



HACETTEPE  
ÜNİVERSİTESİ

Computer Science Engineering Department

BBM204

Experiment 1

Analysis Of sorting Algorithms Java

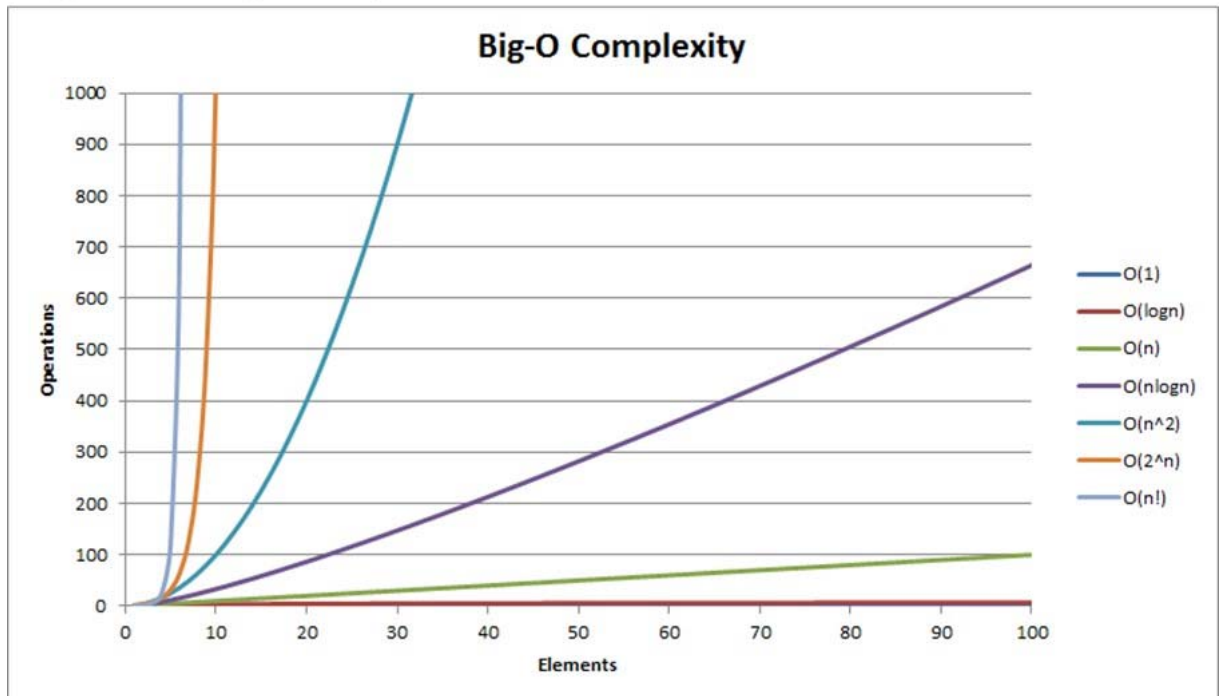
Ayoub Othman  
21202955

# 1. Introduction

In this experiment you expected from us coding a program which execute 5 algorithm / problem (Comb sort, Cocktail Shaker Sort, Gnome Sort, Bitonic Sort, Stooage Sort Algorithm) that determined before then measuring the required time for all of the algorithms according to given input size 'n' and plotting a graph to show performance with this 'n' values (It had to help us for making some observations)

In the and we had to analyze the time and space complexity of them. Here is a graph which is showing some curves of different big O values

## Big-O Complexity Chart



In this program, I have generated random numbers that are stored inside a array, dependent on the user to determine their number and the values confined between them. **to get worse case i reverse arrays after sorted and sort it again**

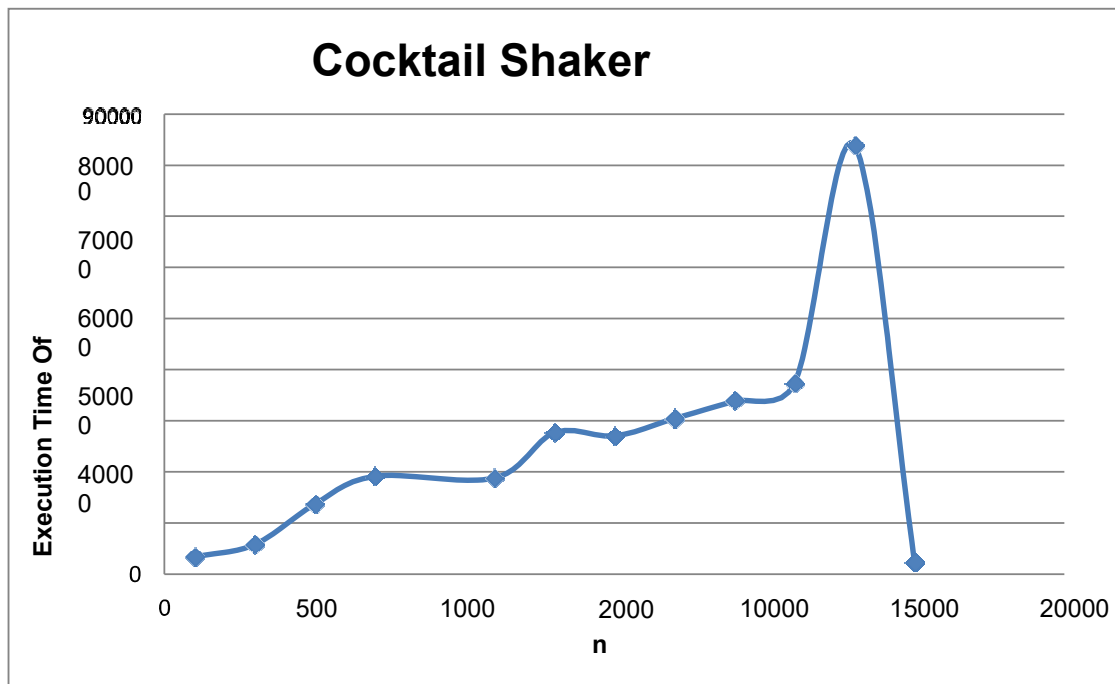
## 2. Problems

### 2.1. Cocktail Shaker Sort ( $O(n^2)$ )

**Cocktail shaker sort**, also known as **bidirectional bubble sort**, **cocktail sort**, **shaker sort** (which can also refer to a variant of selection sort), **ripple sort**, **shuffle sort**,<sup>[3]</sup> or **shuttle sort**, is an extension of bubble sort. The algorithm extends bubble sort by operating in two directions. While it improves on bubble sort by more quickly moving items to the beginning of the list, it provides only marginal performance improvements. This algorithm has worldwide usage so the method for the solution was the same in everywhere. Shortly this algorithm sorts an array by comparing every adjacent pair of elements of it, after that the biggest number goes to the end. Then the same operation is used again. This algorithm uses two for loops for traverse the array. Its time complexity is  $O(n^2)$  because we use nested two for loops (for a  $n$  size array). It is not useful for big data because of its time complexity. It may be useful for small sizes Here is my code piece:

```
class cocktailSort1
{
    void cocktailSort1(int nums[])
    {
        boolean swapped;
        do {
            swapped = false;
            for (int i = 0; i <= nums.length - 2; i++) {
                if (nums[i] > nums[i + 1]) {
                    //swap if two elements are in the wrong order
                    int temp = nums[i];
                    nums[i] = nums[i + 1];
                    nums[i + 1] = temp;
                    swapped = true;
                }
            }
            if (!swapped) {
                break;
            }
            swapped = false;
            for (int i = nums.length - 2; i >= 0; i--) {
                if (nums[i] > nums[i + 1]) {
                    int temp = nums[i];
                    nums[i] = nums[i + 1];
                    nums[i + 1] = temp;
                    swapped = true;
                }
            }
        } while (swapped);
    }
}
```

Graph Of Cocktail Shaker Sort

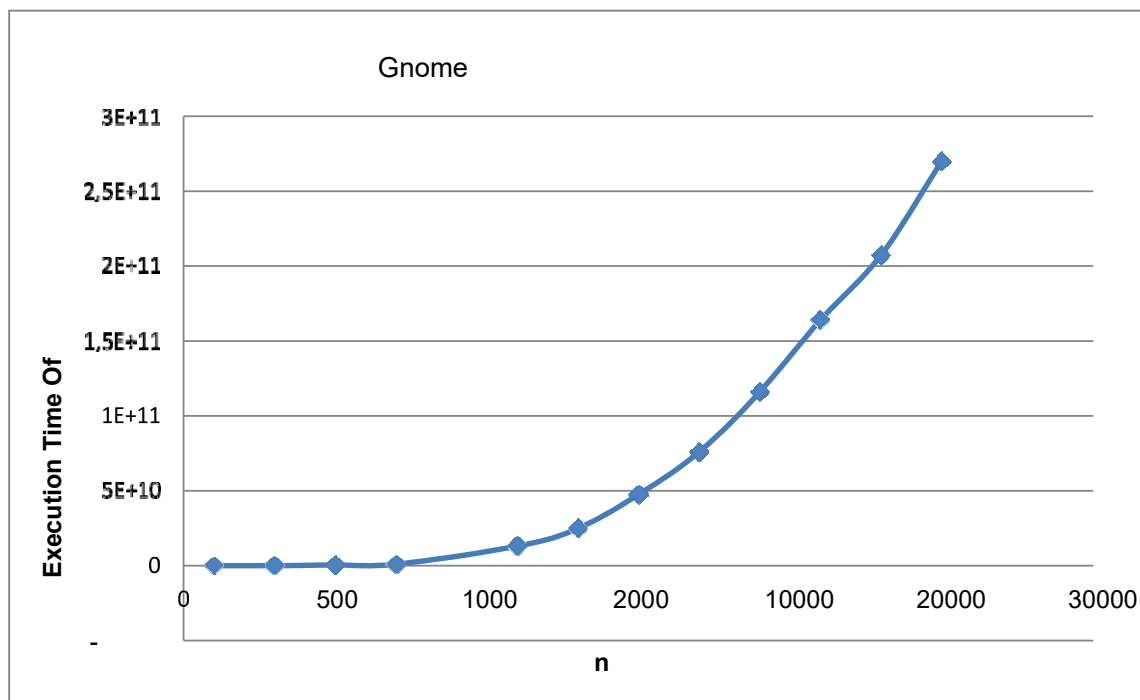


## 2.2. Gnome Sort Algorithm

### Explanations and Code Piece Of Gnome Sort Algorithm ( $O(n^2)$ )

**Gnome Sort** is extremely simple, and is very similar to the principle behind the **Insertion Sort**. The first principle to bear in mind when doing a Gnome Sort is that as the algorithm progresses, all items to the left of the current item are always in order. Each iteration of the algorithm moves the current item to the correct spot with respect to the previously sorted items.

The first part of the algorithm looks at the first two items in the data set and swaps them around if they are in the wrong order. Then, for each iteration thereafter, the next item is bubbled backwards towards to head of the list until a value is encountered which is less than the current item (or the head of the list is reached). At this point, the current item is left alone, and the sort continues from the next item.



## 2.3 Bitonic Sort Algorithm

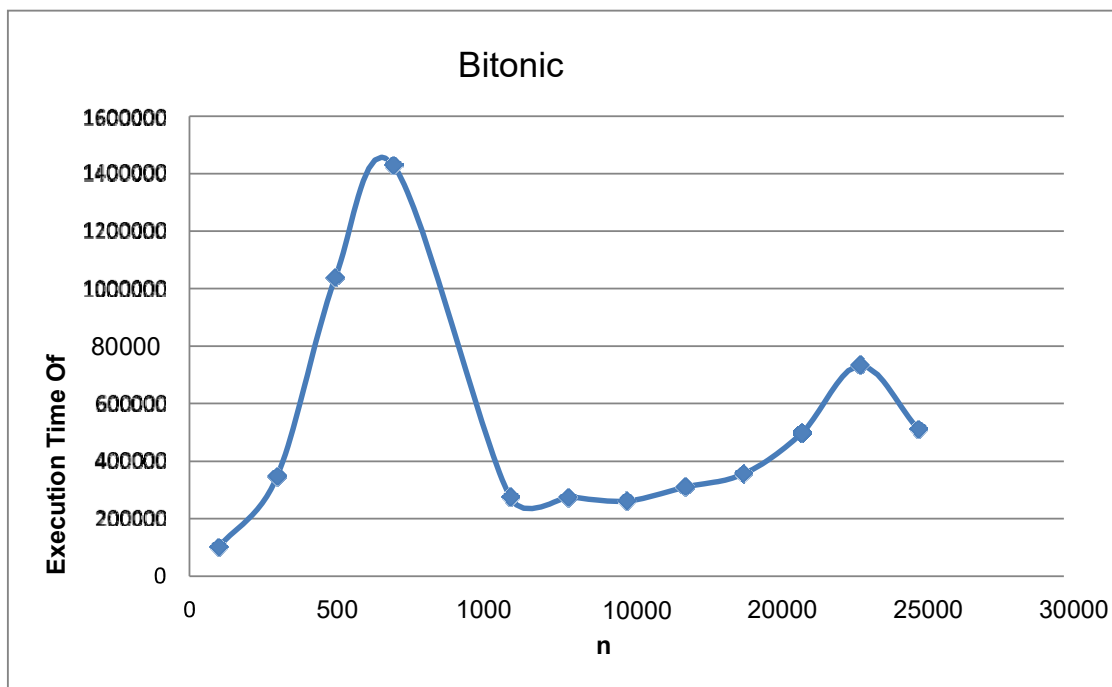
### Explanations Of Finding Bitonic Sort ( $O(n \log 2n)$ )

Bitonic Sort is a classic parallel algorithm for sorting.

Bitonic sort does  $O(n \log 2n)$  comparisons.

The number of comparisons done by Bitonic sort are more than popular sorting algorithms like Merge Sort [ does  $O(n \log n)$  comparisons], but Bitonic sort is better for parallel implementation because we always compare elements in predefined sequence and the sequence of comparison doesn't depend on data. Therefore it is suitable for implementation in hardware and parallel processor array

Graph Of Bitonic

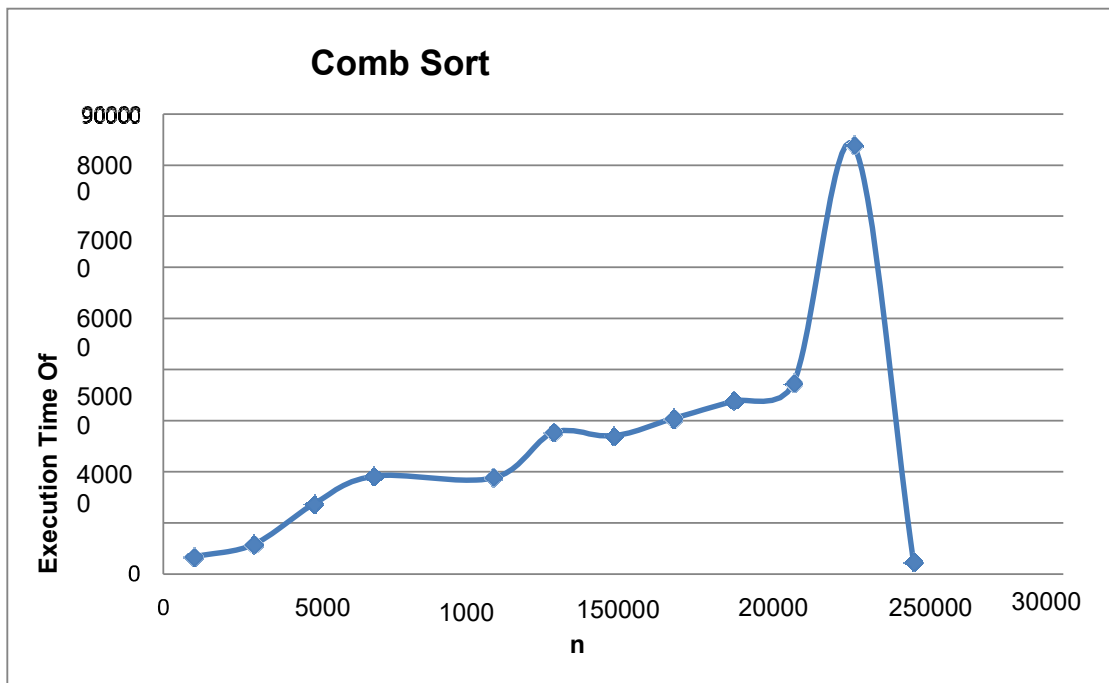


## 2.4. Comb Sort Algorithm

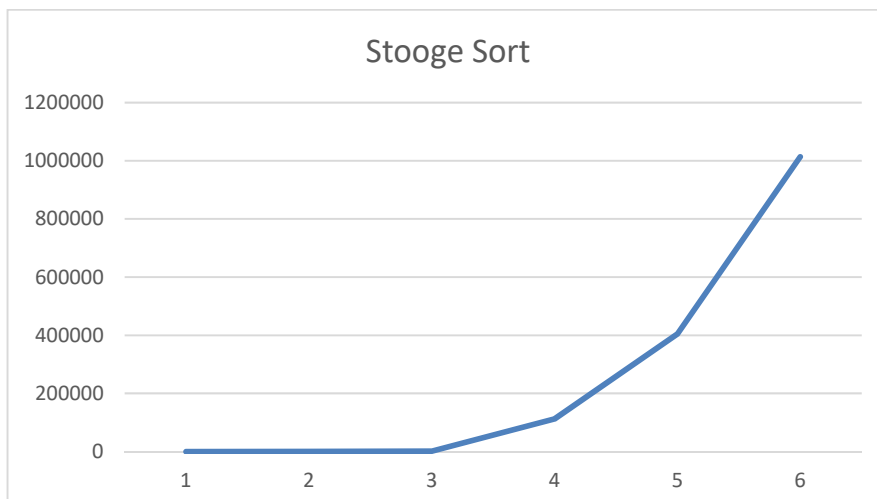
Explanations and Code Piece Of Comb Sort Algorithm ( $O(\log n)$ )

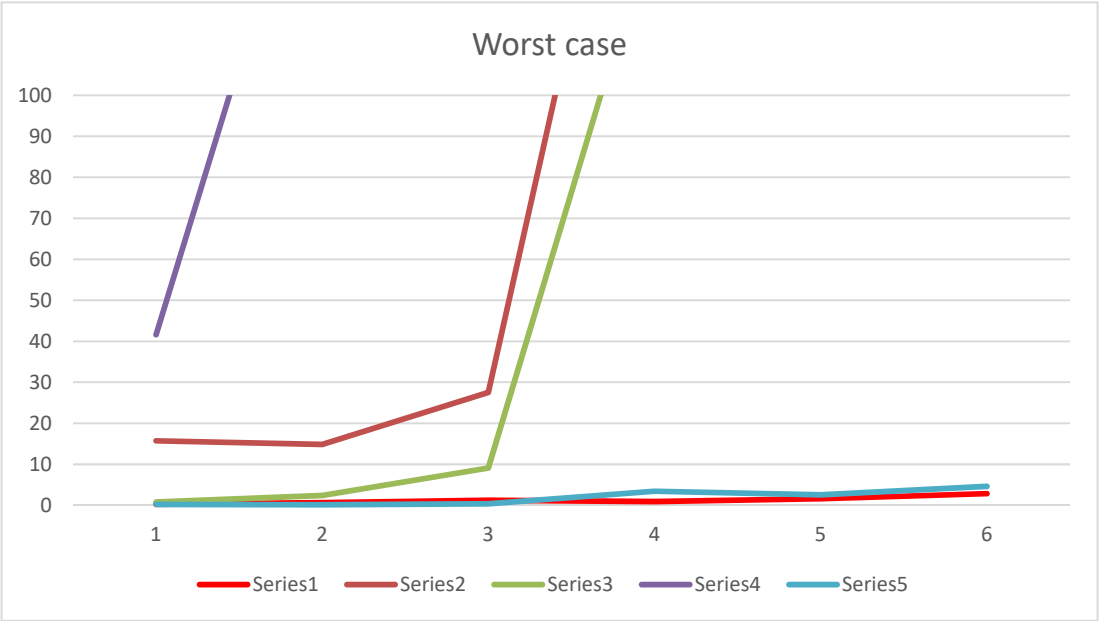
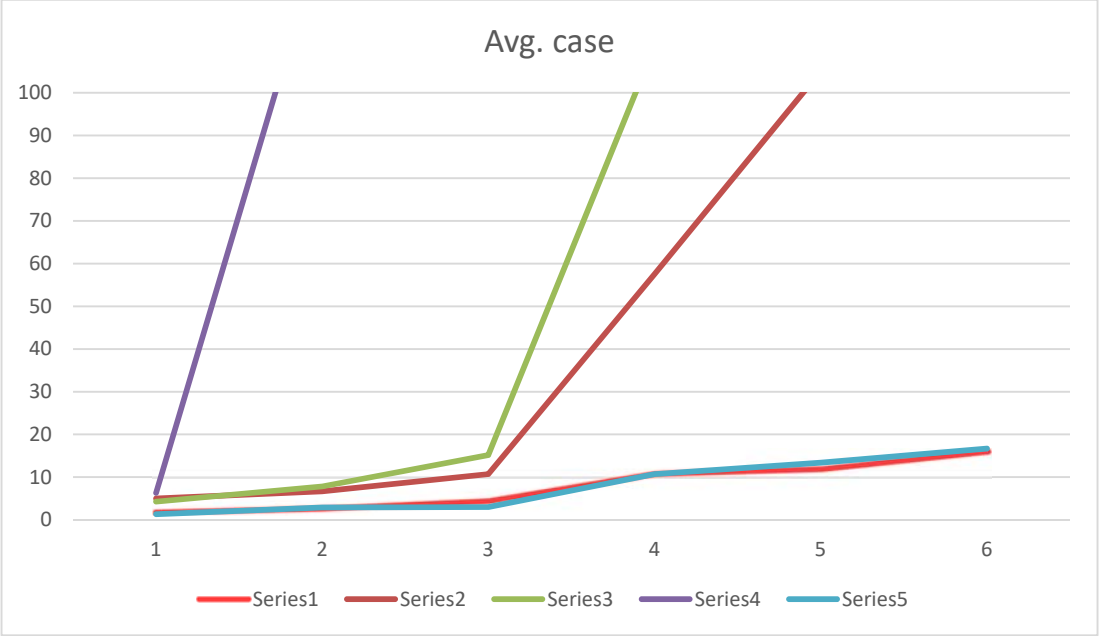
Comb Sort is mainly an improvement over Bubble Sort. Bubble sort always compares adjacent values. So all inversions are removed one by one. Comb Sort improves on Bubble Sort by using gap of size more than 1. The gap starts with a large value and shrinks by a factor of 1.3 in every iteration until it reaches the value 1. Thus Comb Sort removes more than one inversion counts with one swap and performs better than Bubble Sort.











Graph Of Comb Sort Algorithm



**Stooge Sort** is a recursive sorting algorithm. It is inefficient but interesting sorting algorithm. It divides the array into two overlapping parts ( $2/3$  each). Then it performs sorting in first  $2/3$  part and then it performs sorting in last  $2/3$  part. After that, sorting is done on first  $2/3$  part to ensure the array is sorted.





	A	B	C	D	E	F	G	H
1	Algorithms/n	500	1000	2000	10000	15000	20000	
2	Comb Sort	1.815012	2.710676	4.458976	10.84231	11.930209	16.01813	
3	Gnome Sort	4.993453	6.701491	10.716385	57.56169	105	249.8448	
4	Shaker Sort	4.258054	7.81505	15.187992	110.9115	259.583779	470.4121	
5	Stooge Sort	6.285159	136.8861	1243.78197	112656.3	404400.627	1013515	
6	Bitonic Sort	1.344482	2.891861	2.997651	10.70336	13.383245	16.69708	
7								
8								
9								
10								
11								
12								
13								
14								
15								
16								
17	Algorithms/n	500	1000	2000	10000	15000	20000	
18	Comb Sort	0.263686	0.667899	1.224482	0.914612	1.625931	2.848045	
19	Gnome Sort	15.72918	14.85404	27.52873	207.8907	317.974024	513.2378	
20	Shaker Sort	0.808032	2.420936	9.075848	143.4716	323.884062	547.1652	
21	Stooge Sort	41.60395	173.724	1730.91603	159974.3	271656.373	1250439	
22	Bitonic Sort	0.223818	0.14487	0.360397	3.446864	2.548043	4.610951	
23								
24								

## My Experiences and Opinions

### 1.1. Opinion

Although i found this experiment boring i knew it will be really profit to me in future because when i got a job i will be have to think about the space and time factors so I must to learn using of time and space efficiently while coding.

Shortly if this assignment is done conscious it would be absolutely helper I think.

### Experiences

Firstly, i want to mention about measuring time method which is used in this experiment. I had to measure only the time which is passed while the algorithm processing (the code piece which completely belong this algorithm). For this purpose i separated the other operations (generating randomNumbers, creation of some objects etc.) which has not direct connection with algorithm to different distinct methods so I could measure the time by putting the method call of algorithm between “start” and “end” values(with “system.nanoTime”)then subtracting end from start.



Another subject which i want to mention about is plotting graph. I was not know plotting graph on computer until this assignment but when i had to do this i watched video, did some research and learnt it. Again i think this will be help me in the future

Lastly; i studied the midterm subjects.learnt the time measuring methods and making observations from the graphs while dealing with this experiment. It was not hard and detailed but it was informative

