prefix is reduced resulting in smaller portions of the load being assigned to each reducer. Figure 2-9 and table 2-2 show findings from the research carried out.

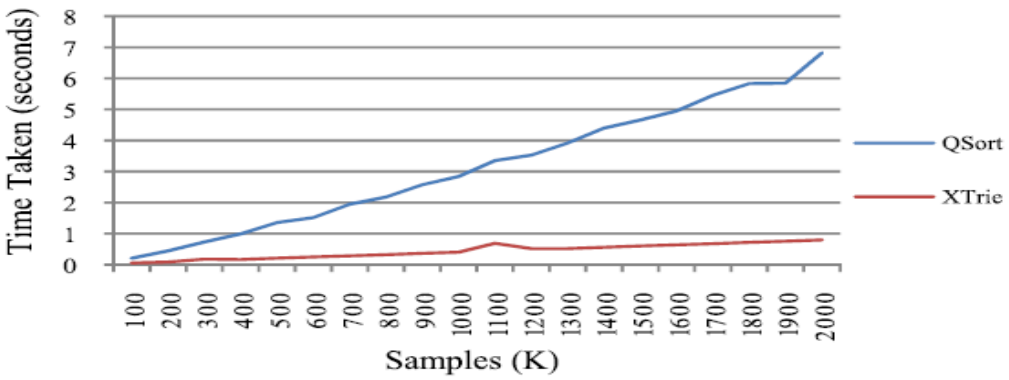


Figure 2-9: Time complexity comparison of QuickSort and Xtrie (Source: Slagter, Hsu, Chung, & Zhang, 2013, p. 552)

Table 2-1: Space Utility of Xtrie as compared to Etrie (Source: Slagter, Hsu, Chung, & Zhang, 2013, p. 552)

	Xtrie	Etrie
Number of nodes	77	56
Space required	256	64
Space utilization (%)	30.08	87.5

Overall, according to the experimental results achieved from the two techniques, Xtrie and Etrie outperform the partitioning method used in TeraSort in terms of time complexity. The Etrie architecture was concluded to conserve memory, allocate more computing resources on average, and do so with less time complexity.

2.5 Theoretical Framework

Cormen, Leiserson and Rivest (2009) define an algorithm as any well-defined computational procedure that takes in input values and produces some output values. It is thus a sequence of computational steps that transform the input into the output commonly with the intention of solving a known problem. A review of two theories,

computational complexity theory and quantum theory, is made with reference to their suitability to this study.

2.5.1 Computational Complexity Theory

The amount of resources used by a computer to solve a problem, is referred to as the complexity of the computation. Computational complexity theory therefore attempts to offer a means of measurement of these resources, most critical of them being time and space according to Wichert (2013).

Computational complexity is thus expressed as a function of the size of input of an instance of a given problem (Kaye, Laflamme, & Mosca, 2007). For instance, a problem involving multiplication of a set of n numbers may be solved computationally using $n^2 + 2$ units of time or space. This gives a relative indication of how resource intensive the algorithm is rather than give exact amounts required because other factors such as the computer hardware come into play during execution.

Avoiding low- level detail such as the computer hardware, measurements are done based on the order of the polynomial function used to describe the complexity of the problem, ignoring any constants present. Hence, $n^2 + 2$ and $8n^4 + 2n + 3$ are said to have a runtime of $O(N^2)$ and $O(N^4)$ respectively.

A. Orders of magnitude

In order to classify an algorithm as "efficient" or "inefficient", their complexity is considered based on their growth function (Wilf, 2002). Using this method makes it easier to compare the resource utilization of algorithms that solve the same problem.

Symbols used in this classification are:

i. 'o' ('little oh of')
ii. 'O' ('big oh of')
iii. 'Θ' (Big theta)
iv. '∼' ('asymptotically equal to')
v. 'Ω' ('omega of')

Let f (x) and g(x) be two functions of x. Each of the symbols listed compares the rate of growth of f and g. An understanding of these notations as explained by Wilf (2002) is necessary to aid in determining complexities of algorithms.

i. ‘o’ (little oh)

Is mathematically defined as

$$\lim_{x \to \infty} \frac{f(x)}{g(x)} = 0$$

If f (x) = o(g(x)), then this means that f grows slower than g does when x is very large (Wilf, 2002). Examples of functions that exhibit this relationship are shown in (a) and (b) below:

a) $x^3 = o(x^5)$
b) $\sin x = o(x)$

In the case of two computer programs, where one inverts n×n matrices in $600n^3$ units of time, and the other does the same task in $o(n^{2.7})$ units of time, then for all sufficiently large values of n the performance of the second program will be superior to that of the first program. However, the first program might execute faster on small matrices.

ii. ‘O’ (big oh)

Is mathematically defined as

$$\lim_{x \to \infty} \frac{f(x)}{g(x)} < \infty$$

When f(x) = O(g(x)), f is said to not grow faster than g but may grow at the same rate or slower than g. Therefore, ‘O’ notation allows for a less or equal to comparison of functions. For example, the below are relationships expressed in ‘O’ between functions in (a) and (b)

a) $\sin x = O(1)$

b) $x^3 + 16x^2 + 38\cos x = O(x^5)$

'O' is sufficient in general description of an algorithm but is less specific as compared to 'o' notation. It gives the worst case use of resource by an algorithm and also commonly referred to as the asymptotic upper bound.

iii. 'Θ' (big theta)

Is mathematically defined as

$$\lim_{x \to \infty} \frac{f(x)}{g(x)} \in \mathbb{R} > 0$$

A relationship f(x) = Θ(g(x)) means that f and g grow at an equal rate but their coefficients are unknown. To show such a relationship between functions, (a) and (b) act as examples

a) $(x+1)^2 = \Theta(3x^2)$
b) $(1 + 3/x)^x = \Theta(1)$

'Θ' offers a more exact description of an algorithm as compared to 'O' and 'o'. If $f(x) = \Theta(x^2)$ then $\frac{f(x)}{x^2}$ is bound between two non-zero constants for sufficiently large values of x. Therefore, f grows quadratically with x.

iv. '∼' ('asymptotically equal to')

Is mathematically defined as

$$\lim_{x \to \infty} \frac{f(x)}{g(x)} = 1$$

The expression f(x) ~ g(x) means that f and g grow at the same rate and more specifically, that their ratio is equal to 1 for sufficiently large values of x. To illustrate this relationship between functions, (a) and (b) serve as examples

Examples are:

a) $x^4 + x \sim x^4$

b) $(2x^3 + 8x + 5)/(x^2 + 13) \sim 2x$

This notation is the most specific of all notations, giving specific coefficients unlike 'Θ' notation. $3x^2 = \Theta(x^2)$ may be deemed as true but $3x^2 \sim x^2$ is not considered true.

v. 'Ω' (big omega)

Is mathematically defined as

$$\lim_{x \to \infty} \frac{f(x)}{g(x)} > 0$$

'Ω' negates the 'o' notation. If, $f(x) = \Omega(g(x))$, then it is correct to say that $f(x) = o(g(x))$ is not true. This notation is useful in expressing the least amount of resource that can be used to successfully complete a task. For instance, to multiply a pair of matrices each with n elements will take a program $n \times n = n^2$ units of time therefore possesses a complexity of $\Omega(n^2)$. It therefore represent best case use of resources and is commonly called the asymptotic lower bound of an algorithm.

B. Common runtime functions

Having discussed the notations useful in estimating the resource usage of an algorithm, it is necessary to have an understanding of actual functions used together with these notations. In this way, conclusions may be drawn in terms of whether an algorithm is efficient of inefficient in its resource utilization.

The 'O' notation has been used to describe these mathematical functions because it represents the worst case as discussed in 2.4.1.1.2. This pessimistic approach ensures an algorithm is gauged safely. The arrangement of control structures in the program body informs the mathematical function an algorithm exhibits.

The functions are briefly discussed from 2.4.1.2.1 through to 2.4.1.2.8 according to Stephens (2013). N is considered to be the size of input or the number of input items.

i. O(1)

This function represents an algorithm that carries out a task in constant time irrespective of the size of the input. In most cases, such algorithms carry out trivial tasks since they cannot operate on all input items in constant time.

ii. O($log\ N$)

This describes an algorithm that divides the input items by a fixed fraction in repetitive steps. The logarithm of N grows much slower than N resulting in quick execution as the input size is increased.

iii. O(N)

An algorithm with this complexity is said to have a linear complexity, performing less efficiently than O($log\ N$) but quite well in reality.

iv. $O(N^{1/2})$

An algorithm with this complexity performs better than that of complexity O(N) but worse than that of O(log N). This makes it suitable for practical application.

v. O(N $log\ N$)

To represent the complexity an algorithm that iterates through all members of an input set then partitions the input set by a fixed fraction, O(N × log N) is used.

Comparison based sorting algorithms such as merge sort are described in this way. It is considered the optimal complexity for any comparison based sort. Comparison based sorts compare items within the input set

vi. $O(N^2)$

Algorithms that have 2 nested iterations over all items in the input set, have $O(N^2)$ performance. The degree of N varies depending on the highest number of nested loops present within their body. If 3 nested loops are present, then the performance is $O(N^3)$ and if 4, the complexity is $O(N^4)$. An algorithm with a complexity that has a degree of N has a polynomial runtime and the higher the degree, the worse it performs.

vii. $O(2^N)$

This function grows very quickly, making algorithms with this complexity suitable for small values of N dictating an optimal selection of input. Heuristics may be beneficial in this case, where good but not the best results are expected from the algorithm.

viii. O(N!)

N factorial is defined integers greater than 0 as:

$$N! = 1 \times 2 \times 3 \times \dots N$$

It grows much faster than 2^N making it almost impractical to use algorithms with this complexity. Such algorithms are only applicable with an optimal arrangement of input items.

Table 2-1 numerically compares growth rates of the functions discussed to illustrate the characteristics explained about each function.

N	$LOG_2(n)$	SQRT(n)	n	n^2	2^n	n!
1	0	1	1	1	2	1
5	2.32	2.23	5	25	32	625
10	3.32	3.16	10	100	1,024	1.0×10^{7}
15	3.90	3.87	15	225	3.3×10^{4}	2.9×10^{16}
20	4.32	4.47	20	400	1.0×10^{6}	5.24×10^{24}
50	5.64	7.07	50	2,500	1.1×10^{15}	1.8×10^{83}
100	6.64	10	100	1×10^{4}	1.3×10^{30}	1.0×10^{198}
1000	9.96	31.62	1,000	1×10^{6}	1.1×10^{301}	—
10000	13.28	100	1×10^{4}	1×10^{8}	—	—
100000	16.60	316.22	1×10^{5}	1×10^{10}	—	—

Table 2-2: Comparison of Runtime Functions (Source: Stephens, 2013, p. 17)

Paying close observation to the numerical run times, the most acceptable complexities for large values of input N are $\log_2 n$, $n^{\frac{1}{2}}$ and n.

2.5.2 Quantum Theory

Quantum physics is a branch of science that deals with discrete, indivisible units of energy called quanta as described by the Quantum Theory (Prabhakaran, 2009). Quantum Theory is founded on these main ideas:

i. Energy is not continuous, but comes in small but discrete units.
ii. The elementary particles behave both like particles and like waves.
iii. It is physically impossible to know both the position and the momentum of a particle at the same time. This is also known as the uncertainty principle of Heisenberg.

Unlike the case in classical physics, the nature of entities are only known upon measurement because nature is viewed to intrinsically have randomness as a property.

Two important aspects of quantum computing over the traditional classical computing are superposition and entanglement. They form a basis in understanding quantum computing operations that move away from traditional computing once these aspects come into play. They are discussed in A and B.

A. Superposition

The principle of superposition is obeyed by all waves in material media provided their amplitudes are not too great, and is rigorously obeyed by all electromagnetic waves and quantum waves according to Prabhakaran (2009). Quantum mechanics states that a wave function represents a superposition state. A thought experiment by Austrian Physicist, Erwin Schrodinger, is the Schrodinger's Cat Paradox that illustrates the concept of superposition.

According to Sidharth (2008) explains the experiment is placed in an enclosure alongside some radioactive material and cyanide. In the event that the material decays, the electron will fall into the cyanide which will be released resulting in the cat's instant death. The other possibility is that the decay will not occur and thus the cat will not be killed. A probability that the cat dies or lives exists. However, it is unknown which definite state the cat is in until the enclosure is opened making the cat both dead and alive. This is referred to as a superposition of the dead and alive states.

Opening the enclosure is synonymous to making an observation taking measurement of two superposed quantum waves. Measurement collapses the superposed wave function to a definite state whether dead or alive (Prabhakaran, 2009).

B. Entanglement

Quantum entanglement is also referred to as quantum non-separability whereby, according to (Sidharth, 2008), if two systems interact then separate, they still possess a common state vector despite the distance between them. A change in one system causes an equivalent change in the second system.

Sub-atomic particles, which are of particular interest in quantum physics, possess a common wave function in spite of distance between them. Measurement of momentum one of these particles guarantees that the momentum of the second particle is known without the need to measure it. The particles are said to be entangled. Rieffel and Polack (2011) explain however, that this special relationship between the systems is lost when measurement is done on any of the particles.

Entanglement is an important aspect taken advantage of in quantum computing so as to achieve speed up over classical algorithms. It is an aspect unique to quantum

systems and enables parallel computation in multi- qubit systems. It can only be simulated by classical systems but not as efficiently as if it were on native architecture.

C. Quantum computing

Quantum computing is a relatively young form of computing that has its origins in the 1980s and is now the subject of high-budget research by technology giants such as Google and IBM because of its foreseen potential. International Business Machines (IBM) describes quantum computing as a form of computing that is "radically different" from conventional computing. As radical as it may be viewed to be, it obeys the Turing Thesis in form of the strong Church Turing thesis that is based on a quantum Turing machine.

The building blocks of classical computing are bits while quantum computing makes use of qubits. Qubit states are manipulated by means of quantum gates just as bit values are altered via logic gates. Quantum bits are the fundamental units of information in quantum information processing in much the same way that bits are the fundamental units of information for classical processing. Various types of particles have been used to act as physical qubits such as ions, electrons, atoms and protons (Patch & Smalley, 2003) These particles spin to face either up or down which represent the binary 0 or 1 characteristic of digital information in contrast to the use of absence or presence of current as is the case in conventional computing.

Qubits are represented using vectors and ket notation showing the probability that the qubit is either a 0 or 1. In figure 2-10, a generalization of this vector is shown and how it is measured by means of a Bloch Sphere. A qubit has a continuum of possible values where a and b are complex numbers obtained by finding the square root of probabilities of the qubit value being a 0 or 1 respectively according to Rieffel and Polack (2011). A qubit can therefore possess infinitely many states since a and b can assume infinitely many values.

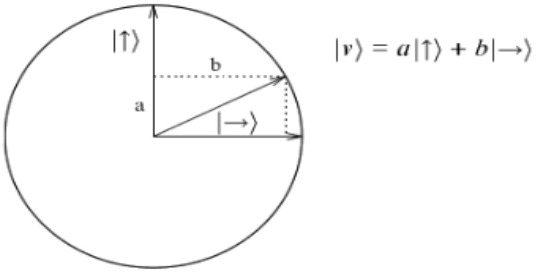


Figure 2-10: Measurement of a state vector (Source: Rieffel & Polak, 2011, p. 12)

Qubits have the ability to represent more information than the same number of classical bits. Due to the superposition principle, n qubits represent information that would be represented by 2^n bits. Memory conservation is already viewed a position of advantage of quantum systems over classical systems.

D. Quantum algorithms

Quantum algorithms make use of the ability of qubits to be in superposition or of sets of qubits to be entangled. Their efficiency is calculated in the same way as it is done for their classical counterparts making it easy to compare them.

Quantum algorithms are made up of circuits of basic quantum gates and qubits used. Operations within them are however viewed as black box functions whereby results are only known once they are executed to completion to avoid destroying entangled states or superposition that may be present in the system (Rieffel & Polak, 2011).

With respect to sorting, quantum comparison of numbers is generally achieved by most algorithms using a swap gate or a controlled swap gate (Odeh & Abdelfattah, 2016). A swap gate swaps two qubits while a controlled swap gate swaps only when the control qubit has a measure value of 1. Figure 2-11 shows a black box comparison using a controlled swap gate.

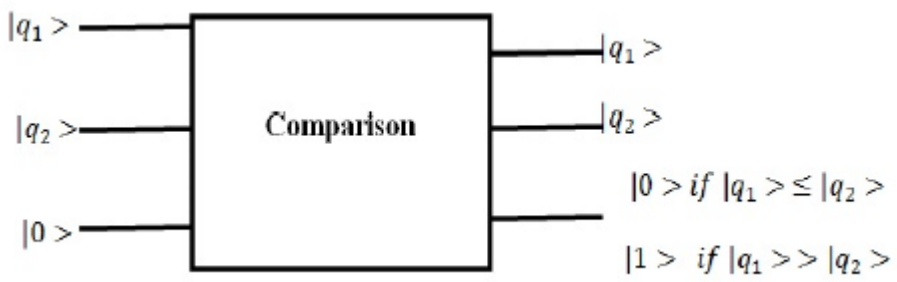


Figure 2-11: Black Box Model for Qubits Comparison (Source: Odeh & Abdelfattah, 2016, p. 2)

Grover's search algorithm is an example of a quantum algorithm that has proven to be more efficient than classical equivalents with a complexity of $O(\sqrt{N})$ according to (Tang & Su, 2014). Shor's algorithm is also considered a breakthrough in solving an NP hard problem of factoring. It is considered impossible to solve it using classical means thus making it a good choice for encryption purposes. In relation to sorting, attempts have been made in coming up with efficient quantum sorting algorithms with better complexities than best known sorts in classical computing.

Quantum sort algorithms developed by researchers over the years from as far back as 1992 to as recently as 2016 are discussed in (i) – (iv).

i. Goldstone , Gutmann and Farhi algorithm,

Farhi *et al.* introduced a quantum searching algorithm by using Insertion sort. The authors proposed algorithm used comparison gates to search in N sorted elements. The complexity for this proposed algorithm is $O(0.562\ N \log_2 N)$ (Farhi, Goldstone, & Gutmann, 2002).

ii. Høyer, Neerbek, and Shi algorithm

The main drawback of this sorting algorithm is the high cost of quantum components, and comparable complexity was achieved by the classical algorithms such as quick sort and merge sort. (Høyer, Neerbek, & Shi, 2002) Ω(N log N).

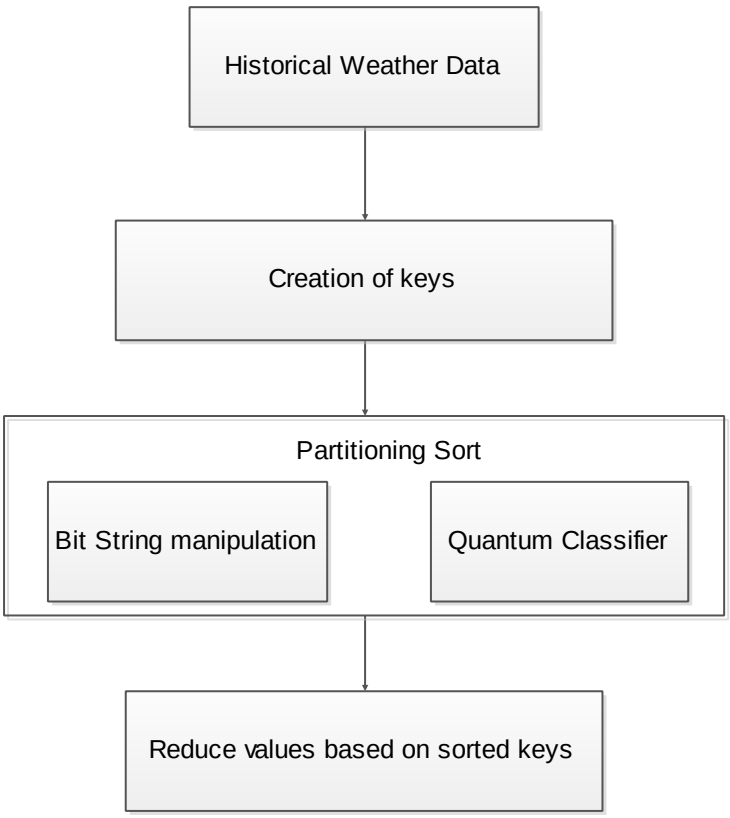
iii. Odeh, Elleithy, and Almasri algorithm

A novel quantum sort algorithm formulated by using insertion and decision tree for a set of N elements. The algorithm has a complexity of $(N-2)\log(N-2)$. The algorithm begins by reading the numbers to be sorted and converts them to binary representation. It then adds 1 to the most significant bit and adds each number to a set containing numbers having their most significant bits as 10 or 11. The 2 most significant bits are then truncated. This procedure is repeated until each number has only one bit left (Elleithy, Odeh, & Almasri, 2013).

iv. Odeh and Abdelfattah algorithm

This algorithm sorts N positive integers based on entanglement qubit property. The algorithm is similar to the algorithm in (iii) but instead divides the numbers between two sets of 00 and 11 (Odeh & Abdelfattah, 2016).

2.6 Conceptual Framework



The proposed conceptual model for an optimal algorithm is as illustrated above. It describes the input as observational weather data collected over a long period of time suitable for analysis as established in section 2.1. The proposed algorithm then creates keys to reference records within the historical weather data in a similar fashion to that of TeraSort as discussed in section 2.4 B.

Sorting these keys makes up the third stage whereby a partitioning method is used. Furthermore, the sorting procedure will consist of two parts, a classical component that operates on the keys in form of bits and a quantum component that categorizes the keys into partitions. The classical sort follows a partition sorting technique similar to that of quick sort as expounded on in section 2.4 A. The algorithm looks to tap into the power of quantum computing as explained in section 2.5.2 C.

The final step will involve reduction of the values present in the data based on sorted keys from the previous step. Reduction of data is one of the constituent phases in the TeraSort algorithm.

Chapter Three: Research Methodology

3.1 Introduction

This chapter outlines the methodology used in carrying out this research in consideration of the research objectives outlined in chapter one, the nature of this study as well as methodologies used in past related works. The research design, which guides decision variables to be considered, data collection methods as well as data analysis techniques, algorithm development tools and modules will be outlined. A time plan followed in carrying out proposed activities is also included.

3.2 Research Site

The study utilizes historical data collected in 35,000 Integrated-Surface-Database weather stations spread around the world over a period from the year 1901 to the year 2017. Some stations recorded observations hourly while others took gaps 3-hour period gaps between each observation. Parameters measured are atmospheric temperature or dew point, atmospheric pressure, atmospheric winds, precipitation among others. The map below shows the distribution of aforementioned stations across the globe.

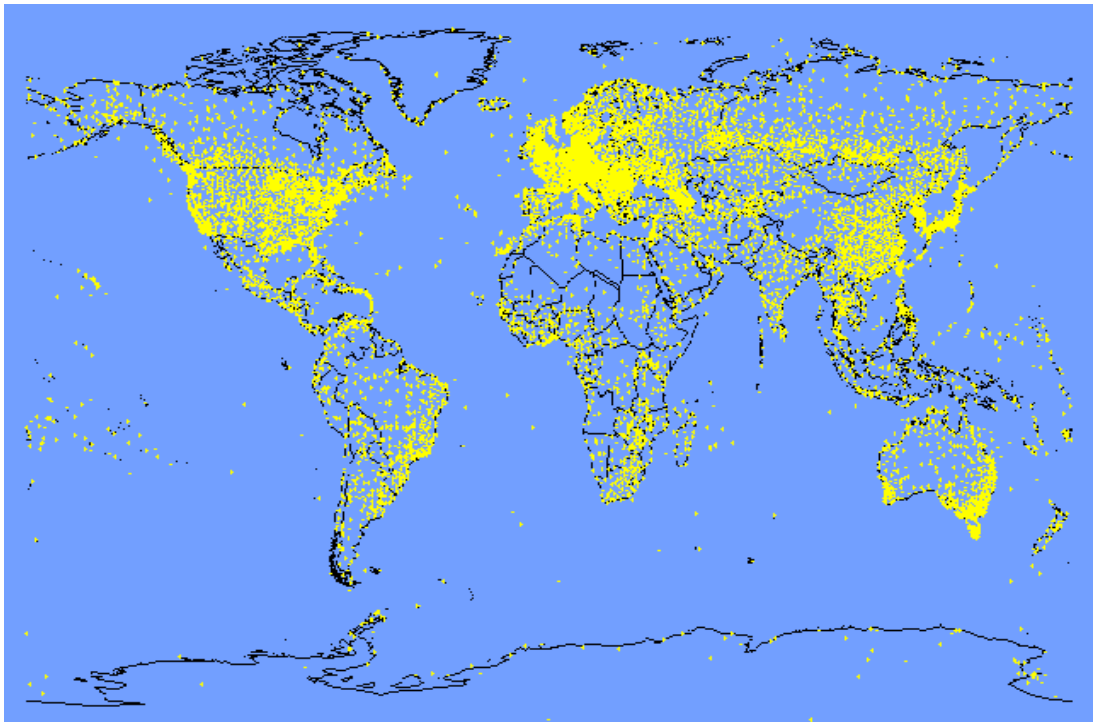


Figure 3-1: Dot Distribution Map of Integrated Surface Database Stations (Source:National Centers for Environmental Information, July 2017

3.3 Research Design

A research design describes a conceptual structure a study is to be conducted within (Kothari, 2004). It outlines means and methods for collection and analysis of data (Habib, Pathik, & Maryam, 2014). For this study, an exploratory design was used taking into account that such a design assists in shedding light on a particular subject. Other researchers will therefore gain vital information that will assist them in forming an initial hypothesis (Habib, Pathik, & Maryam, 2014).

In the current era of big data, data analytics is an area of interest for continuous improvement guaranteeing constant and incremental research on this topic. Furthermore, with the ever growing surge of hope in quantum computing due to its demonstrated benefits, this study hopes to incentivise its application in data analytics.

A quantitative approach was adopted to demonstrate the effect of introduction of a quantum component in a MapReduce data analysis framework. This was achieved by measuring the research variables described below.

3.4 Research Variables

A variable is a concept that impacts the study and can be quantified according to Kothari (2004). Variables considered in this research are:

i. Execution time – the amount of time spent by the Central Processing Unit (CPU) to carry out operations within the algorithm to their logical end.
ii. Process memory allocated – the maximum amount of memory allocated to the program inclusive of the stack, heap and swap memory.

These variables will give an indication of the efficiency of the algorithm implemented.

3.5 Data Acquisition

The algorithm proposed in this study operates on structured historical climatic data. As primary data for analysis, atmospheric conditions measured and recorded at weather stations at set intervals will be collected and used to inspire the algorithm design as well as carrying out functionality testing. Among weather parameters recorded, atmospheric temperature values will be of use in this research because they were consistently recorded in all stations across the globe.

3.6 Design and Development Approach

The preferred methodology of development was an agile approach and more specifically extreme programming style. In extreme programming, the design applied should be as simple as possible in order to satisfy the requirements for the functionality desired (Measey, 2015). Hence, the algorithm design and code was continuously reviewed during implementation. This approach allowed for flexibility in changes of design.

3.6.1 Data Management

Weather data collected is stored in form of American Standard Code for Information Interchange (ASCII) files. It is necessary to convert these files into a human readable format to facilitate in the analysis of this data. These files will be converted into an intermediate object files and finally into comma separated values (CSV) files.

3.7 Proposed System Modules

A modular design was adopted for easier development and testing of the algorithm, with each module having coarse grained functionality. Modules included are:

i. Mapper – Receives input and splits it into independent data chunks to create key, value pairs
ii. Sort – Sorts keys received from the mapper module

iii. Reducer – Combine values based on output from the sort module to give a compact set of processed output

3.8 Data Reliability and Validity

Reliability is an indicator of a measure's internal consistency. A measure is reliable when different attempts at measuring something converge on the same result while validity refers to the accuracy of a measure or the extent to which a score truthfully represents a concept (Habib, Pathik, & Maryam, 2014). Integrated surface global hourly data to be used in this research undergoes quality checks by organizations such as The US Air Force Combat Climatology Center (AFCCC), the National Climatic Data Center (NCDC), and the US Navy's Fleet Numerical Meteorological and Oceanographical Command Detachment (FNMOD). They continuously use and incrementally improve on automated data control software. Such software ensures correct data formats are used and there exists consistency between parameters and between observations.

3.9 Ethical considerations

Data collected for the purpose of this study is publically available and authorised for open use by the National Centers for Environmental Information and the National Oceanic and Administration (NOAA).

Chapter Four: Analysis and Design

4.1 Introduction

As this study proposes to come up with an optimal algorithm, it is necessary to understand key functions it should fulfil as well as the actual data it is expected to operate on. This chapter hence focuses on analysing desired algorithm requirements and characteristics of data that will be in use, therefore integrating them into a design that will guide in the implementation of the algorithm.

Analysis and design approaches will borrow from well-known, standard practices considering the environment the algorithm will be in use as well as entities expected to interact with it.

4.2 Proposed Algorithm

The design illustrates the different stages along which the algorithm accepts input data, reorganizes the data and manipulates it to give the desired result as output. The data structure and source were taken into account when constructing the algorithm structure. The structure therefore provides a solid blueprint for implementation purposes.

4.2.1 Algorithm Characteristics

A. Data Source

Defines where the input data was collected from. This data was collected in weather stations around the world known as integrated surface database stations. These stations have continuously measured atmospheric phenomena since the year 1901 till present date. Primary data captured was recorded using digital and manual means owing to the advent of the digital age in the 1970s. This data is stored in a public repository managed by the National Oceanic and Atmospheric Administration (NOAA).

B. Data

Defines the actual facts and figures collected from the data source to be manipulated by the algorithm. Each data file collected contains the station code where

the data was collected, the time it was collected in terms of the year, month, date, hour and minute it was collected. Also included are values of measured weather conditions such as temperature, precipitation, humidity, pressure and wind-observation.

C. Algorithm

Describes the flow of execution of instructions that manipulate the input data into the desired output. The algorithm can be divided into a sequence of four broad stages or steps outlined as:

Step One: Take an input data set of recorded temperature values as well as station code and time the values were recorded

Step Two: Form key, value pairs where the station code and time of collection constitute the key. This and the step above constitute the map phase.

Step Three: Sort the keys accordingly. This is the shuffle/ sort phase.

Step Four: Average temperature values of similar keys. This the reduce phase.

D. Algorithm Structure

Depicts the broad steps the algorithm will constitute as described in figure 4-1. The diagram below illustrates these steps in form of phases and expected output from each phase.

A file constituting multiple weather data records is taken in as input by the mapper once the user initiates an execution of the program. This is to say that the user accesses the program via the mapping module. Records then undergo a map phase which involves creation of keys by the mapper using specific fields within the data and their corresponding values of interest. This process results in a list of key, value pairs that have unique keys with repeated occurrences.

The list of key value pairs undergoes a sorting and shuffle phase where it is sorted with respect to the keys. The sorting module consists of both classical and quantum components where the classical component categorizes keys into partitions of larger and smaller keys based on output from the quantum component hence referred to as a quantum classifier. The sorting phase results in a list of key, value pairs sorted in ascending order.

The sorted list of key, value pairs is further taken to a reduce phase where the reducer eliminates repeat occurrences of each key by aggregating corresponding values by means of a given criterion. The outcome from this reduction is a shorter list with each unique key occurring only once. The reduced list is finally printed and saved in a file for use as required.

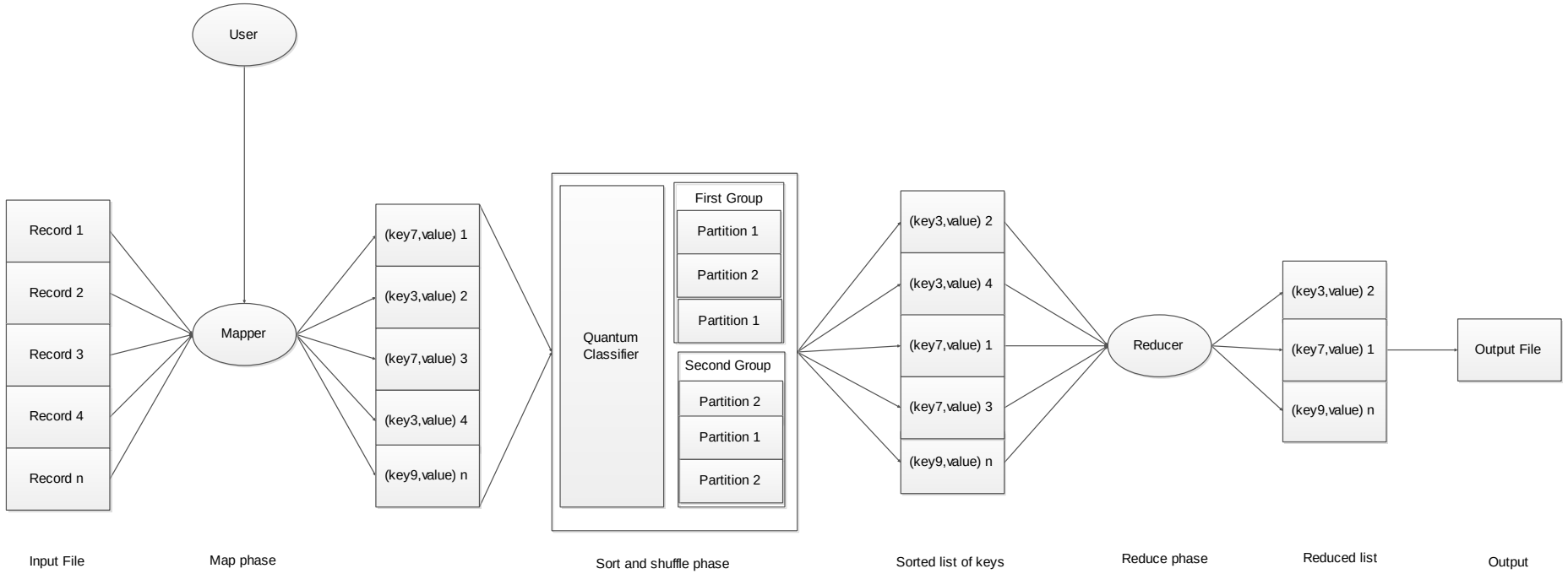


Figure 4-1 : Proposed Algorithm Structure

4.2.2 Diagrammatic Representation

To aid in understanding the proposed algorithm, the following are diagrams that illustrate the working of the algorithm from different viewpoints.

A. Use Case Diagram

This diagram illustrates the list of actions or use cases that the algorithm fulfils. The use cases identified are:

Select input dataset – the selects the data that they wish to be analysed

Initiate MapReduce job –the user initiates analysis on the selected input.

Map (key, value) pairs – the algorithm creates key, value pairs from values extracted from the data set.

Sort keys – keys are sorted according to their magnitude.

Aggregate values – values are grouped according to their respective keys

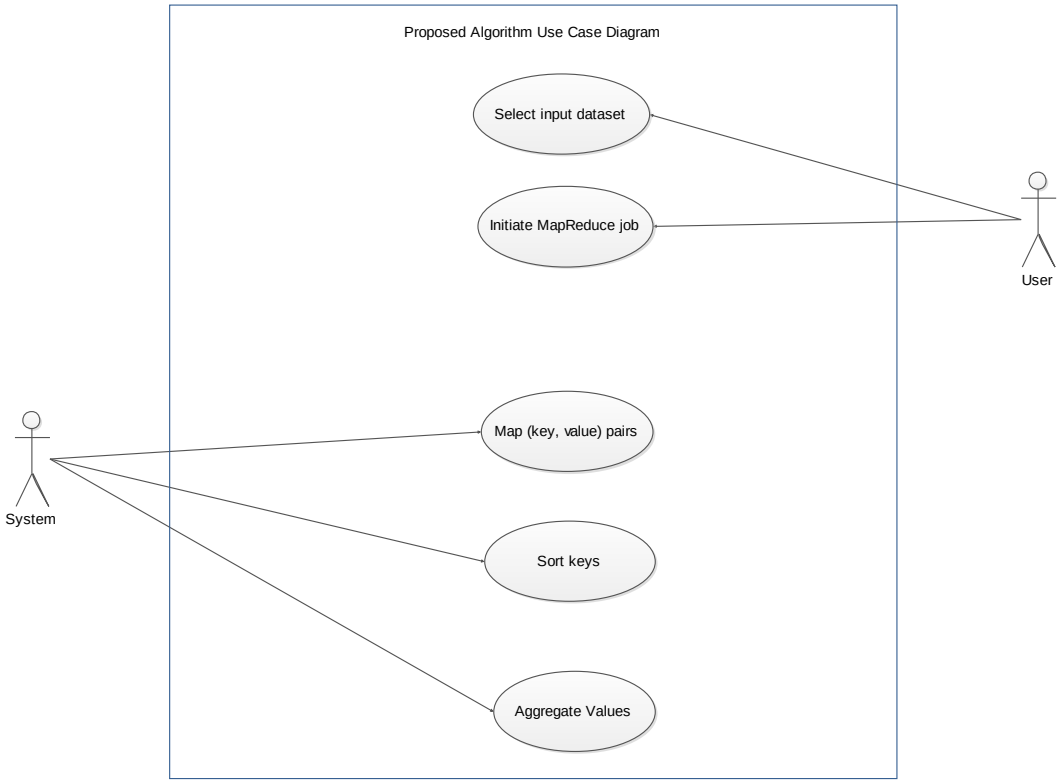


Figure 4-2: Proposed Algorithm Use Case

B. Flow Chart Diagram

A flow chart diagram, figure 4-3, describes the sequence of operations the algorithm undertakes from start to completion.

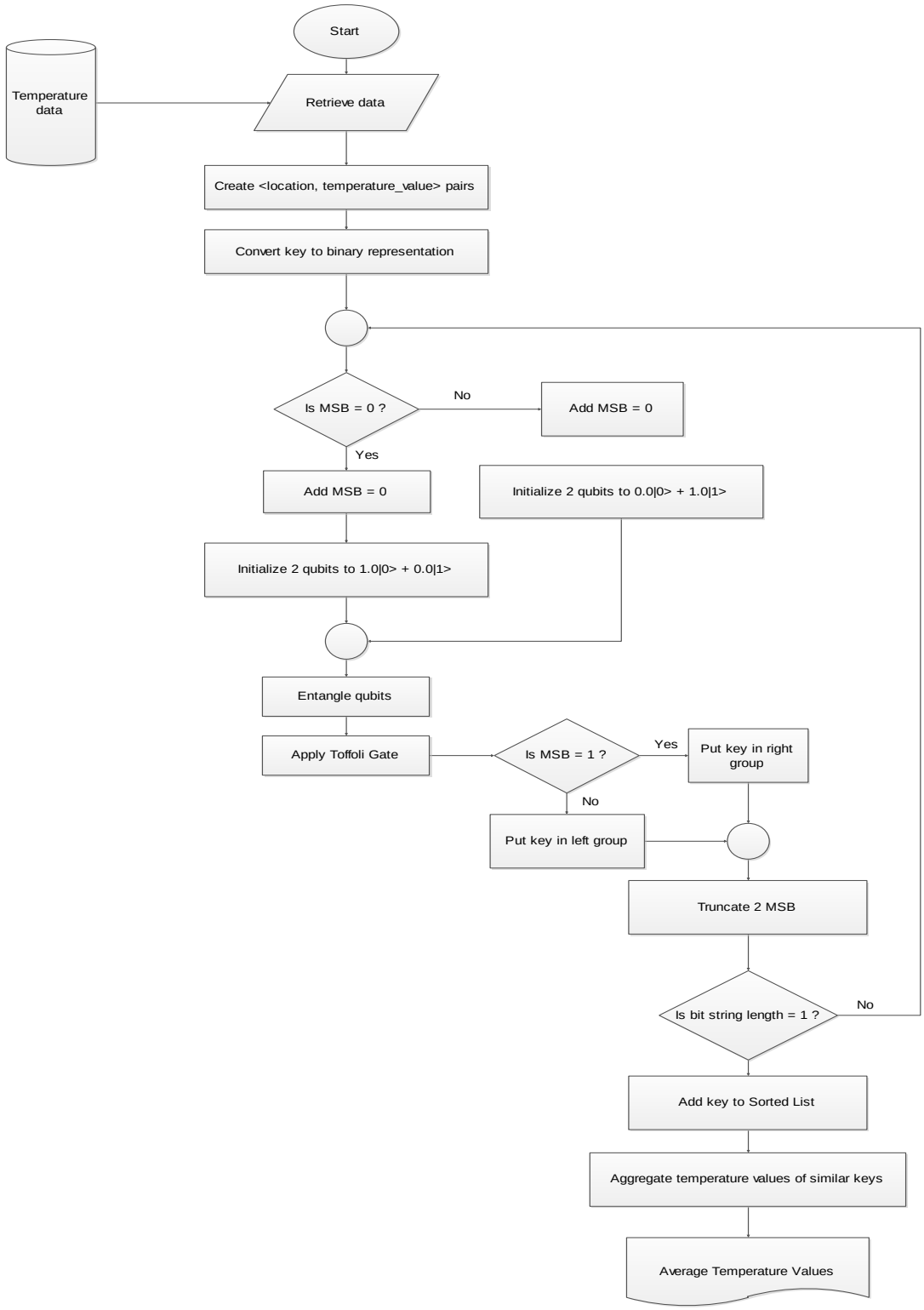


Figure 4-3: Program flowchart

C. Data Flow Diagram

A data flow diagram describes the movement of data through processes included in the algorithm. The diagram is divided in two levels, the second level, level 1 (figure 4-5), describing the first level, level 0 (figure 4-4), in greater detail.

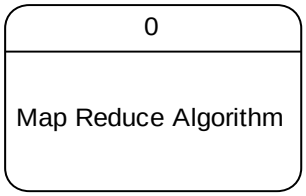


Figure 4-4: Level 0

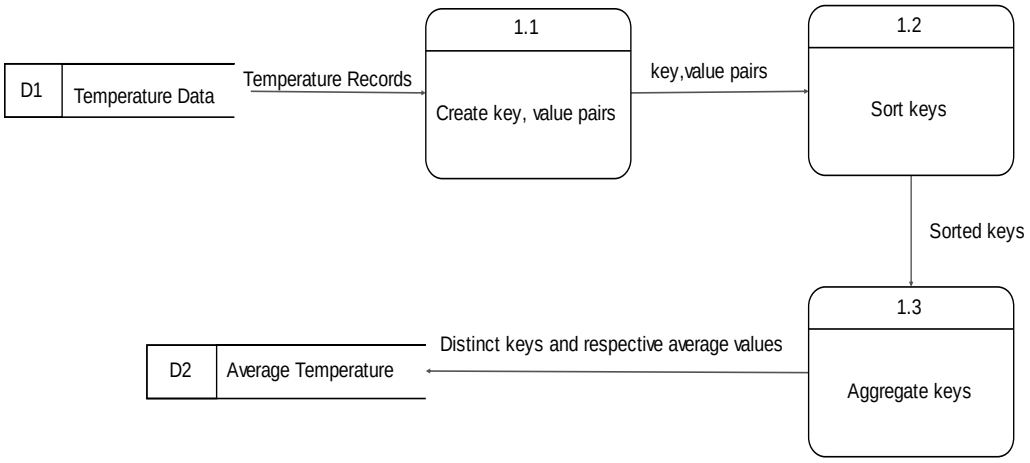


Figure 4-5: Level 1

Chapter Five: Algorithm Implementation and Validation

5.1 Introduction

Implementation of the proposed algorithm includes preparation of weather data, tools used to program the algorithm, steps taken during programming and the validation of resultant code. Data collected is converted into comma separated values which is a comprehensible format and easily consumable by the algorithm. The main components of the algorithm are mapping, sorting, and finally aggregation of weather data. Mapping involves creation of key value pairs, then the keys are sorted in the sort phase then aggregation of the values by means of the sorted keys.

Sorting is achieved by means of bit string manipulation and a partitioning method (Odeh & Abdelfattah, 2016). A quantum function facilitates the sorting phase by use of quantum bits (qubits) and application of quantum gates on them. The algorithm was implemented without any form of parallelization and on a single computing node.

5.2 Execution Environment

Defines the conditions in which the algorithm is expected to operate within. Also, noting that a component of the algorithm will be quantized, a quantum simulator will facilitate quantum computation.

5.3 Algorithm Components

5.3.1 Mapping Function

This function takes in weather data arranged as columns in comma separated value (CSV) files and creates key, value pairs from this data. The keys are then sent to the Sort function to be sorted.

5.3.2 Sorting Function

This function sorts keys received from the mapper in ascending order by converting the keys into binary representation and applying a quantum recursive partitioning function on it.

5.3.3 Aggregate Function

This function groups values according to sorted keys from the sort function. Groupings made are dependent on the columns of data selected as keys in the algorithm.

5.4 Algorithm Implementation

5.4.1 Development Tools

The algorithm was programmed using F# functional programming language. For the purpose of quantum computing simulation, Microsoft LiQUi|> simulator offers quantum bit manipulation abilities. Hence, the preferred Integrated Development Environment (IDE) is Microsoft Visual Studio Community.

5.4.2 Data Preparation

Weather data downloaded is specifically categorized as Integrated Surface Hourly Data by the National Oceanic and Atmospheric Administration (NOAA). It is stored in form of ASCII (American Standard Code for Information Interchange) files whose content is displayed as shown in figure 5-1 (See Appendix C figure 10-1).

```
0029029500999991901010120004+61483+021350FM-12+000699999V0202001N015919999999N0000001N9-00671+99999102391ADDGF105991999999999999999999
0029029500999991901010206004+61483+021350FM-12+000699999V0202501N013919999999N0000001N9-00391+99999102191ADDGF108991999999999999999999
0029029500999991901010213004+61483+021350FM-12+000699999V0202501N015919999999N0000001N9-00221+99999102261ADDGF108991999999999999999999
0029029500999991901010220004+61483+021350FM-12+000699999V0202501N015919999999N0000001N9-00171+99999102191ADDGF108991999999999999999999
0029029500999991901010306004+61483+021350FM-12+000699999V0202501N013919999999N0000001N9-00221+99999102271ADDGF108991999999999999999999
```

Figure 5-1: ASCII file content

As noted from the above data sample, this format is not human readable hence further processing was required to obtain easily comprehensible output. Each file downloaded was subjected to a java program provided alongside the ISD data within the repository, which produced an intermediate .out file as output after execution. The program converts ASCII code to recognizable character and decimal equivalents. Figure 5-2 shows the content of the .out file obtained (See Appendix C figure 10-2).

```
029500 ***** 190101011300 200  31 *** *** OVC * * *  0.0 ** ** ** ** ** ** ** ** *   14 **** 1025.9 ***** ****** *** ***
***** ***** ***** ***** **
029500 ***** 190101012000 200  36 *** *** BKN * * *  0.0 ** ** ** ** ** ** ** ** *   20 **** 1023.9 ***** ****** *** ***
***** ***** ***** ***** **
029500 ***** 190101020600 250  31 *** *** OVC * * *  0.0 ** ** ** ** ** ** ** ** *   25 **** 1021.9 ***** ****** *** ***
```

Figure 5-2 : Intermediate file content

This format is more readable than the previous ASCII format but still poses a challenge for use as input due to its lack of proper order. The data was therefore further processed into comma separated values which introduces column breaks between related groupings making it easily consumable as input. The final output is shown in figure 5-3 (See Appendix C figure 10-3).

USAF	WBAN	YR--MODA	HRMN	AW	AW	AW	W	TEMP
28060	*****	19100101	600	**	**	**	*	15
28060	*****	19100101	1300	**	**	**	*	14
28060	*****	19100101	2000	**	**	**	*	11

Figure 5-3: Final input data file content

The column header "USAF" indicates the station identification code, "YR—MODA" indicates the year, month, and date the data was collected, "HRMN" indicates the hour and minute the data was collected while "TEMP" column contains temperature values. These are the columns of interest to this study in the dataset obtained.

Data from these three columns were joined to form keys and corresponding values extracted from the "TEMP" column, representing temperature data.

5.4.3 Mapping Function

This function serves as the entry point to the program taking in the data prepared in the stages described above to form key, value pairs. Custom data types called weatherData and keyData were created to hold the input records and keys respectively. Their declaration is as shown in figure 5-4 and 5-5.

```
type weatherData =
     {stationCode : string; yearMonth : string; hourMin : string; temp: string}
```

Figure 5-4: Record Data Type

```
type keyData =
     {sCode : string; yrMonth : string ; hrMin : string}
```

Figure 5-5: Key, Value Pair Data Type

Records consisting of values extracted from the columns "USAF", "YR—MODA", "HRMN" and "TEMP" were added to a list of the type weatherData. Values from "USAF", "YR—MODA", "HRMN" were joined to form keys which were added to a list of the type key Data. Both lists store all column values except temperature values in form of the string data type. Temperature values are stored as floating point numbers.

A third list used for all subsequent processing is created using a custom data type called numberCarrier declared as illustrated in figure 5-6.

```
type numberCarrier =
     {mutable bitString : string; numberInIntegerFormat : string ; temp:float}
```

Figure 5-6: Subsequent Processing Data Type

This type stores the keys formed in binary representation, the actual keys are stored as 64- bit length strings and the temperature values as floating point values. Mapper() proceeds to call the sorting function and the aggregating function. The Mapper() function code snippet is included in Appendix D, figure 10-1.

5.4.4 Sorting Function

In this function, the keys are sorted using their binary representation by means of partitioning and application of quantum operations. It receives the processing list from Mapper() and creates two empty lists referred to as left and right. Members of the processing list are iterated through with a number of operations performed on each as follows:

i. The bit string length is checked to ascertain if it is greater than 1.
ii. If the above statement is true, a register of 3 qubits belonging to one Ket vector is created.
iii. The most significant bit (MSB) of each bit string is checked to determine whether it is a 0 or 1. If 0, bit 0 is added as the MSB and the first and second qubits are initialised to 1.0|0> + 0.0 |1>. If 1, bit 1 is added as the MSB and the first and second qubits are initialised to 0.0|0> + 1.0|1>.
iv. A circuit with the initialised quantum register and quantum function is created and compiled. A second circuit equivalent to the first is then created by aggregating unitary gates in the function to larger unitary gates. This second circuit executes faster due to fewer matrix multiplications represented by gates. The quantum function is then called by means of this second circuit, which transforms the state of qubits passed in the register.
v. The most significant qubit is checked to determine whether it is a 0 or 1. If 0, add the key to the left list and if 1, add it to the right list.
vi. The 2 most significant bits are truncated from the key bit string value.

vii. If the bit string length as checked in step (i), is equal to 1, then no further processing is done to the key and is hence added to the final list of keys.

Once the processing list is traversed to completion, the sorting function recursively calls itself passing the left list then the right list as long as either of the two lists contain items. The sorting function code snippet is included in Appendix D, figure 11-2.

5.4.5 Quantum Function

The 3 qubit register initialised in the sorting function is received as input to this function. The qubits are entangled and transformed by means of Toffoli or Controlled Controlled NOT (CCNOT) gate then finally measures the qubits by means of a Measure gate.

The Toffoli gate is a reversible gate that takes in 3 qubits as input and produces 3 qubits as output. The truth table in figure 5-4 describes its working:

T_3 = CC-NOT	⟨000⟩	...					⟨110⟩	⟨111⟩
⟨000⟩	1	0	0	0	0	0	0	0
⟨001⟩	0	1	0	0	0	0	0	0
⟨010⟩	0	0	1	0	0	0	0	0
⟨011⟩	0	0	0	1	0	0	0	0
⟨100⟩	0	0	0	0	1	0	0	0
⟨101⟩	0	0	0	0	0	1	0	0
⟨110⟩	0	0	0	0	0	0	0	1
⟨111⟩	0	0	0	0	0	0	1	0

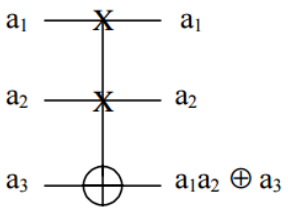


Figure 5-7 : Toffoli Gate

The quantum function code snippet is included in Appendix D, figure 10-3.

5.4.6 Reducing Function

After the sorting function terminates, Mapper() calls Reducer(), which reduces the sorted keys to monthly average temperature values. The function iterates through the sorted list, totalling the temperature values of repeated keys and taking their average. Tracking of repeated keys is done by means of an accumulator variable which is added

by 1 every time a key is repeated. This is simply achieved because the list of keys is sorted. The quantum function code snippet is included in Appendix D, figure 10-4.

5.4.7 Program Output

Once the functions described above execute successfully, output arrived at is displayed in figure 5-5:

Month	Average Temperature
Month 1	29.73
Month 2	18.60
Month 3	22.74
Month 4	35.08
Month 5	43.69
Month 6	57.41
Month 7	63.72
Month 8	63.30
Month 9	51.50
Month 10	49.01
Month 11	32.39
Month 12	35.42

Figure 5-8 : Output file

This specific output shows monthly average temperature values derived from 1095 records taken in one station over a duration of one year. This exhibits the reduction ability the program possesses.

5.4.8 Asymptotic Analysis

To give a true picture of how the implemented algorithm performs, asymptotic analysis of each function is carried out and summed up to give the total expected performance. The analysis has been made using Big- O notation.

The mapping function carries out the following operations:

i. Three initialization operations and one file-fetching operation which can be represented as O(4)

ii. To create the key value pairs, at least 9 operations are carried out for each record in the input dataset within a loop. This is represented as O(9N) where N is the input dataset size.

The asymptotic function can therefore be given as $O(4) + O(9N)$. Dropping the negligible constant complexity yields a minimum complexity for the whole mapping function as described in equation 5.1

Equation 5-1: Mapping Function Complexity

$$O(9N)$$

Operations carried out by the sorting function are:

i. The sorting function carries out at least 3 initialization operations This represents a complexity of O(3).
ii. The function iterates though the given set of keys. Each iteration branches to one of two possible branches of computation with a minimum of 6 operations being carried out in each branch. Therefore the complexity for either branch is O($\log_2 6N$).
iii. The function is called recursively until all key bit strings are reduced to one bit. Since the algorithm uses a 64 bit register, it is recursively executed at most 63 times for each key item in the dataset. The recursion halts when the maximum number of keys in each list is equal to 2. Hence the complexity translates to O ($63N \log_2 \frac{6N}{2}$)

Therefore the complexity function to perform sorting is $O(3) + O(63N \log_2 \frac{6N}{2})$. Dropping the negligible constant complexity yields the complexity shown in equation 5-2.

Equation 5-2 : Sorting Function Complexity

$$O(3) + O(63N \log_2 \frac{6N}{2})$$

The quantum function carries out 3 operations and is called by the sort function 63N-2 times and hence has a complexity of O(3(63N-2)). Dropping constants present in the asymptotic function yields the complexity in equation 5-3.

Equation 5-3: Quantum Function Complexity

$$O(3(63N))$$

The reducing function carries out a minimum 3 initialization operations represented as O(3). It then iterates through the sorted dataset performing a specific operation for the first operation and a minimum 3 operations for the remaining keys in the dataset giving the complexity O(1) + O(3(N-1)). Dropping constants within the function yields the complexity of the reducer as described by equation 5-4.

Equation 5-4: Reducing Function Complexity

$$O(3N)$$

Aggregating the individual complexities per function gives:

$$\mathrm{O}(9\mathrm{N}) + O\left(63N \log_2 \frac{6N}{2}\right) + \mathrm{O}\big(3(63\mathrm{N})\big) + \ \mathrm{O}(3(\mathrm{N}))$$

Dropping constant complexities and multipliers present in the function results in equation 5-5.

Equation 5-5: Aggregate Complexity

$$O(N) + O\left(N \log_2 \frac{N}{2}\right) + O(N) + O(N)$$

Taking the greatest value of N above, $O(N)$ is obtained as the overall complexity of the program. This means that the performance of this program is directly proportional to the size of given input or in other words, the program has a linear complexity.

5.4.9 Testing

The program was tested using a varying number of records to view its performance with a growth in the number of input records to be operated on. Sets of 1000 records, 5000 records, 10,000 records, 100,000 records and 200,000 records

were created from the weather data collected and used as test data sets. It is of importance to note that the absolute measures recorded vary depending on multiple factors including the machine architecture, operating system state among others. They however were recorded with the aim of establishing the general behaviour of the algorithm with an increase in input.

The program was executed ten times with each given set of temperature records. The actual time taken by the Central Processing Unit (CPU) to run the program to completion was recorded as well as the maximum amount of memory allocated to the process. This memory is inclusive of stack, heap and swap memory taken up by the process.

The average values for both execution time and memory usage were calculated for each test set. The average values act as representative performance values for the each given test data set.

A. Input Size -1,000 Records

Recordings made for an input data set of 1000 records are shown in table 5-1:

Table 5-1: Test output for 1000 records

Test Number	CPU Execution Time (ms)	Maximum Memory Used (MB)
1	8.97	112.5
2	9.14	114.7
3	9.03	114.8
4	9.35	132.6
5	9.89	113.6
6	9.05	116.8
7	8.95	117.1
8	9.46	132.4
9	9.39	137.1
10	5.09	131.3
Average	8.83	122.29

B. Input Size -5,000 Records

Recordings made for an input data set of 5000 records are shown in table 5-2:

Table 5-2: Test output for 5,000 records

Test Number	Execution Time (ms)	Memory Used (MB)
1	46.85	134.3
2	46.57	136.3
3	46.05	139.5
4	45.91	140.5
5	45.72	135.3
6	46.06	136.4
7	45.03	138
8	49.08	137.2
9	48.63	139.3
10	48.48	133.5
Average	46.83	137.03

C. Input Size -10,000 Records

Recordings made for an input data set of 10,000 records are shown in table 5-3:

Table 5-3: Test output for 10,000 records

Test Number	Execution Time (ms)	Memory Used (MB)
1	114075	138.6
2	101001	144.3
3	95039	139.5
4	102002	141.5
5	86025	140.7
6	71071	139.2
7	105082	144.7
8	93023	136.1
9	99017	134.3
10	111018	143.8
Average	97735.3	140.27

D. Input Size -100,000 Records

Recordings made for an input data set of 100,000 records are shown in table 5-4:

Table 5-4: Test output for 100,000 records

Test Number	Execution Time (ms)	Memory Used (MB)
1	723038	315.6
2	721021	330.4
3	583017	346.9
4	553014	313.2
5	713011	315.4
6	603016	318.4
7	611031	320.8
8	693025	322.9
9	596014	316.4
10	703281	344.4
Average	649946.8	324.44

E. Input Size -200,000 Records

Recordings made for an input data set of 200,000 records are shown in table 5-5:

Table 5-5: Test output for 200,000 records

Test Number	Execution Time (ms)	Memory Used (MB)
1	1111075	376.4
2	1195016	366.2
3	1033041	340.2
4	1122019	377.8
5	1216020	418.3
6	1151012	365.5
7	1094012	359.3
8	1216015	345.2
9	1092076	388.5
10	1173051	405.1
Average	1140333.7	374.25

F. Graphical Interpretation of Test Recordings

Plotting the average values of CPU execution time in each data set against the corresponding input size realizes the graph 5-9:

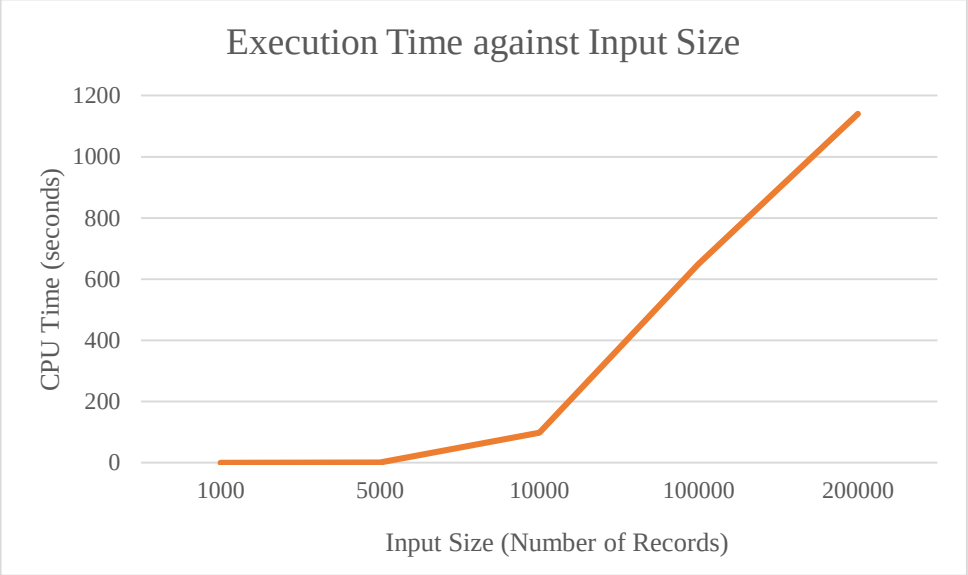


Figure 5-9 : Graph of Execution Time against Input Size

The graph illustrates that a substantial increase in the number of input records corresponds to an increase in execution time. This is to say that the execution time is directly proportional to the size of input. The linear relationship described is more apparent when large datasets are compared.

Plotting the average values of memory usage in each data set against the corresponding input size realizes the graph 5-10.

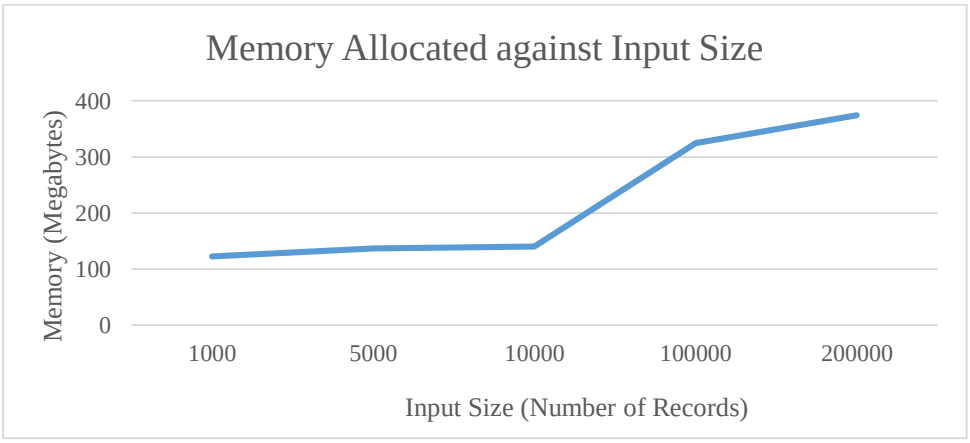


Figure 5-10: Graph of Memory allocated against Input Size

Similarly, an increase in the record size translates to an increase in memory allocated as it was noted in the case of CPU execution time. However, the increase in memory is pronounced for large increments in the size of input. The above findings prove the asymptotic analysis made to be true for both program execution time as well as process memory allocation.

Chapter Six: Conclusion

6.1 Introduction

Data analytics is an integral part in today's world where information is regarded as an essential asset. This is particularly true in climatology whereby analysis is required to make sense out of large amounts of historical data. In this way, it is possible to make weather forecasts and detect changes in climate patterns. Computing lends a hand in this analysis, broadening the capability to unearth a great amount of information. It is hence imperative to streamline computing techniques applied in data analytics. Use of algorithms and frameworks that can efficiently handle large datasets ensures good use of computing power available. The design and implementation of the proposed algorithm in this research aimed at accomplishing the much needed efficiency within the MapReduce framework in comparison to other established algorithms such as TeraSort.

Based on the conceptual framework arrived at in chapter two, the proposed algorithm was designed, implemented and tested, noting key findings instrumental in gauging the usability of this algorithm. This chapter reviews findings made in the course of this study in relation to the objectives set in the first chapter, gives recommendations on improvements that may be made on the proposed algorithm and suggests areas for further research.

6.2 Weather data used in climate analysis and modelling

The first objective was to investigate what data is required for climate analysis and modelling. In the literature review carried out, historical weather data was found to be essential in climate analysis techniques such as trend analysis and climate modelling for the purpose of calibration of models. Since climate can only be determined from weather data collected over lengthy periods of time, historical weather data is essential in climate studies.

6.3 Challenges associated with analysis of large weather datasets

Investigating challenges associated with the analysis of large weather datasets was listed as the second objective. Challenges noted during literature review are computing – intensive analysis carried out analysing large datasets especially for the use in areas such as numerical weather prediction.

6.4 Existing Algorithms for analysis of large weather datasets

The third objective for this study was to review existing algorithms used particularly in analysing large datasets. Sorting algorithms were of particular importance since they are integral in analytical procedures. Various algorithms such as quick sort and merge sort were analysed, using asymptotic analysis within the literature review. Algorithms such as merge sort were seen to have the best time complexity of $O(N \log_2 N)$, where N is the size of input, hence popularly used to sort large data sets. TeraSort algorithm, used in MapReduce benchmarking, was reviewed and to be established as a form of radix sort which has a complexity of $O(Nw)$ where N is the input size and w the average key length.

6.5 Sorting Algorithm for Large datasets

Implementing an optimal sorting algorithm for the analysis of large datasets was developed in a MapReduce fashion with the help of quantum computing. This was achieved through a design that took into account the type of data to be operated on as well as the framework it was to be applied in. The algorithm proposed materialised into a program consisting four interrelated functions that produce the desired output.

6.6 Sorting Algorithm Testing

The final objective of this study was to test the efficacy of the developed program with respect to its performance in a MapReduce framework. Input test data samples varying in size were subjected to the program to measure the performance with respect to CPU time execution and process memory allocation. Results obtained from the tests revealed a linear relationship between the two variables and the size of the input dataset. This confirms the asymptotic analysis carried out on the implemented algorithm which arrived at a complexity of $O(N)$.

6.7 Strengths of Developed Algorithm over TeraSort

TeraSort is developed using tries which is similar to radix sort that has a complexity of $O(Nw)$ where N is the input size of keys to be sorted and w the length of each key. The sorting function as analysed in chapter five has a complexity of $O\left(N \log_2 \frac{N}{2}\right)$ which is more desirable than that of radix sort. Furthermore, the developed algorithm has the potential to take advantage of quantum parallelism.

6.8 Limitations of the Developed Algorithm

The developed algorithm is designed synchronously to run on one node as compared to TeraSort which is asynchronously runs on a multiple nodes. Also, the developed algorithm was run on a simulator due to lack of supporting hardware, capping the performance to that which classical computing can offer.

6.9 Recommendations

The study proposes the following recommendations based on findings made:

i. Implement the algorithm to support distributed processing as required in a real MapReduce environment. The load divided by will be divided by k nodes in the network. Supposing the network overhead is denoted by c, the complexity can be estimated to be $O(\frac{N+c}{k})$.

ii. Parallelize the algorithm to execute asynchronously which gives a significant improvement in performance for very large input datasets.

iii. Use Quantum Phase Registers (Emami, Fattahi, & Keshmiri, 2008) which store data in terms of data state, data phase and data probability. This introduces quantum parallelism in sorting four keys in one operation with a length of two bits giving a 25 % chance that each key is correctly sorted. The smaller the register the higher the probability of occurrence of each number.

iv. In the event that a greater number of quantum operations are required, optimize the circuit containing the gates to be applied by combining the unitary matrices. Consequently, the algorithm will perform better if it carries out larger but fewer matrix multiplications.

6.10 Suggestions for Future Research

Quantum computing is an area of research that promises great potential in providing powerful mechanisms to parallelise computing. The study suggests that a further research may be conducted to parallelise the developed program using quantum constructs.

References

Akthar, N., Ahamad, M. V., & Ahmad, S. (2016). MapReduce Model of Improved K-Means Clustering Algorithm Using Hadoop MapReduce. *2016 Second International Conference on Computational Intelligence & Communication Technology (CICT)* (pp. 192-198). Ghaziabad: IEEE.

Balaji, V. (2015). Climate Computing: The State of Play. *Computing in Science & Engineering*, 9-13.

Baldwin, D., & Scragg, G. (2004). *Algorithms and Data Structures : The Science of Computing.* Hingham, US: Charles River Media / Cengage Learning.

Barak, B., & Arora, S. (2007). *Computational Complexity: A Modern Approach.*

Bhardwaj, A., Singh, V. K., & Vanraj. (2015). Analyzing BigData with Hadoop cluster in HDInsight azure Cloud. *2015 Annual IEEE India Conference (INDICON)* (pp. 1-5). New Delhi: IEEE.

Board on Atmospheric Sciences and Climate Staff. (2001). *Climate Services Vision : First Steps Toward the Future.* Washington,US: National Academies Press.

Coiffier, J. (2011). *Fundamentals of Numerical Weather Prediction.* UK: Cambridge University Press.

Cormen, T. H., Leiserson, C. E., & Rivest, R. L. (2009). *Introduction to Algorithms (3).* Cambridge, US: The MIT Press.

Cormen, T., & Balkcom, D. (2016). *Overview of quicksort*. Retrieved from Khan Academy: https://www.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/overview-of-quicksort

Du, D.-Z., & Ko, K.-I. (2014). *Wiley Series in Discrete Mathematics and Optimization : Theory of Computational Complexity (2).* Somerset, US: Wiley.

Edwards, P. N. (2010). *Infrastructures : A Vast Machine : Computer Models, Climate Data, and the Politics of Global Warming.* Cambridge,US: The MIT Press.

Elleithy, K., Odeh, A., & Almasri, M. (2013). Sorting N elements using quantum entanglement sets. *Innovative Computing Technology (INTECH), 2013 Third International Conference on* (pp. 213-216). London: IEEE.

Emami, M. S., Fattahi, M. R., & Keshmiri, H. (2008). Comparison between New Quantum Approaches for Finding the Minimum or the Maximum of an Unsorted Set of Integers. *Seventh IEEE/ACIS International Conference on Computer and Information Science (icis 2008).* IEEE.

Farhi, E., Goldstone, J., & Gutmann, S. (2002). Quantum Computation by Adiabatic Evolution. *Algorithmica*, 429-449.

Graham, S. L., Snir, M., & Patterson, C. A. (2005). *Getting Up to Speed : The Future of Supercomputing.* Washington, US: National Academies Press.

Grandinetti, L., Joubert, G. R., & Kunze, M. (2015). *Advances in Parallel Computing : Big Data and High Performance Computing.* Fairfax,US: IOS Press.

Groner, L. (2014). *Learning JavaScript Data Structures and Algorithms.* Packt Publishing.

Gupta, A., Mehrotra, A., & Khan, P. M. (2015). Challenges of Cloud Computing & Big Data Analytics. *Computing for Sustainable Global Development (INDIACom), 2015 2nd International Conference on* (pp. 1112-1115). New Dehli: IEEE.

Guttorp, P. (2014). Statistics and Climate. *Statistics and Its application*, 87-101.

Habib, M. M., Pathik, B. B., & Maryam, H. (2014). *Research Methodology - Contemporary Practices : Guidelines for Academic Researchers (1).* Newcastle-upon-Tyne, UNITED KINGDOM: Cambridge Scholars Publishing.

Hanuliak, P., & Hanuliak, M. (2014). *Analytical Modelling in Parallel and Distributed Computing.* Oxford, GB: Chandos Books Oxford.

Harper, K. C. (2012). *Weather by the Numbers : The Genesis of Modern Meteorology.* Cambridge, MA: The MIT Press.

Hashem, H., & Ranc, D. (2016). Extending Standard MapReduce Algorithms. *2016 IEEE Second International Conference on Multimedia Big Data* (pp. 159-165). Taipei: IEEE.

Hirvensalo, M. (2013). Quantum Computing. In A. L. Runehov, & L. Oviedo, *Encyclopedia of Sciences and Religions* (pp. 1922-1926). Dordrecht: Springer Netherlands.

Høyer, P., Neerbek, J., & Shi, Y. (2002). Quantum complexities of ordered searching, sorting, and element distinctness. *Algorithmica*, 429-448.

Iafrate, F. (2015). *From Big Data to Smart Data (1).* Somerset, US: Wiley-ISTE.

IBM. (2016). *IBM Quantum Computing*. Retrieved from IBM: https://www.research.ibm.com/quantum/expertise.html

Integrated Surface Database (ISD). (2014). Retrieved from National Centers for Environmental Information: https://www.ncdc.noaa.gov/isd

Jararweh, Y., Alsmadi, I., & Al-Ayyoub, M. (2014). The analysis of large-scale climate data: Jordan case study. *Computer Systems and Applications (AICCSA), 2014 IEEE/ACS 11th International Conference*, (pp. 288-294). Doha.

Joussaume, S. (2012). Modelling the Earth's climate system: data and computing challenges. *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:* (pp. 2325 - 2356). Salt Lake City, UT: IEEE.

Kaye, P., Laflamme, R., & Mosca, M. (2007). *An Introduction to Quantum Computing.* Oxford, GB: OUP Oxford.

Kothari, C. R. (2004). *Research Methodology : Methods and Techniques (2).* Daryaganj, IN: New Age International.

Lautenschlager, M., Adamidis, P., & Kuhn, M. (2015). Big Data Research at DKRZ – Climate Model Data Production Workflow. In L. Grandinetti, G. R. Joubert, & M. Kunze, *Advances in Parallel Computing : Big Data and High Performance Computing* (p. 168). Hamburg: IOS Press.

McCool, M., Reinders, J., & Robison, A. (2012). *Structured Parallel Programming.* Elsevier Science.

McGuffie , K., & Henderson-Sellers, A. (2014). *Climate Modelling Primer (4).* Somerset, GB: Wiley-Blackwell.

McGuffie, K., & Henderson-Sellers, A. (2001). Forty Years of Numerical Climate Modelling. *International Journal of Climatology*, 1067–1109.

Measey, P. (2015). *Agile Foundations.* BCS Learning & Development Limited.

National Aeronautics and Space Administration. (2015, July 31). *What's the Difference Between Weather and Climate?* Retrieved from NASA: http://www.nasa.gov/mission_pages/noaa-n/climate/climate_weather.html

National Energy Resources Australia. (n.d.). *How will changes in climate affect our lives?* Retrieved from National Centre for Atmospheric Science: https://www.ncas.ac.uk/index.php/en/how-will-changes-in-climate-affect-us

National Research Council Staff; Commission on Geosciences; Environment and Resources; Division on Earth and Life Studies. (1999). *Capacity of U. S. Climate Modeling Efforts to Support Climate Change Assessment Activities.* Washington, US: National Academies Press.

Odeh, A., & Abdelfattah, E. (2016). Quantum sort algorithm based on entanglement qubits {00, 11}. *Long Island Systems, Applications and Technology Conference (LISAT), 2016 IEEE* (pp. 1-5). Farmingdale, NY: IEEE.

Patch, K., & Smalley, E. (2003). *Quantum Computing : Prospects and Pitfalls.* Boston, US: Technology Research News.

Prabhakaran, S. (2009). *Quantum Mechanics (1).* Jaipur, IN: Book Enclave.

Rieffel, E. G., & Polak, W. H. (2011). *Scientific and Engineering Computation : Quantum Computing : A Gentle Introduction.* Cambridge, US: The MIT Press.

Schnas, J. L., Le, T. J., Mattman, C. A., Lynne, C. S., Cinquini, L., Ramirez, P. M., & Hart, A. F. (2016, September 16). Big Data Challenges in Climate Science:

Improving the next-generation cyberinfrastructure. *IEEE Geoscience and Remote Sensing Magazine*, pp. 10-22.

Shaffer, C. A. (April 16, 2009). *A Practical Introduction to Data Structures and Algorithm Analysis Third Edition (Java).* Department of Computer Science, Virginia Tech, Blacksburg, VA 24061: Prentice Hall.

Sidharth, B. (2008). *Thermodynamic Universe : Exploring the Limits of Physics.* Singapore, US: WSPC.

Slagter, K., Hsu, C.-H., Chung, Y.-C., & Zhang, D. (2013). An improved partitioning mechanism for optimizing massive data analysis using MapReduce. *Journal of Supercomputing*, 539-555.

Song, J., Xu, S., Zhang, L., Pahl, C., & Yu, G. (2015). Performance and Energy Optimization of the Terasort Algorithm by Task Self-Resizing. *Information Technology and Control*, 1-12.

Spellman, F. R. (2012). *The Handbook of Meteorology.* Scarecrow Press.

Stephens, R. (2013). *Essential Algorithms : A Practical Approach to Computer Algorithms (1).* Somerset, US: Wiley.

Tang, Y., & Su, S. (2014). Application of Grover's Quantum Search Algorithm to Solve the Transcendental Logarithm Problem. *10th International Conference on Computational Intelligence and Security*, 445-449.

Tannir, K. (2014). *Optimizing Hadoop for MapReduce.* Olton:GB: Packt Publishing.

Thomas, A., & Herzfeld, U. C. (2004). REGEOTOP: New climatic data fields for East Asia based on localized relief information and geostatistical methods. *International Journal of Climatology*, 1283-1306.

Vikram, M., & Sudhakar, N. (2016). A Novel Algorithm to Improve Performance in Mapreduce Environment using Join operations. *International Conference on Information Communication and Embedded System.*

von Storch, H., & Zwiers, F. W. (2003). *Statistical Analysis in Climate Research.* Melbourne, Australia: Cambridge University Press.

Wang, B., Jiang, J., Wu, Y., Yang, G., & Li, K. (2016). Accelerating MapReduce on Commodity Clusters: An SSD-Empowered Approach. *IEEE Transactions on Big Data*, 1-13.

Warner, T. T. (2011). *Numerical Weather and Climate Prediction.* Cambridge,UK: Cambridge University Press.

Weart, S. R. (2008). *The Discovery of Global Warming.* Havard University Press.

Wecker, D., & Svore, K. M. (2014, February 18). *LIQUi|>: A Software Design Architecture and Domain-Specific Language for Quantum Computing.* Retrieved from https://arxiv.org/abs/1402.4467

Wichert, A. (2013). *Principles of Quantum Artificial Intelligence.* Singapore, US: WSPC.

Wilf, H. S. (2002). *Algorithms and Complexity (2).* Natick, US: A K Peters/CRC Press.

Zwieflhofer, W., & Mozdzynski, G. (2005). *Use Of High Performance Computing In Meteorology - Proceedings Of The Eleventh Ecmwf Workshop.* River Edge, US: World Scientific.

Appendix A: Time Plan

Table 8-1 A: Activity Schedule

Progress Stage	Stage Description	Proposed Date
1	Scoping of the research study	September 1st 2016- September 15th - 2016
2	Choice of the research topic	September 16th 2016 – September 23rd 2016
3	Research problem clarification, research objectives, purpose and significance	September 24th 2016 – October 3rd 2016
4	Foundation of literature review	October 4th 2016 – October 15th 2016
5	Proposal of research methodology	October 16th 2016- October 21st 2016
6	Advanced literature review	October 22nd 2016-November 10th 2016

Table 8-1 B: Activity Schedule

Progress Stage	Stage Description	Proposed Date
7	Detailed proposal of research methodology	November 12th 2016 – November 21st 2016
8	Proposal defence	November 28th 2016 -November 30th 2016
9	Data collection	December 1st 2016- December 14th 2016
10	Data analysis and interpretation	December 15th 2016 – December 24th 2016
11	Research report writing – first draft	December 26th 2016– February 20th 2017
12	Final draft of research report	February 22nd 2017 – March 10th 2017
13	Submission of research for examination	March 25th 2017 –March 29th 2017

Appendix B: Research Activity Gantt chart

Figure 8-1: Gantt chart

Appendix C: Raw Data

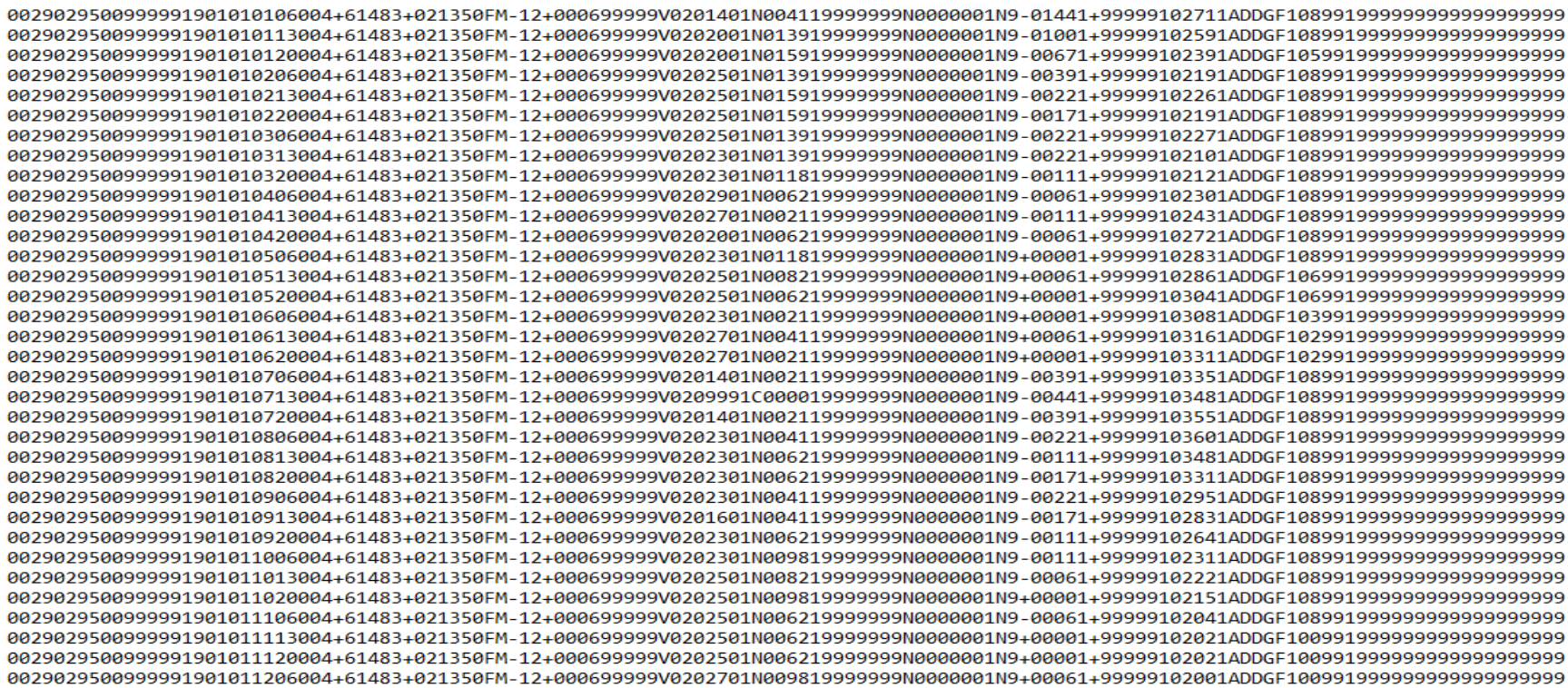


Figure 9-1 : ASCII data file

Data retrieved from NOAA repository was in ASCII format as shown in figure 9-1.

```
  USAF  WBAN YR--MODAHRMN DIR SPD GUS CLG SKC L M H  VSB MW MW MW MW AW AW AW AW W TEMP DEWP    SLP   ALT    STP MAX MIN
PCP01 PCP06 PCP24 PCPXX SD
029500 ***** 190101010600 140   9 *** *** OVC * * *  0.0 ** ** ** ** ** ** ** ** *    6 **** 1027.1 ***** ****** *** ***
***** ***** ***** ***** **
029500 ***** 190101011300 200  31 *** *** OVC * * *  0.0 ** ** ** ** ** ** ** ** *   14 **** 1025.9 ***** ****** *** ***
***** ***** ***** ***** **
029500 ***** 190101012000 200  36 *** *** BKN * * *  0.0 ** ** ** ** ** ** ** ** *   20 **** 1023.9 ***** ****** *** ***
***** ***** ***** ***** **
029500 ***** 190101020600 250  31 *** *** OVC * * *  0.0 ** ** ** ** ** ** ** ** *   25 **** 1021.9 ***** ****** *** ***
***** ***** ***** ***** **
029500 ***** 190101021300 250  36 *** *** OVC * * *  0.0 ** ** ** ** ** ** ** ** *   28 **** 1022.6 ***** ****** *** ***
***** ***** ***** ***** **
029500 ***** 190101022000 250  36 *** *** OVC * * *  0.0 ** ** ** ** ** ** ** ** *   29 **** 1021.9 ***** ****** *** ***
***** ***** ***** ***** **
029500 ***** 190101030600 250  31 *** *** OVC * * *  0.0 ** ** ** ** ** ** ** ** *   28 **** 1022.7 ***** ****** *** ***
***** ***** ***** ***** **
029500 ***** 190101031300 230  31 *** *** OVC * * *  0.0 ** ** ** ** ** ** ** ** *   28 **** 1021.0 ***** ****** *** ***
***** ***** ***** ***** **
029500 ***** 190101032000 230  26 *** *** OVC * * *  0.0 ** ** ** ** ** ** ** ** *   30 **** 1021.2 ***** ****** *** ***
***** ***** ***** ***** **
029500 ***** 190101040600 290  14 *** *** OVC * * *  0.0 ** ** ** ** ** ** ** ** *   31 **** 1023.0 ***** ****** *** ***
***** ***** ***** ***** **
029500 ***** 190101041300 270   5 *** *** OVC * * *  0.0 ** ** ** ** ** ** ** ** *   30 **** 1024.3 ***** ****** *** ***
***** ***** ***** ***** **
029500 ***** 190101042000 200  14 *** *** OVC * * *  0.0 ** ** ** ** ** ** ** ** *   31 **** 1027.2 ***** ****** *** ***
***** ***** ***** ***** **
```

Figure 9-2 : Intermediate data file

Gathered data in ASCII format is converted into human readable characters and stored in intermediate data files with a .out extension as shown in figure 9-2.

1	USAF	WBAN	YR--MODA	HRMN	DIR	AW	AW	W	TEMP
2	29500	*****	19010101	600	140	**	**	*	6
3	29500	*****	19010101	1300	200	**	**	*	14
4	29500	*****	19010101	2000	200	**	**	*	20
5	29500	*****	19010102	600	250	**	**	*	25
6	29500	*****	19010102	1300	250	**	**	*	28
7	29500	*****	19010102	2000	250	**	**	*	29
8	29500	*****	19010103	600	250	**	**	*	28
9	29500	*****	19010103	1300	230	**	**	*	28
10	29500	*****	19010103	2000	230	**	**	*	30
11	29500	*****	19010104	600	290	**	**	*	31
12	29500	*****	19010104	1300	270	**	**	*	30
13	29500	*****	19010104	2000	200	**	**	*	31
14	29500	*****	19010105	600	230	**	**	*	32
15	29500	*****	19010105	1300	250	**	**	*	33
16	29500	*****	19010105	2000	250	**	**	*	32
17	29500	*****	19010106	600	230	**	**	*	32
18	29500	*****	19010106	1300	270	**	**	*	33
19	29500	*****	19010106	2000	270	**	**	*	32
20	29500	*****	19010107	600	140	**	**	*	25
21	29500	*****	19010107	1300	***	**	**	*	24
22	29500	*****	19010107	2000	140	**	**	*	25

Figure 9-3: Final input data file

Data contained in intermediate .out files is converted into comma separated values which are easily consumable as input by the program as illustrated in figure 9-3.

Appendix D: Code Snippets

```
for row in fileFetcher.Rows do
    let infoItem = {stationCode = row.GetColumn "USAF" ; yearMonth = row.GetColumn "YR--MODA"; hourMin = row.GetColumn "HRMN"; tem
    infoList.Add(infoItem)
    let key = {sCode = row.GetColumn "USAF" ; yrMonth = row.GetColumn "YR--MODA"; hrMin = row.GetColumn "HRMN"}
    keyList.Add(key)

    let keyBits = key.sCode + key.yrMonth + key.hrMin
    let testNumber:int64 = int64 keyBits
    let mutable testString = Convert.ToString(testNumber,2)
    let differenceInLenth  = 64 -  testString.Length
    testString <- (String.replicate differenceInLenth "0") + testString
    processingList.Add({bitString = testString; numberInIntegerFormat = keyBits; temp = float infoItem.temp })
```

Figure 10-1 : Mapping Function

The mapping function is responsible for the creation of key, value pairs from the input data.

```
|"0" -> processingList.[i].bitString <- String.Concat("0"+processingList.[i].bitString)
       setOneQubitList.[0].StateSet(1.0,0.0,0.0,0.0)
       setOneQubitList.[1].StateSet(1.0,0.0,0.0,0.0)
       let circ    = Circuit.Compile quantumFunction setOneKet.Qubits
       let circ2    = circ.GrowGates(setOneKet)
       circ2.Dump()
       circ2.Fold().RenderHT("Sorting Circuit")
       circ2.Run setOneQubitList
       match setOneQubitList.[0].Bit.v with
       |0 ->  leftList.Add(processingList.[i])
              show"%A added to Left List"processingList.[i]
       |1 ->  rightList.Add(processingList.[i])
              show"%A added to Right List"processingList.[i]
       |_  -> show""
       processingList.[i].bitString <- processingList.[i].bitString.Substring(2,(processingList.[i].bitStri
|"1" -> processingList.[i].bitString<-String.Concat("1"+processingList.[i].bitString)
       setOneQubitList.[0].StateSet(0.0,0.0,1.0,0.0)
       setOneQubitList.[1].StateSet(0.0,0.0,1.0,0.0)
       let circ    = Circuit.Compile quantumFunction setOneKet.Qubits
       let circ2    = circ.GrowGates(setOneKet)
       circ2.Dump()
       circ2.Fold().RenderHT("Sorting Circuit 2")
       circ2.Run setOneQubitList
       match setOneQubitList.[0].Bit.v with
```

Figure 10-2 : Sorting Function

The sorting function sorts key, value pairs with respect to the keys.

```
let setOneHead  = setOneQubitList.Head
CCNOT[setOneHead;setOneQubitList.[1];setOneQubitList.[2]]
M >< setOneQubitList
```

Figure 10-3 : Quantum Function

The quantum function is used to categorize the keys in either the larger or smaller keys categories.

```
if i > 0 then
    if finalList.[i].numberInIntegerFormat = finalList.[i-1].numberInIntegerFormat then
        show"Key repeated:%s"finalList.[i].numberInIntegerFormat
        totalTemp <- totalTemp + finalList.[i].temp
        accumulator <- accumulator + 1.0
    else
        show"Final Key:%s"finalList.[i-1].numberInIntegerFormat
        show"Total Temp : %f \t\t Accumulator : %f"totalTemp accumulator
        averageTemp <- totalTemp / accumulator
        show"Key:%s\t\t Average:%f"finalList.[i-1].numberInIntegerFormat averageTemp
        totalTemp <- finalList.[i].temp
        accumulator <- 1.0
```

Figure 10-4 : Reducing Function

The reducing function compacts the data into desirable information.

Appendix E: Turnitin Report