

Imam Abdulrahman Bin Faisal University College of Science and Humanities studies Department of Computer Science

CS412 – Algorithm Analysis and Design Academic Year (2022-2023) – First Semester

**Supervised by: Dr.Azza Ahmed Abdo Ali**

|  |  |
| --- | --- |
| **Students Names:** | **ID** |
| Wadha Nayef Alsheddi | 2200003246 |
| Fatimah Fadel Alharz | 2200004362 |
| Amal Mohammed Alotaibi | 2200001746 |
| Asma Zaher Alshehri | 2200000484 |
| Dema Hamed Alghamdi | 2200001218 |

## Table of content:

[Introduction… 2](#_TOC_250005)

1. [Theoretical Question… 2](#_TOC_250004)
   1. [Insertion Sort… 2](#_TOC_250003)
   2. [Merge Sort 3](#_TOC_250002)
   3. [Heap Sort 3](#_TOC_250001)
2. [Data Generation and Experimental Setup… 4](#_TOC_250000)
3. Which of the three sorts seems to perform the best?. 5
4. What are the best, average and worst-case analyses that agree with the experimental results? 9
5. The comparison sorts show if the number of comparisons really a good predictor of the execution time? 18
6. Design and implement algorithm and results 22
7. Explanation on why result on theoretical and the experimental differ 23
8. Sorting Algorithms Experimental Run time results 24
9. Group member tasks 26

# Introduction

In this report, we will make a comparison between the following algorithms, which are Insertion Sort, Merge Sort and Heap Sort. The best algorithm in terms of performance will be deduced based on the running time. There is great importance to the sorting process, which is that it saves time when searching, for example, Sorting makes it easy to search for numbers and names and help you order elements that are in a list. and Produce outputs that are easy for humans to read because they are arranged, we will use the following device specifications which are: RAM is 8 GB, the operating system is Windows 11, Processer is Intel Core i7, System Type is 64-bit and We must know that the specifications of the device affect the running time.

# Theoretical Question

Generally sorting algorithms may arrange the input elements in ascending or descending order. However, our analysis will be based on the given code which sorts the input list in ascending order.

# Insertion Sort

Insertion sort is an in-place sorting algorithm that follows an incremental approach. The elements are split into two parts, an unsorted and a sorted part. Every time an element is picked from the unsorted part and compared with the sorted part, then inserted into its correct position.

## Best Case:

The best-case running time of the insertion sort algorithm is **Θ(n)**. It has a linear running time because the inner loop will execute (n-1) times as its body will not be executed. This occurs when the input elements are already sorted in ascending order. We expect that the experimental running time will be the same as the theoretical because the generated input list from the code will be arranged in ascending order.

## Worst Case:

The worst-case running time of the insertion sort algorithm is Θ(n2). It has a quadratic running time because the body of the inner loop will always be executed. As a result, the inner loop will execute (n(n+1)/2)-1) times, which will result in a quadratic running time. This occurs when the input elements are sorted in reversed order. We expect the experimental results to agree with the theoretical because the generated input list will always return a descending order list.

## Average Case:

The average-case running time of the insertion sort algorithm is Θ(n2). It has a quadratic running time because the inner loop's body will approximately be executed in half of the iterations. Hence, the running time of the average case will be as bad as the worst case. This happens when the input elements are almost half sorted. We expect the experimental results to agree with the theoretical if the randomly generated list from the code is almost half sorted.

## Merge Sort

Merge sort is an algorithm that uses the divide and conquer approach, which repeatedly splits the input in half to sort the entered sequence of numbers. This algorithm's running time is not affected by the order of the input list, as the list in all cases will be divided and then combined. Hence, **the merge sort's running time in the best, average, and worst case is Θ(n lg n).** Additionally, the combining step in this sorting algorithm has the greatest effect on the running time. We expect that the experimental running time will be the same as the theoretical since the merge sort always divides the array into two subparts and takes linear time to combine two halves. Therefore, the execution time will be the same in all cases.

## Heap Sort

Heapsort is an algorithm that uses an almost complete binary tree structure to sort the input elements. This algorithm must apply its two main operations, which are Max-Heapify and Build-Max-Heap, regardless of the type of input. Therefore, it is not affected by the input list order. For

this reason, the running time of the heap sort in the **best, average, and worst case is Θ (n lg n).** Therefore, we expect that the experimental running time will agree with the theoretical as the two main operations will always be performed.

## Data Generation and Experimental Setup

* 1. What kind of machine did you use?

To obtain the most accurate results, we used the NetBeans virtual machine on a Windows 11 computer. **Table 1** shows the characteristics of the original and the virtual machines used.

**Table 1 Characteristics of The Original Machine and The Virtual Machine**

|  |  |  |
| --- | --- | --- |
| Characteristics | The Original Machine | The Virtual Machine |
| Operating System | Windows 11 | Ubuntu |
| Processor | Intel Core i7 | 1 Allocated CPUs |
| RAM | 8 GB | 835.8 MB |
| System Type | 64 Bit | 64 Bit |

.

## What timing mechanism?

We used CurrentTimeMillis() method which returns the current time in milliseconds. The accuracy of the value depends on the underlying operating system.

## How many times did you repeat each experiment? Explain your choice.

We repeated each experiment three times because the results were always close. Thus, in most of the sorting algorithms, three repetitions were enough to conclude a reasonable outcome.

## What times are reported?

The time reported was the median of the three obtained running times. Even though computing the average value was recommended, we computed the median because the three trials' values were close to each other. Therefore, the median's value will approximately be the same as the average value. Also, the median is not affected by the outliers' existence, which are the points that differ significantly from other values that might affect the value of the average.

## What is the input to the algorithms? How did you select it?

The input to the algorithms starts from 100 and increases to 100000 by a multiple of 5. We chose this range because it illustrates the input size's effect on the running time clearly for each algorithm. We stopped at the value of 100000 as the running time took too long for some algorithms causing the program to lag and the computer to overheat.

## Did you use the same inputs for all sorting algorithms?

Yes, we used the same input for all sorting algorithms to simplify the comparison between the sorting algorithms and attain the most accurate conclusion.

Additionally, in the random input list, we chose the seed to be number one in order to generate the same input for all sorting algorithms and to get the same number or comparison for each sorting algorithm to make sure we gather same results .

## Which of the three sorts seems to perform the best (consider the best version)

We shall be discussing the best, average, and worst case for each sorting algorithm and we will showcase a graph that contains input size and for each and an expiation for each case and runtime.

**note that**: insertion sort in the average and worst case is taking the most tine because insertion sort is only considered a best in best cases were its **(Θ (n))** , but on average and worst **its (Θ (𝑛^2))** which is much larger than merge and heap sorting.

## Best case:

The best-cases of all sorting algorithms:

* **Insertion sort**: Increasing list. **(**Θ **(n))**
* **Merge sort:** Increasing list. **(**Θ **(n lg n))**
* **Heapsort**: Increasing list. **(O(n lg n))**

Figure 1 illustrates the best cases of each sorting algorithm. The line graph shows that the insertion sort is the fastest sorting algorithm while comparing the best cases of all the sorting algorithms, as the insertion sort performs the best among all sorting algorithms while comparing the best cases as it takes **(**Θ **(n))**. In contrast, the others take and. the heap sort comes as the second-fastest sorting algorithm as it takes (O **(n lg n))**. the last go for the merge sorting algorithm it takes **(**Θ **(n lg n)).**

Figure 1 Best cases for each algorithm

## Average case:

The average case for all sorting algorithms:

* + **Insertion sort:** Random list. **(**Θ **(𝑛^2))**
  + **Merge sort:** Increasing list. **(**Θ **(n lg n))**
  + **Heapsort:** Increasing list. **(O(n lg n))**

Figure 2 illustrates the average cases of each sorting algorithm. The line graph shows that the, as for best is the merge sorting algorithm it takes. **(**Θ **(n lg n))** . heap sort is second best sorting algorithm **(O(n lg n))** . as the insertion sort is the worst for average among all sorting **(**Θ **(𝑛^2))**.

Average Cases

6000000

5000000

4000000

3000000

2000000

1000000

0

100

500

1000

5000

10000

50000

100000

Merge

Insertion

Heap

Figure 2 for average cases for each algorithm

## Worst case:

The worst-cases of all sorting algorithms:

* + **Insertion sort:** Random list. **(**Θ **(𝑛^2))**
  + **Merge sort:** Random list. **(**Θ **(n lg n))**
  + **Heapsort:** Random list. **(O(n lg n))**

Figure 3 illustrates the worst cases of each sorting algorithm. The line graph shows that the insertion is the worst sorting algorithm **(**Θ **(𝑛^2))**, as the merge **(**Θ **(n lg n))**

. comes in best and then comes heap sort **(O(n lg n))** .

Worst Cases

18000000

16000000

14000000

12000000

10000000

8000000

6000000

4000000

2000000

0

100

500

1000

5000

10000

50000

100000

Merge

Insertion

Heap

Figure 3 worst cases for each algorithm

# To outline the privies:

The best performance of the three sorts:

### − Insertion sort in the best case.

* **Merge sort in the average case.**
* **Merge in the worst case.**

For accurate calculation of the best running time, it is essential to have a large input size. In terms of running time, merge sort is the best sorting algorithm since we always take the worst case into account. for operating heap sort only requires a single array and no recursive calls as in merge sort. For large input sizes, merge sort is considered faster than heap sort. It is possible, however, that the fastest algorithm differs from the merge sort algorithm when input sizes are small. The most efficient sorting algorithm will also be insertion sort if the list is sorted.

1. To what extent does the best, average, and worst-case analyses (from class/textbook) of each sort agree with the experimental results?

We have divided the theoretical running time by the experimental running time in this section to compare its asymptotic order with that of the experimental running time. The reason we assume that some experimental results may differ

from theoretical results is due to several factors that could affect the run time,

such as CPU utilization, compiler speed, and programming language. As shown in the diagram, the y-axis represents the quotient of the experimental and theoretical running times, and the x-axis represents the input size.

## Best Case Analysis:

Insertion sort

The insertion sort best-case analysis is shown in **Figure 4** and **Table 2**. The graph below show that we concluded that starting at this point, the experiment is consistent with the theory, which is (n).

|  |  |  |
| --- | --- | --- |
| input | Experimental time | Theoretical Result |
| 100 | 3 | 100 |
| 500 | 5 | 500 |
| 1000 | 9 | 1000 |
| 5000 | 40 | 5000 |
| 10000 | 88 | 10000 |
| 50000 | 425 | 50000 |
| 100000 | 830 | 100000 |

Table 2

Insertion Sort Best Case Theory Vs. Experimental

0.035

0.03

0.025

0.02

0.015

0.01

0.005

0

100

500

1000

5000

10000

50000

100000

Figure 4 insertion sort bast case theory vs experimental

## Merge sort

The merge sort best-case analysis is illustrated in **Figure 5** and **Table 3**. In the line graph below, a horizontal line started appearing when n0=50000 and c=0.00648. Hence, at this point, the experiment agrees with the theory of Θ (nlgn).

|  |  |  |
| --- | --- | --- |
| input | Experimental time | Theoretical Result |
| 100 | 10 | 664.385619 |
| 500 | 36 | 4482.892142 |
| 1000 | 73 | 9965.784285 |
| 5000 | 437 | 61438.5619 |
| 10000 | 910 | 132877.1238 |
| 50000 | 5060 | 780482.0237 |
| 100000 | 10664 | 1660964.047 |

Table 3

Merge Sort Best Case Theory Vs. Experimental

0.016

0.014

0.012

0.01

0.008

0.006

0.004

0.002

0

100

500

1000

5000

10000

50000

100000

## Heap sort

Figure 5 Merge sort best case theory vs experimental

The heap sort best-case analysis is illustrated in **Figure 6** and **Table 4**. The line graph below shows no clear horizontal line. When the inputs were 50000 and 100000, however, the difference between the experimental running times got closer than the differences between the other consecutive inputs. The experimental result will agree with the theoretical result if the running time doesn't exceed O(nlgn), since the best-case heap sort has a running time of O(nlgn). Figure 6 shows the experimental result does not exceed (nlgn).

Therefore, the theoretical result is compatible with the experimental result.

|  |  |  |
| --- | --- | --- |
| input | Experimental time | Theoretical Result |
| 100 | 13 | 664.385619 |
| 500 | 78 | 4482.892142 |
| 1000 | 167 | 9965.784285 |
| 5000 | 956 | 61438.5619 |
| 10000 | 2016 | 132877.1238 |
| 50000 | 11212 | 780482.0237 |
| 100000 | 23504 | 1660964.047 |

Table 4

Heap Sort Best Case Theory Vs. Experimental

0.025

0.02

0.015

0.01

0.005

0

100

500

1000

5000

10000

50000

100000

Figure 6 Heap sort best case theory vs experimental

## Average Case Analysis:

Insertion sort average case:

Comparing the average-case analysis of the insertion sort is shown in **Figure 7** As shown in the **Table 5** below, we concluded that the experiment agrees with the theory, which is (n2).

|  |  |  |
| --- | --- | --- |
| input | Experimental time | Theoretical Result |
| 100 | 15 | 10000 |
| 500 | 321 | 250000 |
| 1000 | 1070 | 1000000 |
| 5000 | 20187 | 25000000 |
| 10000 | 66292 | 100000000 |
| 50000 | 1343882 | 2500000000 |
| 100000 | 5349064 | 10000000000 |

Table 5

Insertion Sort Average Case Theory Vs.

Experimental

0.0016

0.0014

0.0012

0.001

0.0008

0.0006

0.0004

0.0002

0

100

500

1000

5000

10000

50000

100000

Figure 7 Insertion sort average theory vs experimental

## Merge sort average case:

The average-case analysis of the merge sort is shown in **Figure 8** and **Table 6**. The graph show that the experiment does not support the theory. However, we can see that the line graph is almost horizontal when the input size is 10000.

Consequently, if we tried larger input sizes, we might begin to see horizontal lines. Additionally, if we repeated the experiment more than three times, a horizontal line may have appeared.

|  |  |  |
| --- | --- | --- |
| input | Experimental time | Theoretical Result |
| 100 | 9 | 664.385619 |
| 500 | 36 | 4482.892142 |
| 1000 | 73 | 9965.784285 |
| 5000 | 468 | 61438.5619 |
| 10000 | 918 | 132877.1238 |
| 50000 | 5200 | 780482.0237 |
| 10000 | 10744 | 1660964.047 |

Table 6

Merge Sort Average Case Theory Vs. Experimental

0.016

0.014

0.012

0.01

0.008

0.006

0.004

0.002

0

100

500

1000

5000

10000

50000

100000

Figure 8 Merge sort avg case theory vs experimental

## Heap sort average case:

**Figure 9 and table 7** as shown below shows the comparison of the heap sort average-case analysis. It indicates that a horizontal line appeared when n0=50000 and c=0.0145. Furthermore, (nlgn) explains the experiment at this point.

|  |  |  |
| --- | --- | --- |
| input | Experimental time | Theoretical Result |
| 100 | 10 | 664.385619 |
| 500 | 81 | 4482.892142 |
| 1000 | 173 | 9965.784285 |
| 5000 | 966 | 61438.5619 |
| 10000 | 2019 | 132877.1238 |
| 50000 | 11341 | 780482.0237 |
| 100000 | 24020 | 1660964.047 |

Table 7

Heap Sort Average Case Theory Vs. Experimental

0.02

0.018

0.016

0.014

0.012

0.01

0.008

0.006

0.004

0.002

0

100

500

1000

5000

10000

50000

100000

Figure 9 Heap sort avg case theory vs experimental

## Worst Case Analysis:

Insertion sort worst case:

**Figure 10** and **Table 8** show the comparison of an insertion sort worst-case scenario. The line graph below does not show a clear horizontal line, which indicates that the experiment doesn't support the theory. We concluded that the experiment did not support the theory. However, the difference between the running time at the inputs 50000 and 100000 began to close compared to that at the other consecutive inputs. Consequently, a horizontal line might appear if we tried larger input sizes.

|  |  |  |
| --- | --- | --- |
| input | Experimental time | Theoretical Result |
| 100 | 33 | 10000 |
| 500 | 629 | 250000 |
| 1000 | 2416 | 1000000 |
| 5000 | 58981 | 25000000 |
| 10000 | 193037 | 100000000 |
| 50000 | 4001041 | 2500000000 |
| 100000 | 15420336 | 10000000000 |

Table 8

Insertion Sort Worst Case Theory Vs. Experimental

0.0035

0.003

0.0025

0.002

0.0015

0.001

0.0005

0

100

500

1000

5000

10000

50000

100000

Figure 10 Insertion sort worst case theory vs experimental

## Merge sort worst case:

We concluded that the experiment does not align with the theory based on the comparison of the merge sort worst-case analysis shown in **Figure 11** and **Table 9**. Nevertheless, a straight line was close to appearing between the input sizes 1000 and 10000 when a straight line was near to appearing. However, the line graph pattern changed suddenly. It might have been due to the few repetitions behind the sudden change in pattern.

|  |  |  |
| --- | --- | --- |
| input | Experimental time | Theoretical Result |
| 100 | 16 | 664.385619 |
| 500 | 81 | 4482.892142 |
| 1000 | 172 | 9965.784285 |
| 5000 | 1026 | 61438.5619 |
| 10000 | 2192 | 132877.1238 |
| 50000 | 10830 | 780482.0237 |
| 100000 | 21614 | 1660964.047 |

Merge Sort Worst Case Theory Vs.

Experimental

0.03

0.025

0.02

0.015

0.01

0.005

0

100

500

1000

5000

10000

50000

100000

Figure 11 Merge sort worst case theory vs experimental

## Heap sort worst case:

A comparison of the worst-case heap sort analysis is shown in **Figure 12** and **Table 10**. From the line graph below, we could see a close appearance of a horizontal line from a input size of 1000 to a size of 10000, but it changed As a result of the few repetitions, it might explain why the line graph pattern fell so rapidly. As more repetitions are performed, computing the average will be more accurate than computing the median, since the values of each trial will differ more significantly. It follows that if the running time does not exceed (nlgn), the experimental results will be consistent with the theoretical results because the worst-case for heap sort is O(nlgn). **Figure 12** shows that the experimental result does not exceed (nlgn). Therefore, the theoretical result is consistent with the experimental result. with the increasing input size.

|  |  |  |
| --- | --- | --- |
| input | Experimental time | Theoretical Result |
| 100 | 15 | 664.385619 |
| 500 | 91 | 4482.892142 |
| 1000 | 201 | 9965.784285 |
| 5000 | 1234 | 61438.5619 |
| 10000 | 2708 | 132877.1238 |
| 50000 | 14626 | 780482.0237 |
| 100000 | 24460 | 1660964.047 |



100000

50000

10000

5000

1000

500

100

0

0.005

0.01

0.015

0.02

0.025

Heap Sort Worst Case Theory Vs. Experimental

Figure 12 Heap worst case theory vs experimental

## For the comparison sorts, is the number of comparisons really a good predictor of the execution time? In other words, is a comparison a good choice of basic operation for analyzing these algorithms?

we have plotted a line graph for every sorting algorithm to find the relation between the number of comparisons and the running time. We have used the worst case for all the three sorting algorithms, as it is the best indicator while comparing. Before plotting the line graphs, we assumed a positive relationship between the number of comparisons and the running time, which means that the number of comparisons is a reliable predictor for the execution time

Table 11insertion sort comparison:

|  |  |  |
| --- | --- | --- |
| **input** | **Comparison** | **Experimental time** |
| **100** | **5000** | 33 |
| **500** | **125000** | 629 |
| **1000** | **500000** | 2416 |
| **5000** | **12500000** | 58981 |
| **10000** | **50000000** | 193037 |
| **50000** | **1250000000** | 4001041 |
| **100000** | **5000000000** | 15420336 |

Table 12 Heap sort comparison

|  |  |  |
| --- | --- | --- |
| **input** | **Comparison** | **Experimental time** |
| **100** | **664.4** | 15 |
| **500** | **4483** | 91 |
| **1000** | **9965.8** | 201 |
| **5000** | **61438.6** | 1234 |
| **10000** | **132877.1** | 2708 |
| **50000** | **780482** | 14626 |
| **100000** | **1660964** | 24460 |

Table 13 Merge sort comparison:

|  |  |  |
| --- | --- | --- |
| **input** | **Comparison** | **Experimental time** |
| **100** | **664.4** | 16 |
| **500** | **4483** | 81 |
| **1000** | **9965.8** | 172 |
| **5000** | **61438.6** | 1026 |
| **10000** | **132877.1** | 2192 |
| **50000** | **780482** | 10830 |
| **100000** | **1660964** | 21614 |

The following figures plot the time and comparisons together in line graphs:

Heap Sort Comparison Vs Time

Median

1800000

1600000

1400000

1200000

1000000

800000

600000

400000

200000

0

100

500

1000 5000 10000 50000 100000

input

Comparison

**Figure 13** Heap sort comparison

Insertion Sort Comparison Vs Time

Median

6E+09

5E+09

4E+09

3E+09

2E+09

1E+09

0

100

500

1000

5000 10000 50000 100000

input

comparision

**Figure 14** Insertion sort comparison

Merge Sort Comparison Vs Time

Median

1800000

1600000

1400000

1200000

1000000

800000

600000

400000

200000

0

100

500

1000

5000

10000 50000 100000

input

Comparison

**Figure 15** Merge sort comparison

According to the line graphs, it is shown that when the number of comparisons increases, the running time increases in all sorting algorithms. However, it is not easy to conclude the reliability of relying on the number of comparisons as a good predictor by only looking at the line graphs. Therefore, to support our conclusion, we have computed the correlation between the number of comparisons and time execution in each line graph. According to the results listed below, we can conclude that the number of comparisons is strongly correlated with the execution time in all three sorting algorithms. Therefore, the number of comparisons is a good predictor, **especially in merge sort.**

1. Design and analysis an improved Divide-and-conquer algorithm compute a n , where n is natural number and calculate time complexity

|  |  |  |
| --- | --- | --- |
| **Design and analysis an improved Divide-and-conquer algorithm**  **compute a n , where n is natural number.**  **- Divide and conquer (recursive) int Findingthepower(int a, int n)** | **Cost** | **Time** |
| **1- if n == 0** | **C1** | **1** |
| **2- return 1** | **C2** | **1** |
| **3- else if n%2==0** | **C3** | **1** |
| **4- return Findingthepower(a, n/2)\* Findingthepower(a, n/ 2)** | **C4** | **n** |
| **5- else** | **C5** | **1** |
| **6 -return a\* findingthepower(a, n/ 2) \* findingthepower(a, n / 2)** | **C6** | **n** |

**Analysis for the running time: T(n)=C1(1)+C2(1)+C3(1)+C4(n)+C5(1)+C6(n)= O(n)**

**Time Complexity: O(n) Space Complexity: O(n)**

|  |  |  |
| --- | --- | --- |
| **- Optimized approach:**  **In the above approach same subproblem is computed twice for each recursive call. We can optimize it by computing the solution of the subproblem once only.** |  | |
| **int findingthepower(int a, int n)** | **Cost** | **Time** |
| **1- if n == 0** | **C1** | **1** |
| **2- return 1** | **C2** | **1** |
| **3- x = findingthepower(a, n / 2)** | **C3** | **Log n** |
| **4- else if n%2==0** | **C4** | **1** |
| **5- return x\*x** | **C5** | **1** |
| **6- else** | **C6** | **1** |
| **7- return a\*x\*x** | **C7** | **1** |
| **Analysis for the running time:**  **T(n)=C1(1)+C2(1)+C3(log n)+C4(1)+C5(1)+C6(1)+C7(1)= O(log n)** |  |  |
| **Time Complexity: O(log n) Space Complexity: O(log n)** |  |  |

1. explain: the theoretical and the **experimental** results:

**According to the tables previously, the input size and theoretical results have a pattern, whereas as input increases, theoretical results increase as well. Furthermore, the input size and the results of the experiment have the same trend where the input increases so do the results.**

**Note that:**

**This is due to the power and speed of the core i7 in the laptop used for running and examining the project, which caused noticeable the difference between the theoretical and experimental results. If the core was less than i7 and slow, the theoretical and experimentalists could have provided results.**

1. Sorting Algorithms Experimental Run time results:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Insertion Sort** | | | | | |
| **Random Generated List:** | | | | | |
| **Inputs** | **Time1(mS)** | **Time2(mS)** | **Time3(mS)** | **Time Median** | **Comparison** |
| **100** | 3 | 15 | 33 | 15 | 5000 |
| **500** | 5 | 321 | 629 | 321 | 125000 |
| **1000** | 9 | 1070 | 2416 | 1070 | 500000 |
| **5000** | 40 | 20187 | 58981 | 20187 | 12500000 |
| **10000** | 88 | 66292 | 193037 | 66292 | 50000000 |
| **50000** | 425 | 1343882 | 4001041 | 1343882 | 1250000000 |
| **100000** | 830 | 5349064 | 15420336 | 5349064 | 5000000000 |
| **Increasing Generated List:** | | | | | |
| **Inputs** | **Time1(mS)** | **Time2(mS)** | **Time3(mS)** | **Time Median** | **Comparison** |
| **100** | 2 | 3 | 3 | 3 | 99 |
| **500** | 5 | 5 | 6 | 5 | 499 |
| **1000** | 9 | 9 | 9 | 9 | 999 |
| **5000** | 41 | 40 | 40 | 40 | 4999 |
| **10000** | 80 | 88 | 102 | 88 | 9999 |
| **50000** | 425 | 431 | 414 | 425 | 49999 |
| **100000** | 830 | 832 | 826 | 830 | 99999 |
|  | **Table 14 I**nsertion sort Experime | | | ntal Run time results | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Merge Sort** | | | | | |
| **Random Generated List:** | | | | | |
| **Inputs** | **Time1(mS)** | **Time2(mS)** | **Time3(mS)** | **Time Median** | **Comparison** |
| **100** | 10 | 9 | 16 | 16 | 664.4 |
| **500** | 36 | 36 | 81 | 81 | 4483 |
| **1000** | 73 | 73 | 172 | 172 | 9965.8 |
| **5000** | 437 | 468 | 1026 | 1026 | 61438.6 |
| **10000** | 910 | 918 | 2192 | 2192 | 132877.1 |
| **50000** | 5060 | 5200 | 10830 | 10830 | 780482 |
| **100000** | 10664 | 10744 | 21614 | 21614 | 1660964 |
| **Increasing Generated List:** | | | | | |
| **Inputs** | **Time1(mS)** | **Time2(mS)** | **Time3(mS)** | **Time Median** | **Comparison** |
| **100** | 9 | 9 | 9 | 9 | 356 |
| **500** | 36 | 36 | 36 | 36 | 2272 |
| **1000** | 98 | 73 | 73 | 73 | 5044 |
| **5000** | 468 | 417 | 468 | 468 | 32004 |
| **10000** | 918 | 907 | 1012 | 918 | 69008 |
| **50000** | 5270 | 5200 | 5105 | 5200 | 401952 |
| **100000** | 10828 | 10744 | 10701 | 10744 | 853904 |

**Table 15** merge sort Experimental Run time results

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Heap Sort** | | | | | |
| **Random Generated List:** | | | | | |
| **Inputs** | **Time1(mS)** | **Time2(mS)** | **Time3(mS)** | **Time Median** | **Comparison** |
| **100** | 13 | 10 | 15 | 15 | 664.4 |
| **500** | 78 | 81 | 91 | 91 | 4483 |
| **1000** | 167 | 173 | 201 | 201 | 9965.8 |
| **5000** | 956 | 966 | 1234 | 1234 | 61438.6 |
| **10000** | 2016 | 2019 | 2708 | 2708 | 132877.1 |
| **50000** | 11212 | 11341 | 14626 | 14626 | 780482 |
| **100000** | 23504 | 24020 | 24460 | 24460 | 1660964 |
| **Increasing Generated List:** | | | | | |
| **Inputs** | **Time1(mS)** | **Time2(mS)** | **Time3(mS)** | **Time Median** | **Comparison** |
| **100** | 10 | 13 | 10 | 10 | 805 |
| **500** | 81 | 82 | 81 | 81 | 5814 |
| **1000** | 173 | 173 | 172 | 173 | 13145 |
| **5000** | 955 | 966 | 974 | 966 | 83378 |
| **10000** | 2071 | 2019 | 1996 | 2019 | 181945 |
| **50000** | 7260 | 11361 | 11341 | 11341 | 1082517 |
| **100000** | 17108 | 24388 | 24020 | 24020 | 2320906 |

**Table 16** heap sort Experimental Run time results

1. Members' participation:

here’s a detailed **table 14** for each group member’s participation on this project:

|  |  |
| --- | --- |
| **Member name:** | **Task:** |
| Wadha Nayef Alsheddi | -Millstone 1: all members have participated in all parts.   * Millstone 2: Which of the three sorts seems to perform the best + excel file   Millstone 3: report + algorithm calculation   * all member has done the revision |
| Fatimah Fadel Alharz | -Millstone 1: all members have participated in all parts.  -Millstone 2: introduction + Theoretical question  -Millstone 3: writing algorithm + calculating run time analysis of each approach  - all member has done the revision |
| Amal Mohammed Alotaibi | -Millstone 1: all members have participated in all parts.  -Millstone 2: coding + Introduction to each sorting algorithm  -Millstone 3: presentation +helping members  - all member has done the revision |
| Asma Zaher Alshehri | -Millstone 1: all members have participated in all parts.  -Millstone 2: a comparison a good choice of basic operation for analyzing these algorithms + excel file  -Millstone 3: excel graphs + project poster  - all member has done the revision |
| Dema Hamed Alghamdi | -Millstone 1: all members have participated in all parts.  -Millstone 2: Data generation and experimental setup + coding  -Millstone 3: writing algorithm +calculating run time analysis of each approach  - all member has done the revision |

**Table 18 members tasks**