Topic

Efficiency of Searching Algorithms

Research Question

To what extent are the binary and jump search algorithms efficient for data sets of increasing sizes?

Subject

Computer Science

Session

May 2019

WORD COUNT: 3400

Table of Contents

1) Introduction

Searching algorithms are an essential aspect of computer science. They are one of the fundamental algorithms in computer science. Searching in a general perspective involves in going through individual data until the required data is found. In the field of computing, searching is viewed more than just searching through datasets. We consider the time taken for the search operations to occur while simultaneously searching through data. There are certain parameters in measuring the efficiency of searching algorithms. Time Complexity is one factor which determines this relation.

In general, the most basic searching algorithms have an average time complexity of $O(N^2)$ where N is the size of data. This is not considered efficient when it comes to searching through large numbers of data. (Rob-bell) This dilemma sparked my curiosity to compare two searching algorithms with data sets of increasing sizes and analyse how efficiently they perform when the datasets increases. This led to the research question "To what extent are the binary and jump search algorithms efficient for data sets of increasing sizes?" Using Java these algorithms were implemented as Java is platform independent meaning it does not require a specified system to run and also it is a high-level object-oriented programming language.

i) Need for Searching Algorithms

To search through an arbitrary element of sequence using random access is one solution to search for items. However, the method becomes inconsistent when performed with large data sets as the time consumed to perform such methods are highly time consuming. With large firms which have trillions of data that they have to search through to fulfil client requests or access a certain file, random access procedure will not be viable and efficient for their needs. Hence, the need for efficient methods to search through data has been in question and searching algorithm is the answer.

How do you find a misplaced phone number? How do you find your lost keys? Searching is what we do to find the answer. Similarly, Searching is performed in a computer in the case of retrieving data or if data has been misplaced or to search for the data in general. It is performed concurrently on the computer along with other tasks that help deliver the best performance.

But, for firms which have large amounts of data are very valuable and critical as most of the operations executed in the firm is based on those data. In the case of finding a certain value or retrieving a certain value after going through the datasets. Searching helps in solving deceptively simple problems which ease the task. There are various pre-coded algorithms that make the job of searching easy.

Searching algorithms are the basic operations that are needed for the efficient processing of data. It is defined as any algorithm which solves the search problem, namely, to retrieve information stored within some data structure, or calculated in the search space of a problem domain. (Geeks for Geeks) There are various searching algorithms and each of them functions differently to produce results.

The method used to search is what differentiates the algorithms from each other and how efficient and effective the method used to search is. These are used everywhere to automate searching tasks.

However, this brings the question of efficiency, obviously different algorithms follow different procedures to get the desired output and, in this case, searching for a value. So, the procedure followed by the algorithm can either generate the output fast or take time to execute. (Wikipedia) From Moore's law, it is evident in the evolution of computational power and its effect on algorithms. (Youris.com) With intensive computational power, the execution of algorithms and its efficiency are at debate, and are needed to be amplified to a greater extent.

There exists data that are exponentially large and require large applications of processing power to process the data and search through it. These data are called Big data. Big data refers to vast and voluminous data sets that may be structured or unstructured. (Oracle) These types of data are used everywhere in leading industries the reason industries opt for the usage of big data. Big data is an incentive for these leading companies as it offers the ability to structure the data for accurate data storage and organization.

ii) **Characteristics of data**

Data is another word for information. Its role has increased significantly in the global economy. Since the development of Big Data, it has offered flexibility for companies to use big data to keep track of their customers and their preferences. While the use of this seems less important, major companies like Netflix uses big data to boost their production which has tremendously helped them increase their customer base. (Insidebigdata)

Big data help the firms to take raw data and analyse through various means to utilize for the efficiency of their production and better decision management. Big data follows the properties of '3V's' which makes it unique and efficient. (Oracle)

1) **Volume**- It observes and tracks data from multiple sources and combines them creating large volumes of data.

2) **Velocity**- The streaming of data occurs in high speed causing it to analyse the data and process it occurs fast and must be dealt timely.

3) **Variety**- The data that is tracked and analysed comes in all types of the form providing flexibility in analysing all types of data.

iii) Binary and Jump Search for Big Data

Binary Search uses a traditional data processing system of searching which can search through datasets. It is one of the most basic and fundamental algorithms in Computer Science. The algorithm works on sorted datasets by dividing the dataset in half and uses a '**Divide and Conquer'** method to search for the value. It moves left and right within the dataset depending on the data being searched.

Jump Search also uses a traditional data processing system of searching which can search through datasets. It is derived from binary search and works only on sorted datasets. The basic idea of this algorithm is that it jumps a certain number of elements and searches for the value, depending on the value, if the value is smaller than the value it is currently on, it will perform a reverse linear search till the value is found, if the value is greater, it will jump again a set of data and perform the above.

This essay will explore the concept of algorithmic efficiency by analysing the Binary and Jump search algorithms in accordance to their efficiency.

iv) **General Complexity Information**

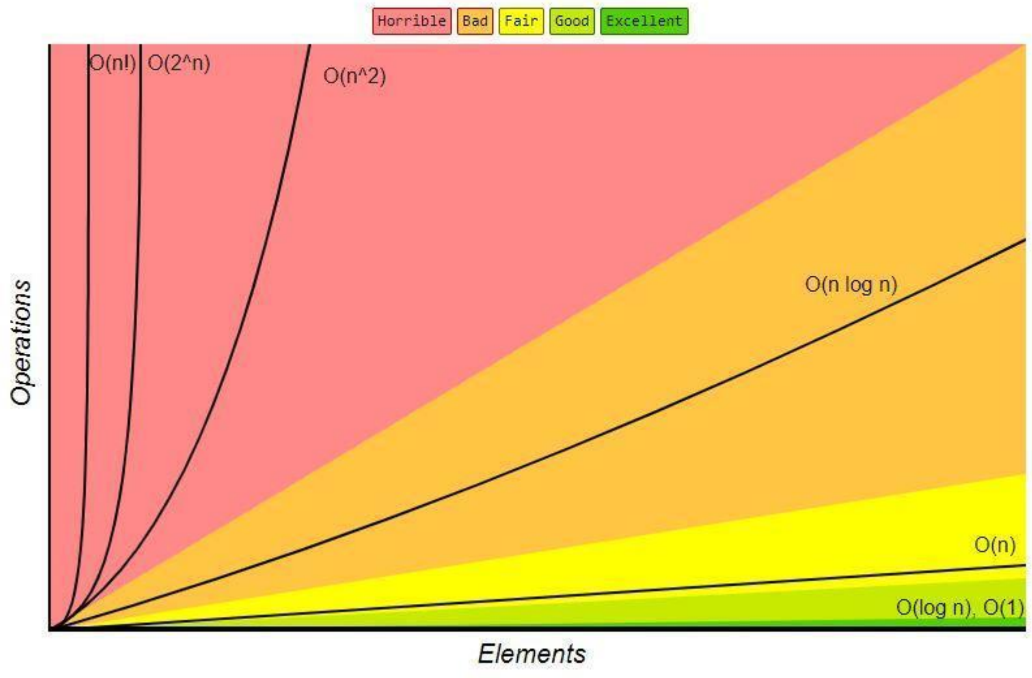


Figure 1 – Graph showing the different possible cases of algorithmic complexity

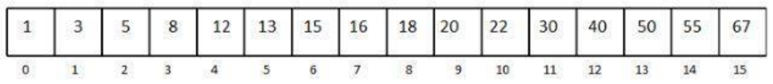
(BIg O complexity chart)

Time Complexity is described as the computational complexity that describes the running time of an algorithm. It is done so by counting the number of elementary operations in the algorithms. It is measured using the Big O Notation. (Wikipedia)

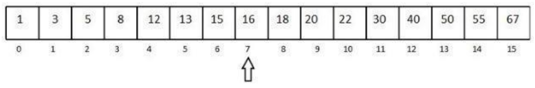
An example of such would be O(N) which stands for Linear time algorithm while O(N^x) where x > 1 stands for a polynomial time algorithm. Figure 1 above shows the time complexities and its value classified from efficient to least efficient (bottom to top).

The complexity of Binary Search is O (Log N). In general, when an algorithm is required to divide a list in half, the complexity above applies as well, this is to elaborate that the Log N is not only restricted to the Binary Search algorithm.

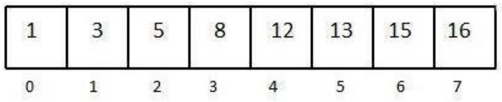
Explained below detailed pictorial representation on how the Binary Search Algorithm works and how the complexity arrives based on its procedure. (Maaz)



We want to find 13 in the given array and to do so, let us perform a binary search. Now the fundamental step of Binary search is to divide the elements and then work on searching so we select the middle element and divide the array in half,



Now, the algorithm works by comparing the target value with the middle element, if the target is equal to the middle element, the algorithm terminates and outputs the number and its position. However, in our case, 13 < 16, so the right side of the array is eliminated from the middle element as our element must be in the left side of the array from the middle element.



$$16 * \frac{1}{2} = 8$$

Now, the algorithm continues this divide and conquer method and so the right side of the array is eliminated again as 13 is less than the middle value.

12	13	15	16

$$8 * \frac{1}{2} = 4$$

Executing the same step again,

12	13

$$4 * \frac{1}{2} = 2$$

Now, executing this step for the final time we get,

13

$$2 * \frac{1}{2} = 1$$

What we can notice in the procedure of the binary search is that every time we find the middle value and deploy the ‘*divide and conquer*’ method our searching range gets divided into half the current range. So, we can assume,

$$16 * \left(\frac{1}{2}\right)^4 = 1$$

And this formula was for this situation. Similarly, for n elements, it is safe to assume that,

$$n * \left(\frac{1}{2}\right)^k = 1$$

Now, separating the power for the numerator and denominator we get,

$$n * \frac{1}{2^k} = 1$$

Multiplying both sides by the value of the denominator we get,

$$2^k * \frac{n}{2^k} = 2^k$$

Using laws of algebra, we can simplify the above equation too,

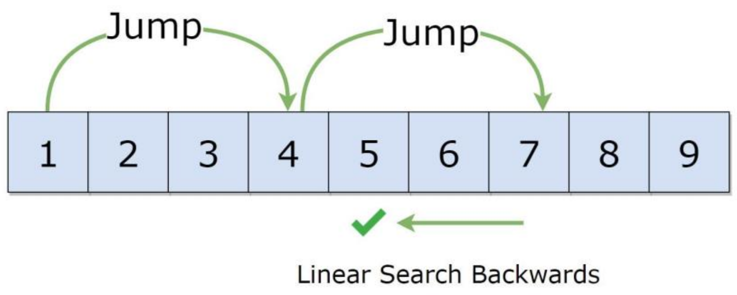
$$n = 2^k$$

Now, if we visit the definition of a logarithm, it says

$$\log_2 n = k$$

This implies that the runtime for a binary search algorithm is O (Log N). And generally, this case is referred to as the worst-case complexity and yet, many use it due to its procedure which is efficient for small datasets.

While, the Jump Search is an algorithm that works by '*jumping'* elements and then performing a reverse linear search until the target is found. The number of elements that the algorithm needs to jump is given by the formula- √N where n is the number of elements present in the sorted array. The significant difference in binary and jump search algorithm is that for the binary search the algorithm could take O (Log N) jumps while the jump search does not need such jumps and only must traverse back once. Both these algorithms are dominating in their situation. The asymptotic complexity of the Jump search algorithm is O(√n).



(Jump Search)

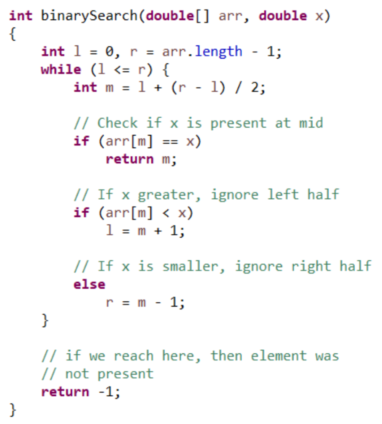
Suppose we have a dataset of size N and the block size to be jumped is B. In the worst case possible, we must perform N/B jumps and, in the case, where the element is not present, we perform B-1 comparisons. This implies that the total jumps that will be made in the worst-case scenario is ((N/B) + (B-1)). The value of this function will be minimum only when B = √N. Hence, the optimal block size to be jumped is determined by the √N. (SANGAM)

2) Investigation

i) **Hypothesis**

I think that the Binary Search will be less efficient than the Jump Search for large datasets since the binary search uses a '**Divide and Conquer**' method to search while the Jump Search jumps through elements and then performs a reverse linear search. For the Binary Search, there will be a load on the computer for the arithmetic operations needed to perform for the successful execution of the algorithm which can severely impact the efficiency rate. On the other hand, the jump search does require arithmetic operations as well but, the amount that is required is less. However, the jump search will be less efficient for smaller datasets because the execution of the algorithm itself runs 2 algorithms, the algorithm itself and a reverse linear search which means that more computational space would be required while performing the jump search and hence for smaller datasets it would not efficient as the binary search which uses simple arithmetic operations.

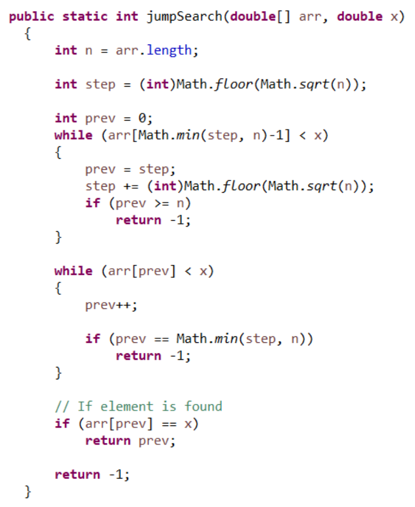
The Binary Search algorithm that I will be using to investigate is given below-



```
int binarySearch(double[] arr, double x)
{
    int l = 0, r = arr.length - 1;
    while (l <= r) {
        int m = l + (r - l) / 2;

        // Check if x is present at mid
        if (arr[m] == x)
            return m;

        // If x greater, ignore left half
        if (arr[m] < x)
            l = m + 1;

        // If x is smaller, ignore right half
        else
            r = m - 1;
    }

    // if we reach here, then element was
    // not present
    return -1;
}
```

Figure 2 (Geeksforgeeks)

The Jump Search algorithm that I will be using to investigate is given below-



```
public static int jumpSearch(double[] arr, double x)
  {
      int n = arr.length;

      int step = (int)Math.floor(Math.sqrt(n));

      int prev = 0;
      while (arr[Math.min(step, n)-1] < x)
      {
          prev = step;
          step += (int)Math.floor(Math.sqrt(n));
          if (prev >= n)
              return -1;
      }

      while (arr[prev] < x)
      {
          prev++;

          if (prev == Math.min(step, n))
              return -1;
      }

      // If element is found
      if (arr[prev] == x)
          return prev;

      return -1;
  }
```

Figure 3 (GeeksforGeeks)

To perform this investigation, I will first use the Binary Search and Jump Search Algorithm and run it with datasets of 10, 100, 1000, 10000, 100000, 1000000, 10000000 and 100000000.

ii) Binary Search Test Results

Data Size	Data	Execution time taken 1 (s)	Execution time taken 2 (s)	Execution time taken 3 (s)	Average Execution time (s)
10	4	0.02009524389	0.01178669068	0.01207440102	0.014652112
100	57	0.00891228249	0.00689965772	0.01609897676	0.010636972
1000	569	0.02673684748	0.02357405516	0.01696143398	0.022424112
10000	845	0.000172801	0.000167254	0.00016768	0.000169245
100000	1267	0.001672109	0.001977176	0.010286947	0.004645411
1000000	12699	0.008570465	0.01072044	0.008230837	0.009173914
10000000	67234	0.058131274	0.059642104	0.074439775	0.064071051
100000000	108967	0.536049326	0.565645524	0.516976875	0.539557242

Table 1 – Table showing relation between increasing data size and average execution time

iii) Jump Search Test Results

Data Size	Data	Execution time taken 1 (s)	Execution time taken 2 (s)	Execution time taken 3 (s)	Average Execution time (s)
10	4	0.000298241	0.0002176	0.00022528	0.00024704
100	57	0.001454401	0.000233387	0.000251307	0.000646365
1000	569	0.00027648	0.000254294	0.00050432	0.000345031
10000	845	0.000441174	0.000432214	0.000463361	0.000445583
100000	1267	0.001869229	0.001785176	0.004195845	0.00261675
1000000	12699	0.008800012	0.011076707	0.010452066	0.010109595
10000000	67234	0.05944157	0.070851504	0.056054685	0.06211592
100000000	108967	0.550264171	0.505569074	0.536763567	0.530865604

Table 2 - Table showing relation between increasing data size and average execution time

It is clear from table 1 and 2 that with the larger data sets the jump search algorithm outperforms the binary search algorithm as the execution time is more efficient and shorter in comparison to the binary search. while the binary search is more efficient with small data sets.

Through the rigorous execution of the algorithms in pursuit of results there are several implications in the process that needs to be considered before measuring the efficiency of one algorithm over another.

iv) Graph for Analysis

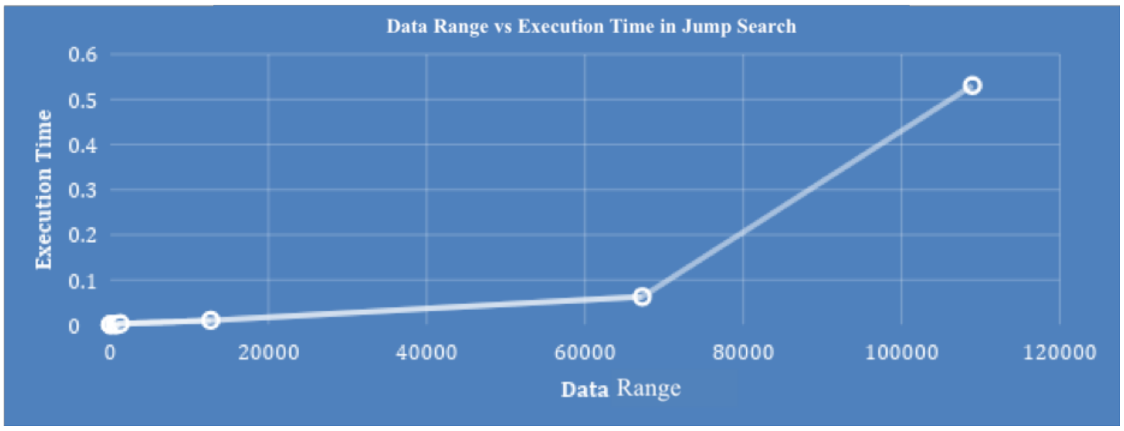


Figure 4 – Graph depicting the relation between execution time and data range

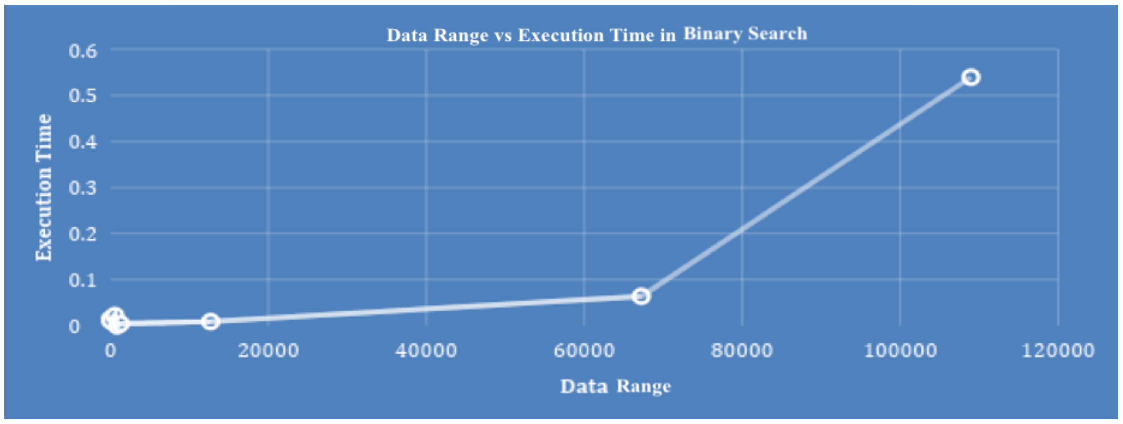


Figure 5 - Graph depicting the relation between execution time and data range

From figure 4 and 5, the point just below 20000 units of data in the binary search, the curve is linear, which implies that this search is less time consuming when searching for values in low datasets.

This means that it is more efficient than the jump search algorithm for small datasets. However, for larger datasets we see the jump search algorithm is more efficient as the execution time is relatively less, in comparison to the binary search algorithm.

To summarize:

Data Size	**Average execution time (s)**	
	Binary search	**Jump Search**
10	0.014652112	0.00024704
100	0.010636972	0.000646365
1000	0.022424112	0.000345031
10000	0.000169245	0.000445583
100000	0.004645411	0.00261675
1000000	0.009173914	0.010109595
10000000	0.064071051	0.06211592
100000000	0.539557242	0.530865604

3) Conclusion

To answer the question “**To what extent are the binary and jump search algorithms efficient for data sets of increasing sizes?**” It is clear from the investigation about the efficiency of the algorithms with the datasets. It was obvious that for **small datasets the binary search is clearly efficient while for the larger datasets the jump search is more efficient than the binary search.**

There are many compelling arguments put out by many computer scientists regarding the accurate measure of the efficiency of an algorithm but each of those arguments lack the implications towards the approach which makes the argument lose its value.

Computational power is a huge part in the role of determining algorithmic efficiency in our case the efficiency of the binary and jump search algorithms. There will always be uncertainties in the results achieved from the investigation. In recent times, there are many systems used in leading firms such as Google which take the help of supercomputers which aid them in managing their data and process requests from the servers given by clients providing accurate results in the shortest time possible. (Rijmenam)

This arouses the dilemma that increased heavy computational power and its role in measuring the accurate efficiency of the algorithm. Google has demonstrated the power of quantum computing and how it can change the dynamic impact of processing power with its claim to be 100 million times faster than the average system. This marks a change in the field of quantum computing. (NIELD) With figures like this, there is a high chance for the accurate measure of algorithms.

This figure was estimated after an in-depth analysis of the theory in quantum mechanics and how it can be related to the system for higher computational power, it has been mentioned that these results are only theoretical as of now since there is no practical method to deploy a fully functional quantum computer with the power to be 100 million times faster than a normal system. (Quantum Mechanics and Quantum Computation)

4) Implications

With the knowledge of algorithmic efficiency, it is important to consider the relation between processing power of the system and the execution time of the algorithm. But processing power is heavily dependent upon the Random-Access Memory (RAM) of the system. This is because that during the execution, the CPU stores a copy of the data that will be stored in the RAM. Therefore, for faster processing speed, a RAM with high processing speed is required. There has been an evolution for the different types of RAM and most systems nowadays come equipped with a DDR5 RAM of minimum 8 Gigabytes (GB) of memory.

The processor used in the investigation is rated '2991' from 4000 according to the Geek Benchmark. This implies that the processor is an average processor which is enough to handle medium workload and graphic content. With an average processor, the experiment on the efficiency of algorithm suggests that the jump search is better with longer data-sets while the binary search is more efficient with small data sets. However, clock speed of a processor is not the only factor which can affect the reliability of the results and efficiency. (Geekbench)

It plays a significant role during the execution time of an algorithm as the data read from the array takes multiple clock cycles which is further influenced by the prefetch and the state of each cache level in the system. While considering the option to change the processor's clock rate, there are many consequences to such a decision because changing the clock rate of the processor alter the memory bus timings (often referred to as RAM Timings) or the Disk I/O of the system. (Computer Hope)

There are multiple processes running in both the algorithms but in the jump search, there are 2 algorithms running making it more CPU intensive which will require more processing power. In the case of the binary search, there are multiple arithmetic operations running which is usually easier for a system to handle.

But, in a system isolation of a program does not occur as the functioning of a system requires many applications running to deliver the desired performance and function properly. In an average system, the most basic background applications that would be executing is the graphics driver which enables us to view content in the system along with the audio drivers which enables us to listen to sound and with the network drivers which helps us in connecting to the internet or establishing a Bluetooth connection and many more.

A CPU never stays idle in the system rather it keeps on executing on some data to keep it active and functional. The process of keeping the CPU active and running/executing programs is followed by the principles of the fetch decode instruction cycle of the CPU. The program execution of a CPU follows a chain command of Fetch-decode-read-execute. (Pal)

Implications	**High RAM**	**Low Ram**	**High Processing Power**	**Low Processing Power**
Increase in execution time	✗	✓	✗	✓
Decrease in execution time	✓	✗	✓	✗

To provide a brief summary of the above table, a system with high ram and high processing power will likely have a reduced execution time meaning that if the searching algorithm is running in such a system, the algorithms will run more efficiently. Contrastingly, if a system with low ram and low processing power will have an increase in execution time meaning that if the searching algorithm is running in such a system, the algorithms will run less efficiently.

5) Limitations

There are various assumptions done in the investigation which can limit the extensibility of the outcomes that are achieved.

Many processes were running in memory meaning that the allocation of memory to the arrays and searching through it could have an impact providing an inaccurate time because it takes into account the time for the memory to execute the process for searching and so the result is not solely only the execution time of the algorithm but it is combined with the process allocation in memory as well.

The type of hardware used can also limit the exact efficiency rate of the algorithm as different hardware perform differently as discussed in detail in implications. For the experiment, A HP Envy 15 Model was used. Various systems have various specifications which will affect the efficiency rates determined.

6) Further Scope

Both the algorithms explored and investigated has one major drawback which is that they function only for structured data. For firms which require the usage of searching algorithms data that are not structured or random, this can be a major issue. A potential solution to this issue can be addressed by successful frameworks such as **'Apache Hadoop'**. Apache Hadoop was developed in Java. Its purpose is to make the task of storing and processing data convenient. Its main objective is to manage unstructured data making it suitable for our scenario. (Apache Hadoop)

In large firms which take the help of big-data based solutions, Apache Hadoop is used. It works with the help of clusters. Clusters are a group of computers that work together which can be viewed as a single system. Hadoop does this effectively by distributing large number of data with the help of cluster and act as a single system.

Apache Hadoop consists of three main features- Hadoop YARN, Hadoop MapReduce and Hadoop Distributed File Systems (HDFS) that ensures it can process and store data effectively and efficiently.

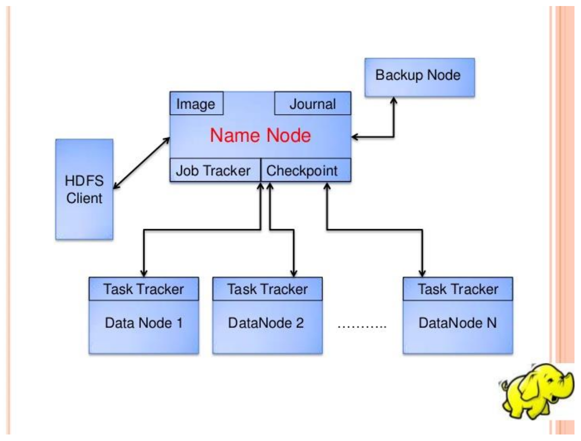


Figure 6 – Pictorial representation of HDFS

(Hadoop Distributed File System (HDFS))

Hadoop YARN is an integral part of the Hadoop framework which provides dynamic resource utilization which can enable users to use multiple Hadoop applications. Hadoop MapReduce helps in managing and processing data. It analyses large datasets parallelly before searching for results. (Clayton) HDFS provides the storage required to process and store data. It is also considered as the '**Secret Sauce'** of Apache Hadoop.

In totality, Apache Hadoop offers a great platform for the processing and storage of large unstructured data. This investigation has opened eyes as to how these algorithms are only mainly suitable for small-scale operations. For firms with large sets of data which can be complex, unstructured and random, they need to delve and explore the applications of Apache Hadoop which has great potential.

7) Appendix

My Computer Specifications:

Processor: Intel® Core™ i5- 6200U CPU @ 2.30 GHz

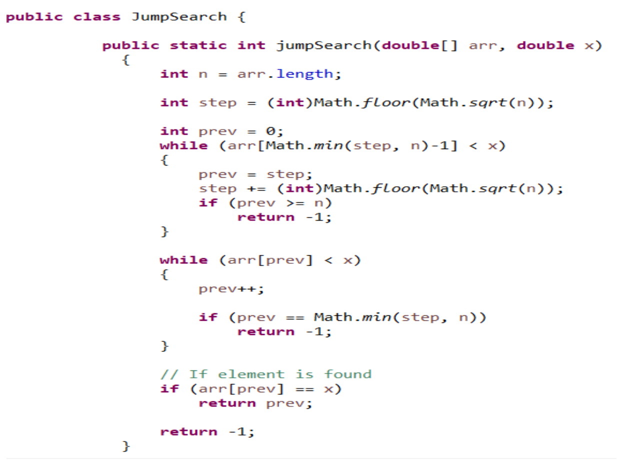
RAM: 8.00 GB

Graphics Processor: Intel® HD Graphics 520

Operating System: Windows 10 Home Premium

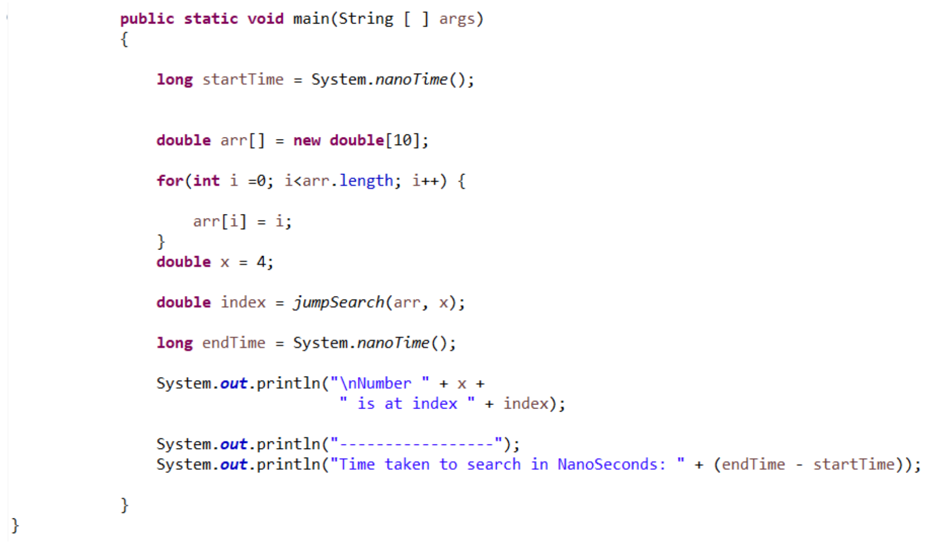
Project Code

JumpSearch.java:



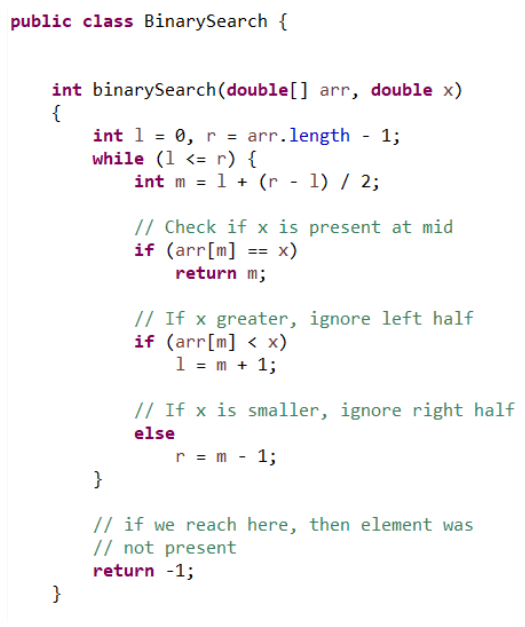
```java
public class JumpSearch {

          public static int jumpSearch(double[] arr, double x)
            {
                int n = arr.length;

                int step = (int)Math.floor(Math.sqrt(n));

                int prev = 0;
                while (arr[Math.min(step, n)-1] < x)
                {
                    prev = step;
                    step += (int)Math.floor(Math.sqrt(n));
                    if (prev >= n)
                        return -1;
                }

                while (arr[prev] < x)
                {
                    prev++;

                    if (prev == Math.min(step, n))
                        return -1;
                }

                // If element is found
                if (arr[prev] == x)
                    return prev;

                return -1;
            }
```

Driver Method:



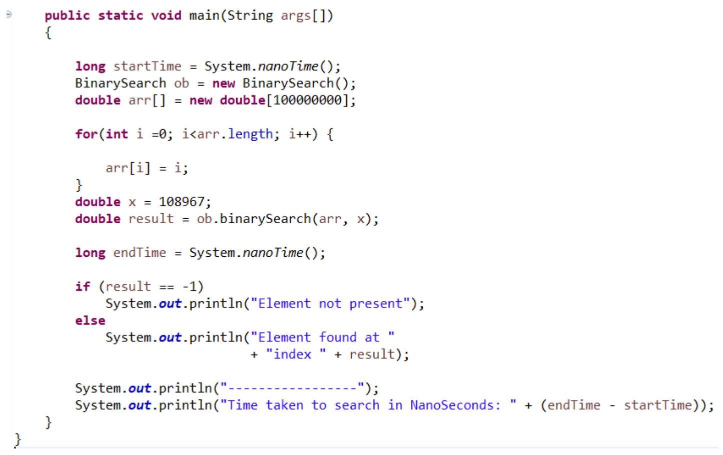
```java
            public static void main(String [ ] args)
            {

                long startTime = System.nanoTime();


                double arr[] = new double[10];

                for(int i =0; i<arr.length; i++) {

                    arr[i] = i;
                }
                double x = 4;

                double index = jumpSearch(arr, x);

                long endTime = System.nanoTime();

                System.out.println("\nNumber " + x +
                                   " is at index " + index);

                System.out.println("----------------");
                System.out.println("Time taken to search in NanoSeconds: " + (endTime - startTime));

            }
}
```

BinarySearch.java:



```
public class BinarySearch {

    int binarySearch(double[] arr, double x)
    {
        int l = 0, r = arr.length - 1;
        while (l <= r) {
            int m = l + (r - l) / 2;

            // Check if x is present at mid
            if (arr[m] == x)
                return m;

            // If x greater, ignore left half
            if (arr[m] < x)
                l = m + 1;

            // If x is smaller, ignore right half
            else
                r = m - 1;
        }

        // if we reach here, then element was
        // not present
        return -1;
    }
```

Driver Method:



```
    public static void main(String args[])
    {

        long startTime = System.nanoTime();
        BinarySearch ob = new BinarySearch();
        double arr[] = new double[100000000];

        for(int i =0; i<arr.length; i++) {

            arr[i] = i;
        }
        double x = 108967;
        double result = ob.binarySearch(arr, x);

        long endTime = System.nanoTime();

        if (result == -1)
            System.out.println("Element not present");
        else
            System.out.println("Element found at "
                                   + "index " + result);

        System.out.println("-----------------");
        System.out.println("Time taken to search in NanoSeconds: " + (endTime - startTime));
    }
}
```

Works Cited

Apache Hadoop. n.d. <https://en.wikipedia.org/wiki/Apache_Hadoop>.

BIg O complexity chart. n.d. <https://cdn-images-1.medium.com/max/2000/0*1zzcLny-dV1hjxAI>.

Blansit, B. Douglas. *Firewalls: Basic Principles and Some Implications*. 4 September 2009. <https://www.tandfonline.com/doi/pdf/10.1080/15424060903167377>.

Clayton, Richard. *What are all algorithms used by Hadoop/MapReduce?* . 5 January 2013. <https://www.quora.com/What-are-all-algorithms-used-by-Hadoop-MapReduce>.

Computer Hope. *Clock cycle*. 26 April 2017. <https://www.computerhope.com/jargon/c/clockcyc.htm>.

Geekbench. *Benchmarks*. n.d. <http://browser.geekbench.com/processor-benchmarks>.

Geeks for Geeks. *Searching Algorithms*. n.d. <https://www.geeksforgeeks.org/searching-algorithms/>.

Geeksforgeeks. *Binary Search* . n.d. <https://www.geeksforgeeks.org/binary-search/>.

GeeksforGeeks. *Jump Search*. n.d. <https://www.geeksforgeeks.org/jump-search/>.

Hadoop Distributed File System (HDFS). n.d.

<https://www.google.com/url?sa=i&source=images&cd=&cad=rja&uact=8&ved=2ahUKEwjd8tzi-f7gAhWDu48KHb49CoQQjRx6BAgBEAU&url=https%3A%2F%2Fwww.slideshare.net%2FNitinKhattar%2Fhdf-11802019&psig=AOvVaw0dHUC6QTi_Md18Dl_7YXw0&ust=1552560760449431>.

Insidebigdata. *How Netflix Uses Big Data to Drive Success*. 20 January 2018. <https://insidebigdata.com/2018/01/20/netflix-uses-big-data-drive-success/>.

Jump Search. n.d. <https://www.google.com/url?sa=i&source=images&cd=&cad=rja&uact=8&ved=2ahUKEwiKvIbu8P7gAhVIuY8KHTVoC8EQjRx6BAgBEAU&url=http%3A%2F%2Ftheoryofprogramming.com%2F2016%2F11%2F10%2Fjump-search-algorithm%2F&psig=AOvVaw1tKQ2AEKmoTJRoKqwcrv86&ust=1552558206424536>.

Maaz. *What does the time complexity O(log n) actually mean?* 28 May 2017. <What does the time complexity O(log n) actually mean?>.

NIELD, DAVID. *Google's Quantum Computer Is 100 Million Times Faster Than Your Laptop*. 10 December 2015. <https://www.sciencealert.com/google-s-quantum-computer-is-100-million-times-faster-than-your-laptop>.

Oracle. *What Is Big Data?* n.d. <https://www.oracle.com/in/big-data/guide/what-is-big-data.html>.

Pal, Neha. *What is fetch cycle, instruction cycle and execution cycle?* 10 September 2018. <https://www.quora.com/What-is-fetch-cycle-instruction-cycle-and-execution-cycle>.

Quantum Mechanics and Quantum Computation. n.d. <https://www.edx.org/course/quantum-mechanics-quantum-computation-uc-berkeleyx-cs-191x>.

Rijmenam, Mark van. *How Google Applies Big Data To Know You - Infographic*. n.d. <https://datafloq.com/read/google-applies-big-data-infographic/385>.

Rob-bell. *A beginner's guide to Big O notation*. n.d. <https://rob-bell.net/2009/06/a-beginners-guide-to-big-o-notation/>.

SANGAM, VAMSI. *Jump Search*. 10 Nvember 2016. <http://theoryofprogramming.com/2016/11/10/jump-search-algorithm/>.

What are important factors which affect the running time of a program? 13 Octiober 2014. <https://www.quora.com/What-are-important-factors-which-affect-the-running-time-of-a-program>.

Wikipedia. *Algorithmic efficiency*. n.d. <https://en.wikipedia.org/wiki/Algorithmic_efficiency>.

—. *Time complexity*. n.d. <https://en.wikipedia.org/wiki/Time_complexity>.

Youris.com. *Processing power beyond Moore's Law*. 20 April 2018. <https://phys.org/news/2018-04-power-law.html>.

Zaveri, Meet. *Algorithms I : Searching and Sorting algorithms*. 10 March 2018. <https://codeburst.io/algorithms-i-searching-and-sorting-algorithms-56497dbaef20>.