جامعة الإمام عبد الرحمن بن فيصل
IMAM ABDULRAHMAN BIN FAISAL UNIVERSITY

Kingdom of Saudi Arabia
Ministry of Education
Imam Abdulrahman bin Faisal University
Computer Science department
College of Science and Humanities

CS 412 Algorithm Analysis and Design (Term 1, 2020-2021)

Analyze Different Sorting Algorithms Theoretically and Experimentally by Using C++ Language Code.

By:

Nawal Awadh Alamri

Masomah Alshaer

Wasan Alshehri

Dana Kurdi

Albandari Alsanhani

Supervised By:
Dr. Azza Ahmed Ali.

Table of Content

Table of Tables:

Tables of Figures

Introduction

Any programming language, including C++, Java, JavaScript, and C#, requires you to first develop an algorithm for the code in order to achieve the best possible code structure. An algorithm is a group of steps used to address a challenge. The employment of an algorithm by the programmer for a better solution increases efficiency and offers clarity [1]. These structures also support a wide range of operations. Traversal, Insertion, Deletion, Searching, Sorting, and Merging are a few examples. This paper will explore many types of sorts, including increase order, decrease order, and random order, that are used to arrange data in clearly defined order. which are: heap sort, merge sort, and insertion sort. Sorting algorithms are important because can reduce the complexity of a problem and increase the efficiency performance.

1. Theoretical question

This section presents the three sorts explanation, algorithm, and time complexity for each case.

1.1. Insertion sort

Its sorting algorithm is the simplest [2]. The data appear to be split into two subarrays. While the left array is sorted, the right array is not. The first element in the left array should be compared to the elements in the right array before being correctly inserted. **Figure - 1-** shows insertion sort algorithm.

```
INSERTION-SORT(A)
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into the sorted sequence A[1 . . j − 1].
4      i = j − 1
5      while i > 0 and A[i] > key
6          A[i + 1] = A[i]
7          i = i − 1
8      A[i + 1] = key
```

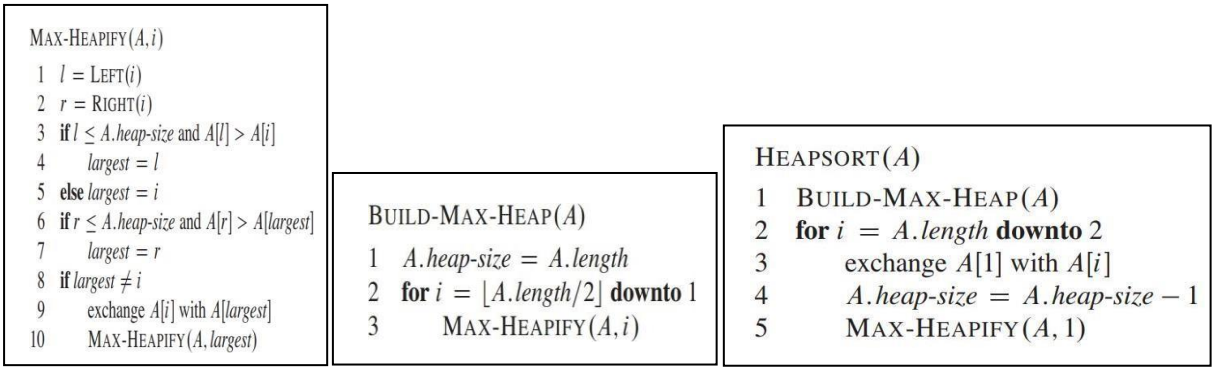
Figure 1: Insertion sort algorithm

	Best Case	**Average Case**	**Worst Case**
Data Order	Ordered Increasingly	Half Ordered Increasingly	Ordered Decreasingly
Time complexity	T(n) = Q(n)	T(n)= $Q(n^2)$	T(n)= $Q(n^2)$

Table 1: Insertion sort information.

1.2. Heap sort

We shall apply comparison in the binary heap data structure-based heap sort. Like selection sort, the minimum element is located first and placed at the beginning. This process will be repeated until all the elements are sorted. **Figure -2-** shows the heap sort algorithm.



```
MAX-HEAPIFY(A, i)
1  l = LEFT(i)
2  r = RIGHT(i)
3  if l ≤ A.heap-size and A[l] > A[i]
4      largest = l
5  else largest = i
6  if r ≤ A.heap-size and A[r] > A[largest]
7      largest = r
8  if largest ≠ i
9      exchange A[i] with A[largest]
10     MAX-HEAPIFY(A, largest)
```

```
BUILD-MAX-HEAP(A)
1  A.heap-size = A.length
2  for i = ⌊A.length/2⌋ downto 1
3      MAX-HEAPIFY(A, i)
```

```
HEAPSORT(A)
1  BUILD-MAX-HEAP(A)
2  for i = A.length downto 2
3      exchange A[1] with A[i]
4      A.heap-size = A.heap-size − 1
5      MAX-HEAPIFY(A, 1)
```

Figure 2: heap sort algorithm

	Best Case	Average Case	Worst Case
Data Order	Ordered Decreasingly	Ordered Increasingly	Half Ordered Increasingly
Time complexity	T(n) = O (n lg n)	T(n) = O (n lg n)	T(n) = O (n lg n)

Table 2: Heap sort information.

1.4. Merge sort

It is a form of data sorting technique used to arrange data in various ways. Divide an unsorted array into smaller, successive arrays until you reach an array with just one element by using the merge sort method. After that, sort each array and merge them one after the other, continuing until you get the sorted array [3]. merge sorts using the Divide and Conquer strategy. The concept of Divide and Conquer involves three steps [2]: **Figure -3-** shows the merge sort algorithm.

1. **Divide:** the problem into subproblems.
2. **Conquer:** the subproblems by solving them.
3. **Combined:** the solutions of the subproblem**.**

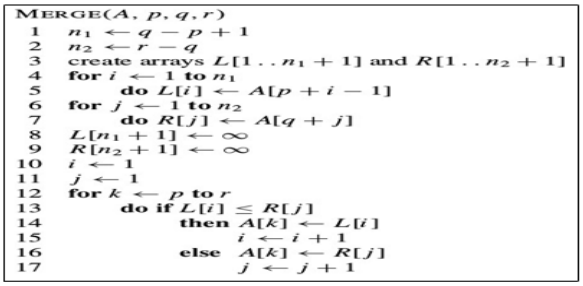


Figure 3 Merge sort

	Best Case	Average Case	Worst Case
Data Order	Ordered Decreasingly	Ordered Increasingly	Half Ordered Increasingly
Time complexity	T(n)= Q(n log n)	T(n)= Q(n log n)	T(n)= Q(n log n)

Table 3: Merge sort information.

2. Data generation and experimental setup.

This section contains subsection which are description of the characteristics of the machine used, timing mechanism, how the input is selected and what is the input for each sort.

2.1. Description of the characteristics of the machine used.

This subsection contains a description of the devices' s characteristics. This project compares between 2 devices characteristics .

# Of PC	PC 1	PC 2
Characteristics		
OS	Windows 11	MacOS Big Sur
Processor	Intel core i5	Apple M1 chip with 8-core CPU, and 16 core Neural Engine
System type	64-bit	64-bit

Table 4 characteristic Description of the machines.

2.2. Timing mechanism

This project used :: now() function, this function in c++ language returns the current time which is Microsecond

```
auto start = high_resolution_clock::now();
```

```
auto stop = high_resolution_clock::now();

auto duration = duration_cast<microseconds>(stop - start);

cout << "Time taken by merge function : "<< duration.count() << "
microseconds";
```

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

We use input 10000,15000,20000,25000,32000 by adding 5000 five time. We use these number because it is big enough to get running time for each algorithm. We stop at value 32000 because bigger number its cause the program to lag.

2.4 What times are reported?

We calculate the time by microseconds unit in each algorithm

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

The input was numbers, and we used a random sequence that generates using the rand() function

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

Yes, cause its fair enough to compare between all type of sorting algorithm and to get accurate result.

3. Selection of the best performance version of quick sort?

3.1 Graph the best-case running time as a function of input size n for the three sorts:

The **Figure (4)** below presents the case of the best running time. In this system, the number of the elements are presenting the x axis while the y axis covers different parameters time such as heapsort running time, insertion sort running time and Merge sort running time. It is sure from the figure the value of elements is increasing with the passage of the time.

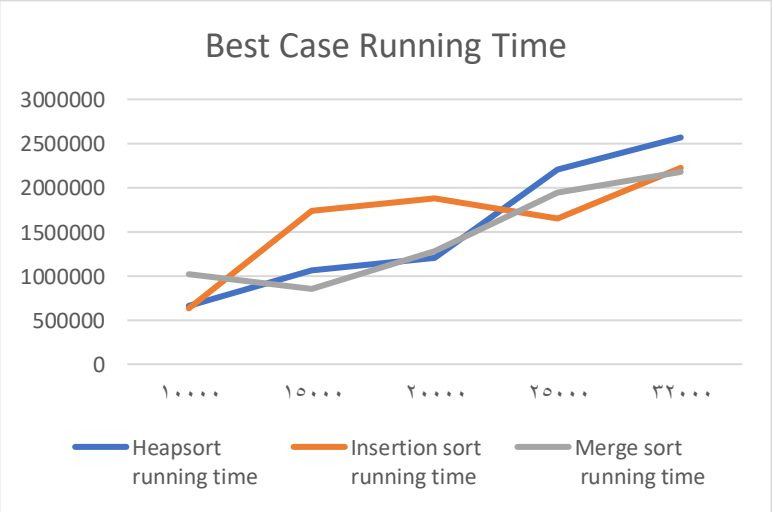


Figure 4 Best case running time

3.2 Graph the worst-case running time as a function of input size n for the three sorts:

For the scenario of the worst-case running time, the values of the elements have a huge gap, and this means that variation in the y axis considering time creates a huge confusion in such scenario. **Figure (5)**

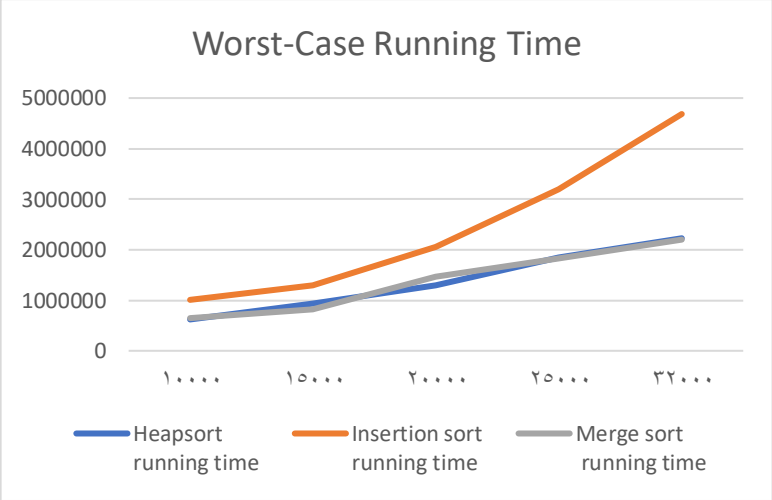


Figure 5 worst case running time

3.3 Graph the average-case running time as a function of input size n for the three sorts:

It can be seen from the **Figure (6)** that the average case running time graph has no such difference from the other graphs because it is the collective graph presenting the information of the other graphs. The information here explains the number of elements on the x-axis and the time element at the y axis.

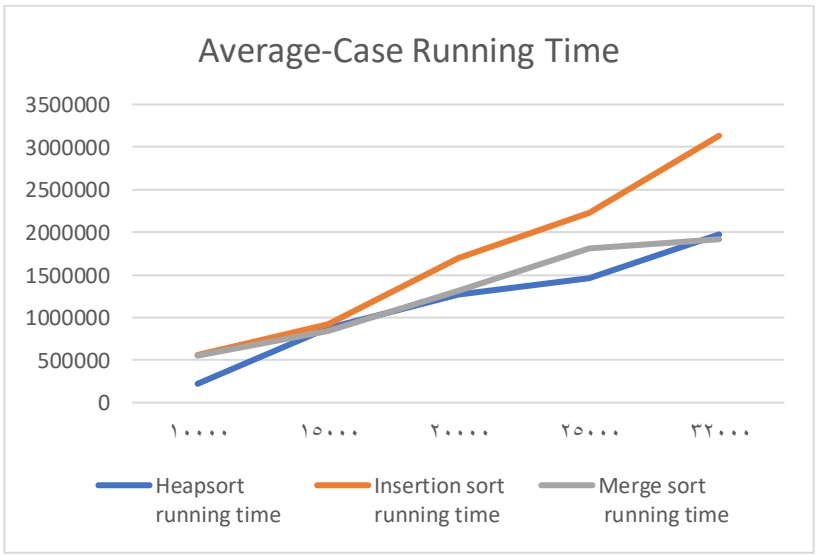


Figure 6 Average case running time

3.4 Discussion of the best performance sorting algorithm:

According to the deep analysis of the sort algorithm from the figure it can be analyzed that the merge sort running time is conisdered as the best sort because it is distinguishable in all of the three different scenarios and the values conisderably remains stable throught the systematic approach applied. The three different scenarios present a whole different kind of nature for the parameters of the time and sorting. The best as per data records is the merge sorting.

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

This section a comparison of the asymptotic of the theorical running time (T) with the experimental running time (P) by dividing them to find the difference rate (P/T) and draw graphs.

4.1 Insertion Sort

The table below show the comparison between theoretical and experimental result in the insertion sort for the cases which are best, worst, and average case

	Worst case (Decrease order)			Average case (Random order)			Bast cases (increase order)		
	T	P	P/T	T	P	P/T	T	P	P/T
10000	100,000,000 ms	148.086 ms	0.000001481 ms	100,000,000 ms	109.816 ms	0.000001098 ms	10000 ms	15.658 ms	0.001565800 ms
15000	225,000,000 ms	316.64 ms	0.000001407 ms	225,000,000 ms	189.798 ms	0.000000844 ms	15000 ms	20.659 ms	0.001377267 ms
20000	400,000,000 ms	551.227 ms	0.000001378 ms	400,000,000 ms	308.331 ms	0.000000771 ms	20000 ms	23.022 ms	0.001151100 ms
25000	625,000,000 ms	850.774 ms	0.000001361 ms	625,000,000 ms	453.528 ms	0.000000726 ms	25000 ms	26.984 ms	0.001079360 ms
32000	1,024,000,000 ms	1381.194 ms	0.000001349 ms	1,024,000,000 ms	715.63 ms	0.000000699 ms	32000 ms	33.923 ms	0.001060094 ms

Table (5) Insertion Sort Theory Vs. Experimental

The below **Figure (7)** represent the comparison of the insertion sort best-case analysis.

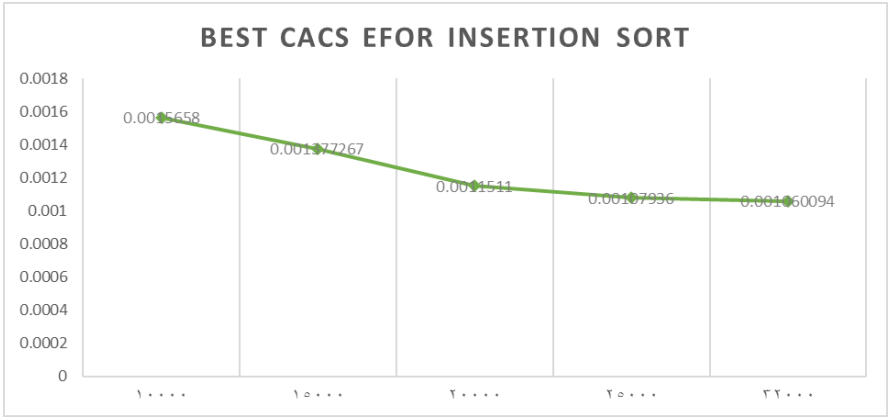


Figure (7) Insertion sort best case

The below **Figure (8) the** comparison of the insertion sort average-case analysis. The graph below shows the comparison of theoretical and practical average case of insertion sort.

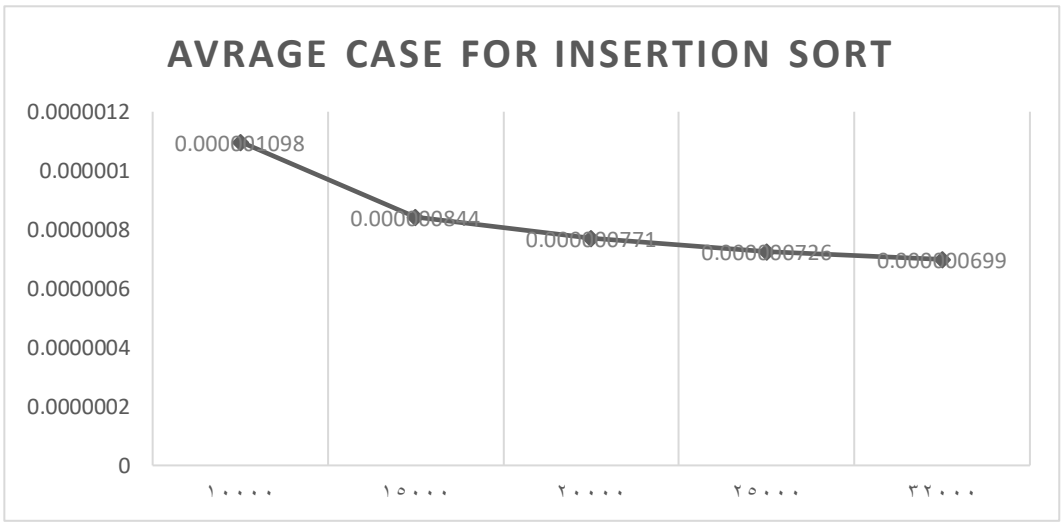


Figure (8) Insertion sort average case

The below **Figure (9)** present the comparison of the insertion sort worst-case analysis. The graph below shows the comparison of theoretical and practical worst case of insertion sort.

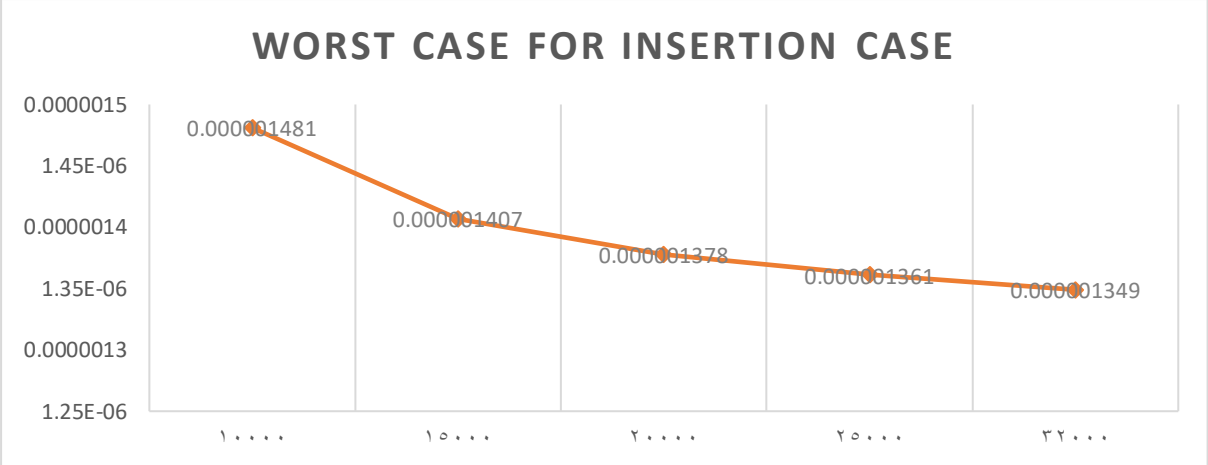


Figure (9) Insertion sort worst case

4.2 Merge Sort

The table show the comparison between theorical and experimental result in the Merge sort for the cases which are best, worst, and average case.

	Worst case (Decrease order)			Average case (Random order)			Bast cases (increase order)		
	T	P	P/T	T	P	P/T	T	P	P/T
10000	40000 ms	25.473ms	0.000636825 ms	40000 ms	33.031 ms	0.000825775 ms	40000 ms	20.513 ms	0.000512825 ms
15000	62641 ms	33.441ms	0.000533852 ms	62641 ms	34.435 ms	0.000549720 ms	62641 ms	34.867 ms	0.000556616 ms
20000	86021 ms	40.562ms	0.000471536 ms	86021 ms	46.638 ms	0.000542170 ms	86021 ms	45.183 ms	0.000525255 ms
25000	109949 ms	48.462ms	0.000440768 ms	109949 ms	55.743 ms	0.000506990 ms	109949 ms	46.206 ms	0.000420249 ms
32000	144165 ms	52.85ms	0.000366594 ms	144165 ms	46.333 ms	0.000321389 ms	144165 ms	51.108 ms	0.000354510 ms

Table (6) Merge Sort Theory Vs. Experimental

The below **Figure (10) present** the comparison of the merge sort best-case analysis. The graph below shows the comparison of theoretical and practical best case of merge sort.

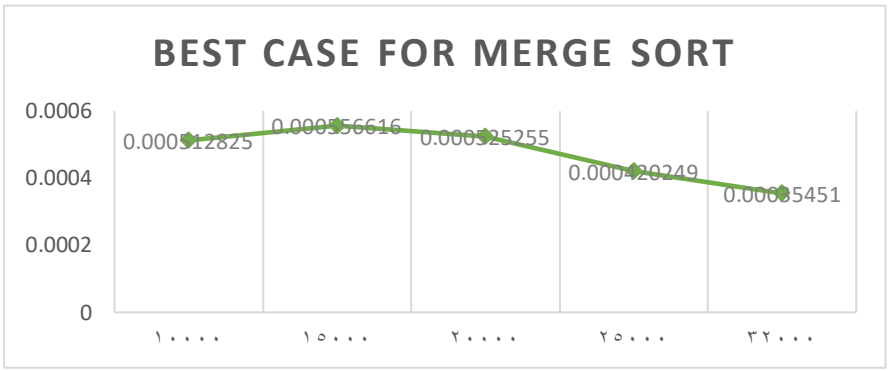


Figure (10) Merge sort best case

The below **Figure (11)** present the comparison of the merge sort average-case analysis. The graph below shows the comparison of theoretical and practical average case of merge sort.

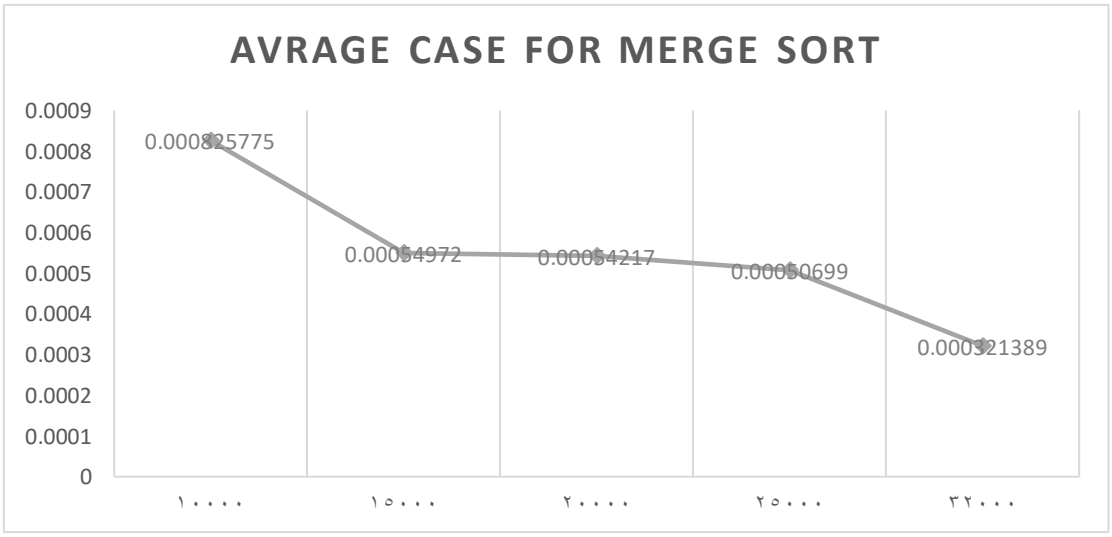


Figure (11) Merge sort average case

The below **Figure (12)** present the comparison of the merge sort worst-case analysis. The graph below shows the comparison of theoretical and practical worst case of merge sort**.**

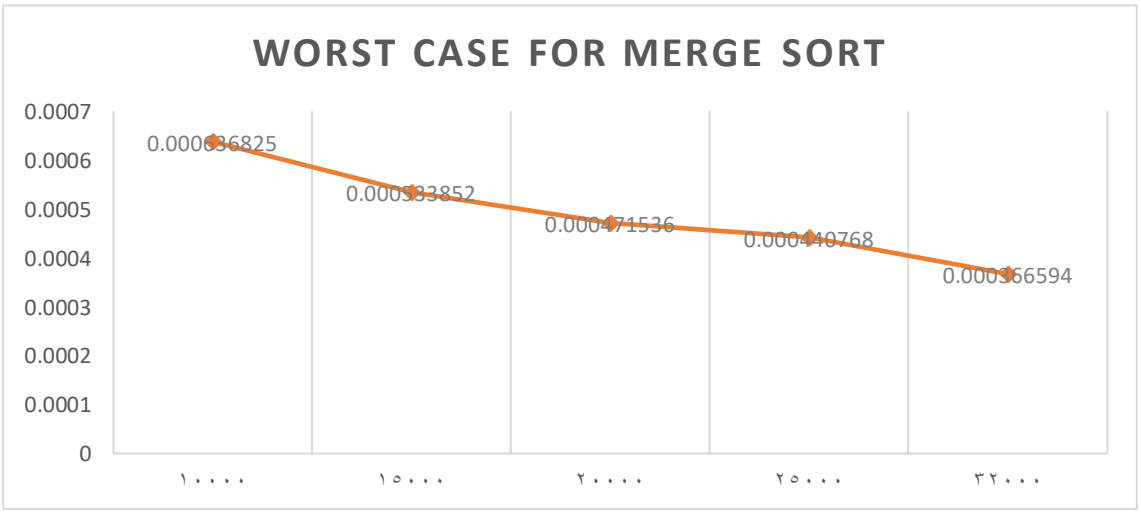


Figure (12) Merge sort worst case

4.3 Heap sort

The table show the comparison between practical and experimental result in the Heap sort for the cases which are best, worst, and average case.

	Worst case (Decrease order)			Average case (Random order)			Bast cases (increase order)		
	T	P	P/T	T	P	P/T	T	P	P/T
10000	40000 ms	27.72 ms	0.000693000 ms	40000 ms	10 ms	0.000250000 ms	40000 ms	26.924 ms	0.000673100 ms
15000	62641 ms	33.633 ms	0.000536917 ms	62641 ms	10.9 ms	0.000174007 ms	62641 ms	33.366 ms	0.000532654 ms
20000	86021 ms	41.99 ms	0.000488137 ms	86021 ms	11.306 ms	0.000131433 ms	86021 ms	41.32 ms	0.000480348 ms
25000	109949 ms	45.568 ms	0.000414447 ms	109949 ms	13.487 ms	0.000122666 ms	109949 ms	42.675 ms	0.000388134 ms
32000	144165 ms	53.478 ms	0.000370950 ms	144165 ms	17.948 ms	0.000124496 ms	144165 ms	50.105 ms	0.000347553 ms

Table (7) Heap Sort Theory Vs. Experimental

The below **Figure (13)** present the comparison of the Heap sort best-case analysis. The graph below shows the comparison of theoretical and practical best case of Heap sort.

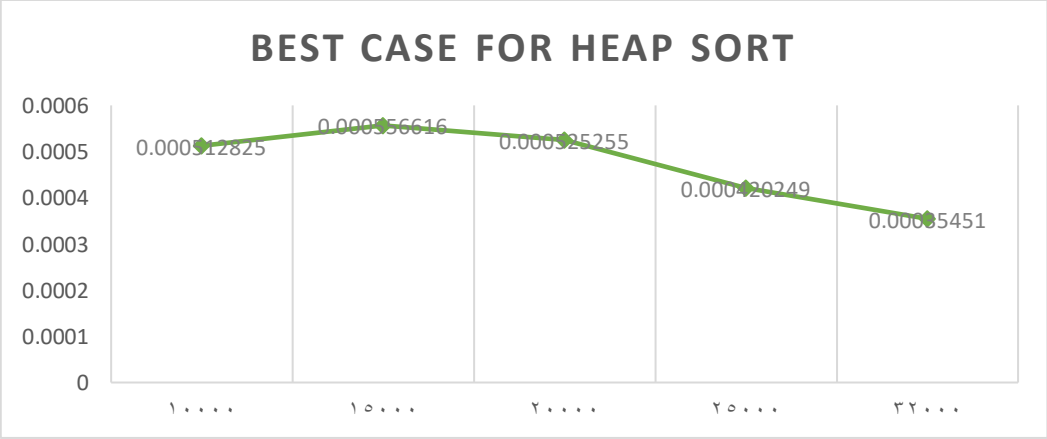


Figure (13) Heap sort best case

The below **Figure (14)** present the comparison of the Heap sort average-case analysis. The graph below shows the comparison of theoretical and practical average case of Heap sort.

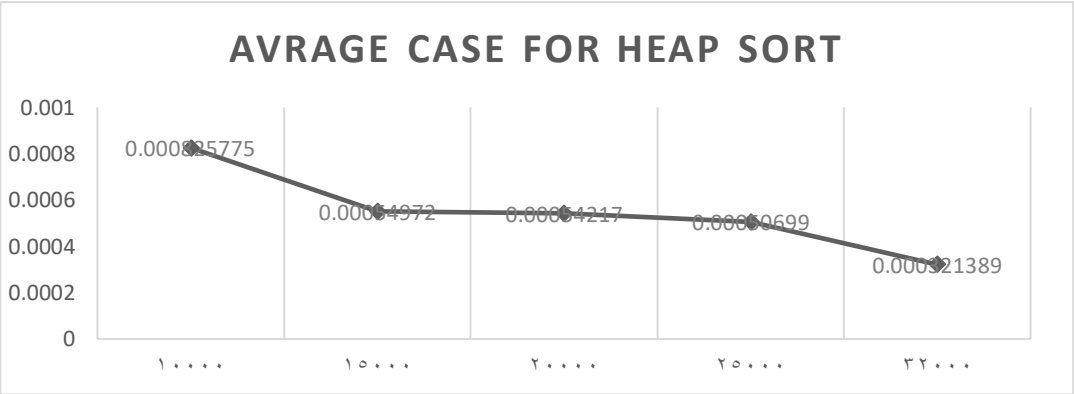
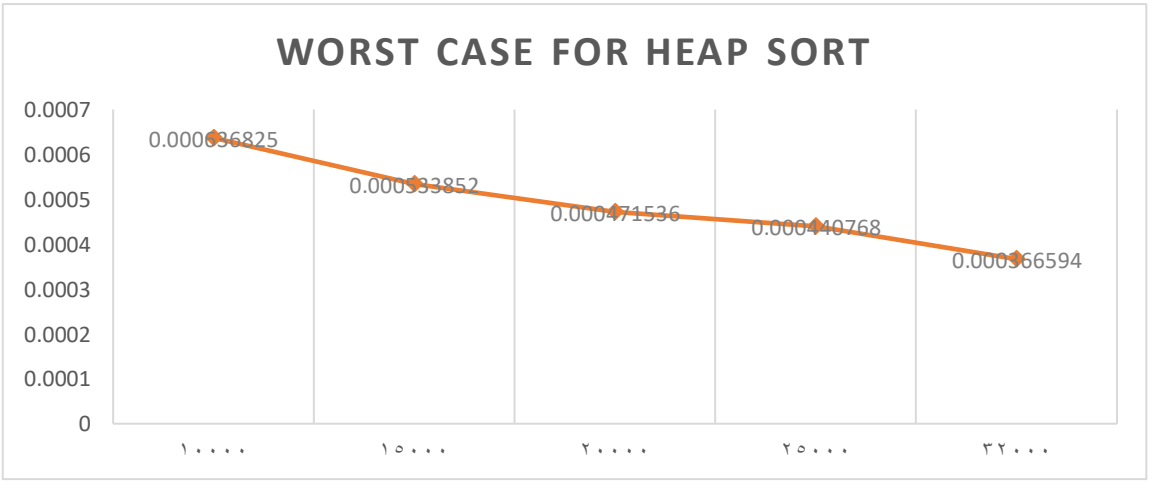


Figure (14)Heap sort average case

The below **Figure (15)** present the comparison of the Heap sort worst-case analysis. The graph below shows the comparison of theoretical and practical worst case of Heap sort



Figure(15)Heap sort worst case

4.2.1 Your determination should be backed up by your experiments and analysis and you must explain your reasoning. If you found the sort did not conform to the asymptotic analysis, you should try to understand why and provide an explanation.

The theoretical asymptotic analysis displays the longest possible time for each sort's instance. As a result, the theoretical analysis may be greater or less accurate than the experimental analysis. There are some variables that could impact the execution time for the various analyses, including CPU usage, compiler speed, and programming language. As expected, the experimental took less time than the theorical in this project.

5. 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?

Yes, the numbers of comparisons are a good predictor of the execution time as show in the **tables (8-9-10)** and **Figures (16-17-18).**

Number of elements	Heap running time	Heap real time	Number of Comparisons
10000	27.72	40000	40000
15000	33.633	62641	62641
20000	41.99	86021	86020
25000	45.568	109949	109948
32000	53.478	144165	144164

Table (8) Heap Comparison

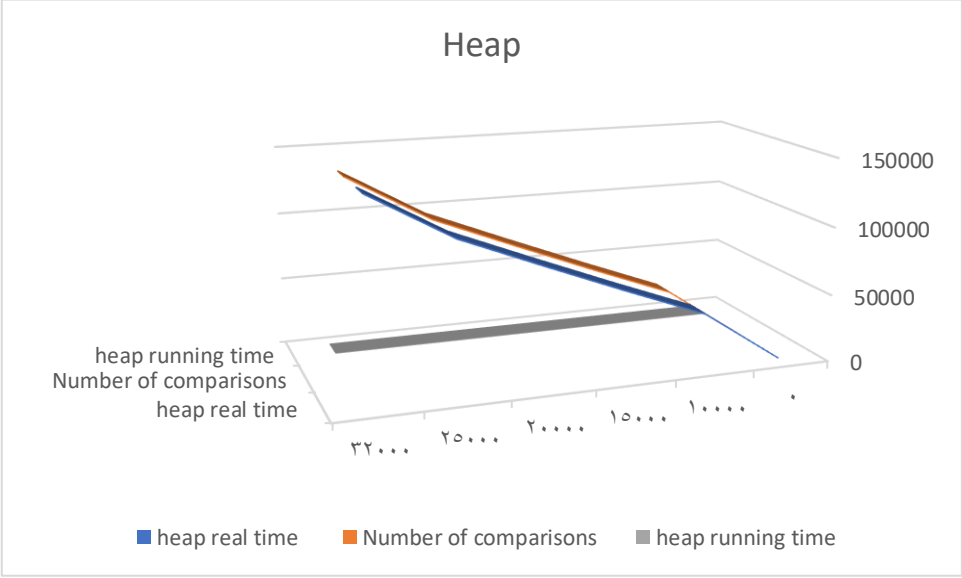


Figure (16) Heap Comparison

Number of elements	Insertion running time	Insertion real time	Number of Comparisons
10000	148.086	100,000,000	50000000
15000	316.64	225,000,000	112500000
20000	551.227	400,000,000	200000000
25000	850.774	625,000,000	312500000
32000	1381.194	1,024,000,000	512000000

Table (9) Insertion Comparison

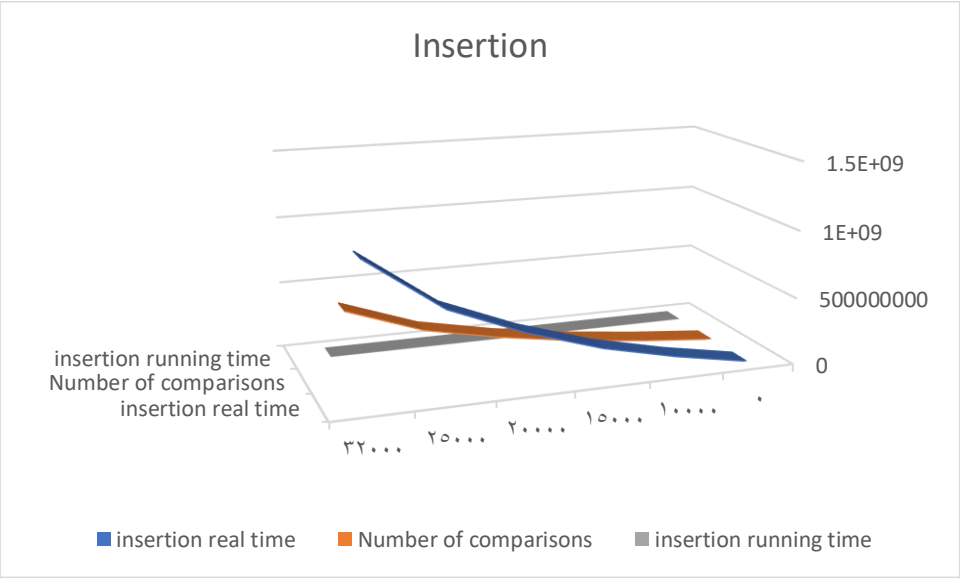


Figure (17) Insertion Comparison

Number of elements	Merge running time	Merge real time	Number of Comparisons
10000	25.473	40000	40000
15000	33.441	62641	62641
20000	40.562	86021	86020
25000	48.462	109949	109948
32000	52.85	144165	144164

Table (10) Merge Comparison

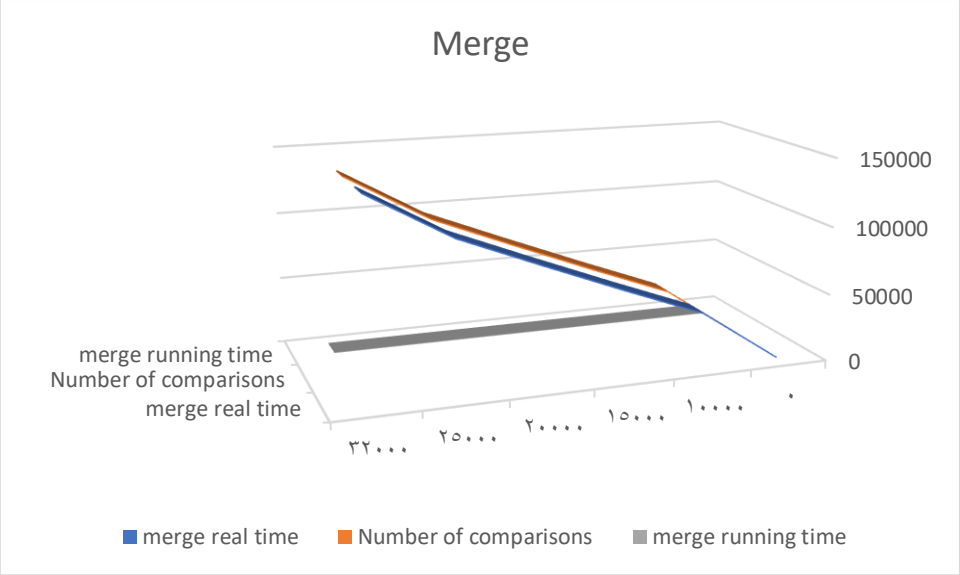


Figure (18) Merge Comparison

6. Part 2

Design and analysis an improved Divide-and-conquer algorithm compute a^n, where n is natural number.

A. Design and analysis and improve Divide-and-conquer algorithm compute a^n, where n is natural number. Efficient Algorithm and time complexity

Power-Divide-Conqure (a,n)	
1- if n==0	C1,1
2- return 1	C2,1
3-a==0	C3,1
4-return 0	C4,1
5-if n%2=0 //n is even	C5,1
6-power(a*a, a/2) //recursive	C6, T(n/2)
7-else //n is odd	C7,1
8- a*power(a*a , n/2)	C8, T(n/2)

Time Complexity : $T(n)=T(n/2)+c = O(\lg n)$

Space Complexity: $O(\lg n)$ for recursive call

B. Brute force Algorithm and time complexity:

1-Int base , exp ,i ,result =1;	C1,1
2-cout<<“Enter base and exponent”>;	C2,1
3-cin >>base>>exp;	C3,1
4-for(i=0; i< exp; i++)	C4,n
{	
5- result=result *base;	C5, n
}	
6-cout<<“base <<“^”<<exp<<“=“<< result;	C6,1
7-return 0 ;	C7,1
}	

Time Complexity: T(n)=T(n)+c =O(n)

7. Reference

1. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to algorithms. MIT Press, 2009.

2. Insertion sort. GeeksforGeeks. (2021, July 8). Retrieved November 27, 2021, from https://www.geeksforgeeks.org/insertion-sort/.

3. Wilkie, “Introduction to algorithms: Chapter Two, Merge Sort,” Medium, 31- Jan-2017. [Online]. Available: https://medium.com/craft- academy/introduction-to-algorithms-chapter-two-merge-sort-edc7aba8d0d9. [Accessed: 26-Nov-2021].