Mannan Abdul

21801066

CS 202

HW # 1

15/07/2020

Q1)

**Insertion Sort) Key: Key (Grey), Sorted elements (Black)**

Initial Array:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | 8 | 3 | 7 | 6 | 2 | 1 | 5 |

1st Swap:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | 8 | 3 | 7 | 6 | 2 | 1 | 5 |

2nd Swap:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 4 | 8 | 7 | 6 | 2 | 1 | 5 |

3rd Swap:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 4 | 7 | 8 | 6 | 2 | 1 | 5 |

4th Swap:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 4 | 6 | 7 | 8 | 2 | 1 | 5 |

5th Swap:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 3 | 4 | 6 | 7 | 8 | 1 | 5 |

6th Swap:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 6 | 7 | 8 | 5 |

Sorted Array:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

**Selection Sort) Key: Selected elements (Grey), Sorted elements (Black)**

Initial Array:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | 8 | 3 | 7 | 6 | 2 | 1 | 5 |

1st Swap:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | 5 | 3 | 7 | 6 | 2 | 1 | 8 |

2nd Swap:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | 5 | 3 | 1 | 6 | 2 | 7 | 8 |

3rd Swap:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | 5 | 3 | 1 | 2 | 6 | 7 | 8 |

4th Swap:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | 2 | 3 | 1 | 5 | 6 | 7 | 8 |

5th Swap:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

6th Swap:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

7th Swap:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

Sorted Array:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

**Bubble Sort) Key: Selected elements (Grey), Sorted elements (Black)**

Initial Array:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | 8 | 3 | 7 | 6 | 2 | 1 | 5 |

1st Pass:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | 8 | 3 | 7 | 6 | 2 | 1 | 5 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | 8 | 3 | 7 | 6 | 2 | 1 | 5 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | 3 | 8 | 7 | 6 | 2 | 1 | 5 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | 3 | 7 | 8 | 6 | 2 | 1 | 5 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | 3 | 7 | 6 | 8 | 2 | 1 | 5 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | 3 | 7 | 6 | 2 | 8 | 1 | 5 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | 3 | 7 | 6 | 2 | 1 | 8 | 5 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | 3 | 7 | 6 | 2 | 1 | 5 | 8 |

2nd Pass:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | 3 | 7 | 6 | 2 | 1 | 5 | 8 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 4 | 7 | 6 | 2 | 1 | 5 | 8 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 4 | 7 | 6 | 2 | 1 | 5 | 8 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 4 | 6 | 7 | 2 | 1 | 5 | 8 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 4 | 6 | 2 | 7 | 1 | 5 | 8 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 4 | 6 | 2 | 1 | 7 | 5 | 8 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 4 | 6 | 2 | 1 | 5 | 7 | 8 |

3rd Pass:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 4 | 6 | 2 | 1 | 5 | 7 | 8 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 4 | 6 | 2 | 1 | 5 | 7 | 8 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 4 | 6 | 2 | 1 | 5 | 7 | 8 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 4 | 2 | 6 | 1 | 5 | 7 | 8 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 4 | 2 | 1 | 6 | 5 | 7 | 8 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 4 | 2 | 1 | 5 | 6 | 7 | 8 |

4th Pass:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 4 | 2 | 1 | 5 | 6 | 7 | 8 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 4 | 2 | 1 | 5 | 6 | 7 | 8 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 2 | 4 | 1 | 5 | 6 | 7 | 8 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 2 | 1 | 4 | 5 | 6 | 7 | 8 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 2 | 1 | 4 | 5 | 6 | 7 | 8 |

5th Pass:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 2 | 1 | 4 | 5 | 6 | 7 | 8 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 3 | 1 | 4 | 5 | 6 | 7 | 8 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 1 | 3 | 4 | 5 | 6 | 7 | 8 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 1 | 3 | 4 | 5 | 6 | 7 | 8 |

6th Pass:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 1 | 3 | 4 | 5 | 6 | 7 | 8 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

Sorted Array:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

**Merge Sort)**

Mergesort calls:

Merge calls:

Sorted: Black

Initial Array:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | 8 | 3 | 7 | 6 | 2 | 1 | 5 |

|  |  |  |  |
| --- | --- | --- | --- |
| 4 | 8 | 3 | 7 |

|  |  |  |  |
| --- | --- | --- | --- |
| 6 | 2 | 1 | 5 |

|  |  |
| --- | --- |
| 3 | 7 |

|  |  |
| --- | --- |
| 6 | 2 |

|  |  |
| --- | --- |
| 1 | 5 |

|  |  |
| --- | --- |
| 4 | 8 |

5

1

2

6

4

4

4

4

7

3

8

4

|  |  |
| --- | --- |
| 4 | 8 |

|  |  |
| --- | --- |
| 2 | 6 |

|  |  |
| --- | --- |
| 1 | 5 |

|  |  |
| --- | --- |
| 3 | 7 |

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 5 | 6 |

|  |  |  |  |
| --- | --- | --- | --- |
| 3 | 4 | 7 | 8 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

**Quick Sort) Key: Unknown (grey), Sorted (black)**

Quicksort calls:

Partition:

Pivot:

Initial Array:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | 8 | 3 | 7 | 6 | 2 | 1 | 5 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | 8 | 3 | 7 | 6 | 2 | 1 | 5 |

**S2**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | 8 | 3 | 7 | 6 | 2 | 1 | 5 |

**S1  S2**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | 3 | 8 | 7 | 6 | 2 | 1 | 5 |

**S1  S2**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | 3 | 8 | 7 | 6 | 2 | 1 | 5 |

**S1  S2**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | 3 | 8 | 7 | 6 | 2 | 1 | 5 |

**S1  S2**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | 3 | 2 | 7 | 6 | 8 | 1 | 5 |

**S1  S2**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | 3 | 2 | 1 | 6 | 8 | 7 | 5 |

**S1  S2**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | 3 | 2 | 1 | 6 | 8 | 7 | 5 |

**S1  S2**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 3 | 2 | 4 | 6 | 8 | 7 | 5 |

|  |  |  |
| --- | --- | --- |
| 1 | 3 | 2 |

|  |  |  |  |
| --- | --- | --- | --- |
| 6 | 8 | 7 | 5 |

|  |  |  |
| --- | --- | --- |
| 1 | 3 | 2 |

|  |  |  |  |
| --- | --- | --- | --- |
| 6 | 8 | 7 | 5 |

S2 S2

|  |  |  |
| --- | --- | --- |
| 1 | 3 | 2 |

|  |  |  |  |
| --- | --- | --- | --- |
| 6 | 8 | 7 | 5 |

S2 S2

|  |  |  |
| --- | --- | --- |
| 1 | 3 | 2 |

|  |  |  |  |
| --- | --- | --- | --- |
| 6 | 8 | 7 | 5 |

S1 S2

|  |  |
| --- | --- |
| 3 | 2 |

|  |  |  |  |
| --- | --- | --- | --- |
| 6 | 5 | 7 | 8 |

S1 S2

|  |  |  |  |
| --- | --- | --- | --- |
| 5 | 6 | 7 | 8 |

More recursions to take place but the sub list has been sorted and therefore no use to show that.

|  |  |
| --- | --- |
| 3 | 2 |

S1

|  |  |
| --- | --- |
| 3 | 2 |

S1

|  |  |
| --- | --- |
| 2 | 3 |

Sorted Array:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

Q2)

Merge Sort recurrence relation:

T(1) = O(1)

Base Recurrence Equations

T(n) = 2T(n/2) + O(n)

We can write O(n) as simply n and O(1) simply as 1.

First, we find T(n/2).

T(n/2) = 2T(n/22) + n/2 putting into base recurrence equation.

T(n) = 2[2T(n/22) + n/2] + n simplifying the equation

T(n) = 22T(n/22) + n + n we then find T(n/22) and put into this equation

T(n/22) = 2T(n/23) + n/22

T(n) = 22[2T(n/23) + n/22] +n + n simplifying again

T(n) = 23T(n/23) + n + n + n we can see a pattern emerging, so we write it in that form

Assuming we do the substitution k times, we have

T(n) = 2kT(n/2k) + kn

Now, to get the base case, we equate T(n/2k) = T(1)

n/2k = 1 => n = 2k => k = log2n using this value of k in the general equation

T(n) = 2log2n \* 1 + nlog2n => n + nlog2n

Therefore, we have **O(nlog2n)** by solving the recurrence relation.

Quick Sort Recurrence Relation:

T(0) = T(1) = 0

Base Recurrence Equations

T(n) = T(n - 1) + O(n)

We can write O(n) simply as n

We find T(n - 1),

T(n - 1) = T(n - 2) + n – 1 Substituting into base equation

T(n) = [T(n - 2) + n - 1] + n Simplifying

T(n) = T(n - 2) + n + n – 1 Finding T(n - 2)

T(n - 2) = T(n - 3) + n – 2 putting into equation of T(n)

T(n) = [T(n - 3) + n - 2] + n + n - 1 Simplifying

T(n) = T(n - 3) + n + n + n - 1 – 2 If we do this k times, we have the general form

T(n) = T(n - k) + kn - Solving for base case, we have

n – k = 0 => n = k Putting in general equation

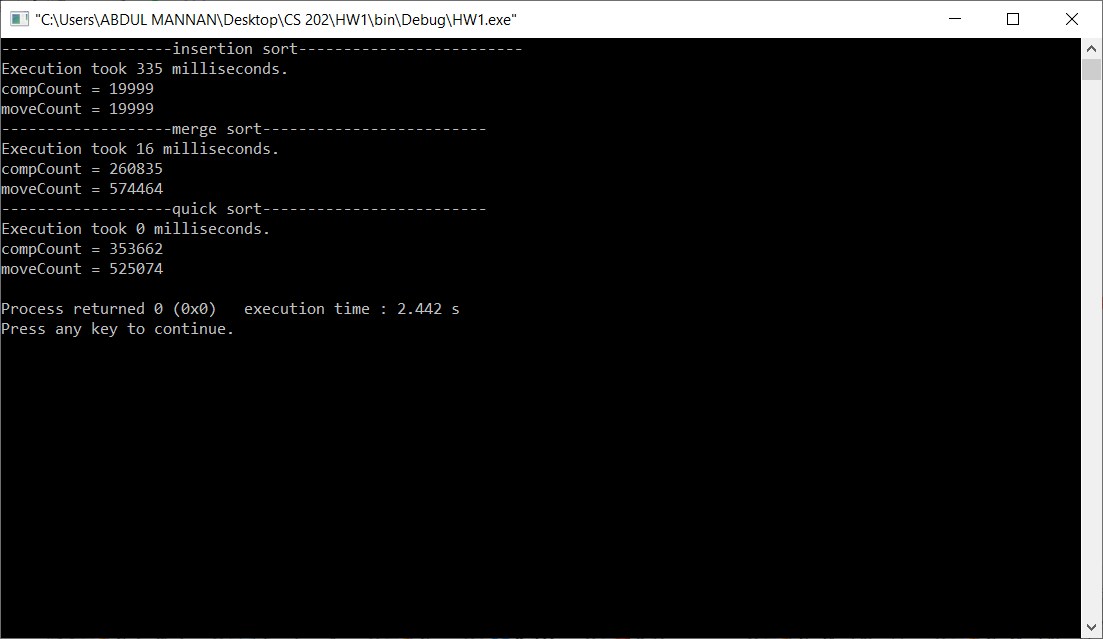
T(n) = 0 + n2 – [n - 1(n – 1 + 1)/2] Simplifying

T(n) = n2 – (n2 – n)/2 => T(n) = (n2 + n) / 2

This gives us **O(n2)** for the worst case using the recurrence relation.

Q3)

Sample Output:



|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Order | Size (N) | Insertion Sort | | | Merge Sort | | | Quick Sort | | |
| Time (s) | Comparisons | Moves | Time (s) | Comparisons | Moves | Time (s) | Comparisons | Moves |
| Random | 20000 | 0.369 | 99919076 | 99939068 | 0.011 | 260835 | 574464 | 0.004 | 353662 | 525074 |
| 22000 | 0.443 | 120819528 | 120841520 | 0.014 | 290045 | 638464 | 0.004 | 395300 | 591768 |
| 24000 | 0.528 | 144090640 | 144114632 | 0.015 | 319233 | 702464 | 0.003 | 427413 | 647435 |
| 26000 | 0.653 | 169302620 | 169328612 | 0.014 | 348960 | 766464 | 0.004 | 474481 | 735140 |
| 28000 | 0.733 | 196315760 | 196343752 | 0.015 | 378604 | 830464 | 0.003 | 544823 | 795201 |
| Descending | 20000 | 0.762 | 199990000 | 200009999 | 0.009 | 139216 | 574464 | 0.900 | 199990000 | 300079996 |
| 22000 | 0.948 | 241989000 | 242010999 | 0.011 | 154208 | 638464 | 1.097 | 241989000 | 363087996 |
| 24000 | 1.052 | 287988000 | 288011999 | 0.012 | 170624 | 702464 | 1.296 | 287988000 | 432095996 |
| 26000 | 1.278 | 337987000 | 338012999 | 0.012 | 186160 | 766464 | 1.518 | 337987000 | 507103996 |
| 28000 | 1.496 | 391986000 | 392013999 | 0.014 | 202512 | 830464 | 1.797 | 391986000 | 588111996 |
| Ascending | 20000 | 0.00007 | 19999 | 19999 | 0.010 | 148016 | 574464 | 0.449 | 199990000 | 79996 |
| 22000 | 0.00009 | 21999 | 21999 | 0.011 | 165024 | 638464 | 0.547 | 241989000 | 87996 |
| 24000 | 0.00009 | 23999 | 23999 | 0.012 | 180608 | 702464 | 0.656 | 287988000 | 95996 |
| 26000 | 0.00009 | 25999 | 25999 | 0.013 | 197072 | 766464 | 0.766 | 337987000 | 103996 |
| 28000 | 0.00012 | 27999 | 27999 | 0.015 | 212720 | 830464 | 0.938 | 391986000 | 111996 |

Graphs for Insertion Sort:

Figure 1: Insertion Sort 1

Figure 2: Insertion Sort 2

Figure 3: Insertion Sort 3

Graphs for MergeSort:

Figure 4: Merge Sort 1

Figure 5: Merge Sort 2

Figure 6: Merge Sort 3

Graphs for Quick Sort:

Figure 7: Quick Sort 1

Figure 8: Quick Sort 2

Interpretation:

First, we examine the case of the insertion sort algorithm. The insertion sort algorithm performed best when our array was in an ascending order, in this case, we do not have to execute the inner loop of the algorithm and therefore we are saved a lot of moves and comparisons which results in us saving time. The best case growth rate for insertion sort is **O(n)**. The algorithm performed even worse when the array was in a descending order, which is the worst case scenario for this algorithm as the inner loop executes the maximum number of times. The worst rate growth rate of the insertion sort algorithm is **O(n2)**. As we can see from the graph, when the array is in a random order, our insertion sort algorithm performs at somewhat of an average between the best and worst case. The average case for insertion sort is still **O(n2)**. Obviously as the size of the array grew, so did the time taken, comparison and moves made grow. The graph shows a trend similar to that of a parabola which shows an exponential growth and but fails to show the trend of a linear growth rate for the best case because it is so little compared to the other two trends and therefore looks more like a straight line.

Next, we examine the case of the merge sort algorithm. The merge sort algorithm performed extremely well with respect to time in all 3 cases, whether the array was random, in ascending order, or descending order. This is because its worst and average case are both **O(nlog2n)**. While the algorithm is slower than the insertion sort algorithm for the array in an ascending order. It is still better to use as it does better in cases that closely resemble real life, that is, we wouldn’t want to sort an already sorted array again. One thing of note is that the number of moves remained the same across all 3 arrays, this is because merge sort first breaks down the arrays completely and then starts building them back up, which means that the number of comparisons may differ but not the number of moves since data has to be transferred to a temp array first and then back to the original. The graphs corroborate our theoretical expectations, that is, the time for all 3 algorithm types is similar. The number of moves are the same and lastly, the number of comparisons are less for the arrays in ascending and descending orders because if we have 2 sub lists, we only need to compare the first element of one sub list to all the elements of the other because of the distribution, all others would either be smaller or larger. As such, we would quickly get to the end of one sub list quicker and can then just move the data in the other sub list to the temp array. If the distribution is random, the comparisons increase because all data from both sub lists needs to be compared to each other before we reach the end of either and proceed to move all the data to the temp array.

Lastly, we examine the quick sort algorithm. The quick sort algorithm performs best when the array is random. This fact is also proven by our experimental results. It can perform even better if we try to apply some sort of algorithm to choose our pivot better. Its best and average case growth rate is **O(nlog2n)**. The algorithm performs way worse in its worst case, when the array is already sorted and we choose the first element as our pivot. In this case its growth rate is **O(n2)**. This is again proven by out graphs. This is because the number of comparisons increase dramatically. In a random array, the array is divided into smaller arrays for sorting, these smaller arrays are a lot smaller than the parent array because one has elements bigger than the pivot and one has elements smaller than the pivot, as such since these arrays are separate they do not have to make comparisons with each other but only within themselves, this reduces the comparisons by a large margin. In sorted arrays that have the first element as a pivot, the smaller array that is made has only one element less that the parent array, which was the pivot. In this smaller array, all the data is going to be compared with each other which makes the number of comparisons in this equal to the number of elements in an array – 1 factorial. This is why both the ascending and descending array have the same number of comparisons. The ascending array still performs faster because it saves time on not having to move elements behind the pivot.

Overall, the quick sort and merge sort are both some of the best algorithms to use while the insertion sort does okay where the number of elements is small.