Question 1

a. $f(n) = 20n^4 + 20n^2 + 5$ is $O(n^5)$

$$0 \leq 20n^4 + 20n^2 + 5 \leq cn^5 \text{ for } n \geq n_0$$

$$0 \leq \frac{20}{n} + \frac{20}{n^3} + \frac{5}{n^5} \leq c$$

Choosing c = 50 and n_0 = 1

$$0 \leq \frac{20}{1} + \frac{20}{1^3} + \frac{5}{1^5} \leq 50$$

$$0 \leq 20 + 20 + 5 \leq 50$$

$$0 \leq 45 \leq 50$$

So, c = 50 and n_0 = 1 satisfy the condition, and $f(n) = 20n^4 + 20n^2 + 5$ is indeed $O(n^5)$.

b. Selection Sort

Gray elements are selected;
Black elements comprise of the sorted part of the array.

Initial Array:

18	4	47	24	15	24	17	11	31	23

After 1st swap:

18	4	23	24	15	24	17	11	31	47

After 2nd swap:

18	4	23	24	15	24	17	11	31	47

After 3rd swap:

18	4	23	11	15	24	17	24	31	47

After 4th swap:

18	4	23	11	15	17	24	24	31	47

After 5th swap:

18	4	17	11	15	23	24	24	31	47

After 6th swap:

15	4	17	11	18	23	24	24	31	47

After 7th swap:

15	4	11	17	18	23	24	24	31	47

After 8th swap:

11	4	15	17	18	23	24	24	31	47

After 9th swap:

4	11	15	17	18	23	24	24	31	47

Sorted Array:

4	11	15	17	18	23	24	24	31	47

Bubble Sort:

Gray elements are selected;
Black elements comprise of the sorted part of the array.

1st pass:

18	4	47	24	15	24	17	11	31	23

4	18	47	24	15	24	17	11	31	23

4	18	**47**	**24**	15	24	17	11	31	23

4	18	24	**47**	**15**	24	17	11	31	23

4	18	24	15	**47**	**24**	17	11	31	23

4	18	24	15	24	**47**	**17**	11	31	23

4	18	24	15	24	17	**47**	**11**	31	23

4	18	24	15	24	17	11	**47**	**31**	23

4	18	24	15	24	17	11	31	**47**	**23**

4	18	24	15	24	17	11	31	23	**47**

2nd Pass:

4	**18**	24	15	24	17	11	31	23	**47**

4	**18**	**24**	15	24	17	11	31	23	**47**

4	18	**24**	**15**	24	17	11	31	23	**47**

4	18	15	**24**	**24**	17	11	31	23	**47**

4	18	15	24	**24**	**17**	11	31	23	**47**

4	18	15	24	17	**24**	**11**	31	23	**47**

4	18	15	24	17	11	**24**	**31**	23	**47**

4	18	15	24	17	11	24	31	23	47

4	18	15	24	17	11	24	23	31	47

3rd Pass:

4	18	15	24	17	11	24	23	31	47

4	18	15	24	17	11	24	23	31	47

4	15	18	24	17	11	24	23	31	47

4	15	18	24	17	11	24	23	31	47

4	15	18	17	24	11	24	23	31	47

4	15	18	17	11	24	24	23	31	47

4	15	18	17	11	24	24	23	31	47

4	15	18	17	11	24	23	24	31	47

4th Pass:

4	15	18	17	11	24	23	24	31	47

4	15	18	17	11	24	23	24	31	47

4	15	18	17	11	24	23	24	31	47

4	15	17	18	11	24	23	24	31	47

4	15	17	11	18	24	23	24	31	47

4	15	17	11	18	24	23	24	31	47

4	15	17	11	18	23	24	24	31	47

5th Pass:

4	15	17	11	18	23	24	24	31	47

4	15	17	11	18	23	24	24	31	47

4	15	17	11	18	23	24	24	31	47

4	15	11	17	18	23	24	24	31	47

4	15	17	11	18	23	24	24	31	47

4	15	17	11	18	23	24	24	31	47

6th Pass:

4	15	17	11	18	23	24	24	31	47

4	15	17	11	18	23	24	24	31	47

4	15	17	11	18	23	24	24	31	47

4	15	11	17	18	23	24	24	31	47

4	15	11	17	18	23	24	24	31	47

7th Pass:

4	15	11	17	18	23	24	24	31	47

4	15	11	17	18	23	24	24	31	47

4	11	15	17	18	23	24	24	31	47

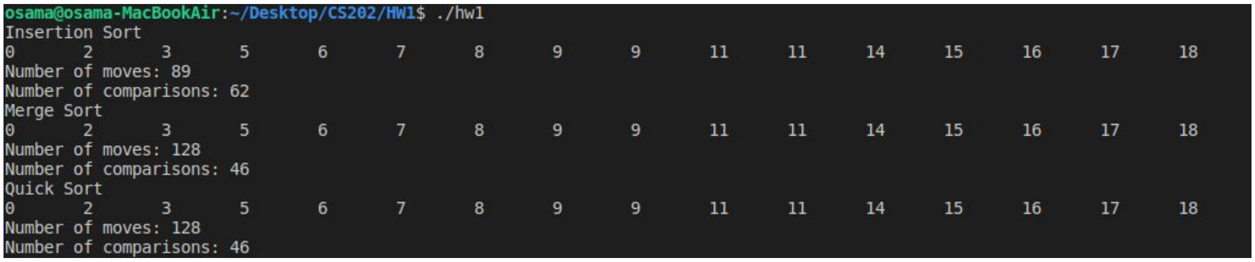
4	11	15	17	18	23	24	24	31	47

Sorted Array:

4	11	15	17	18	23	24	24	31	47

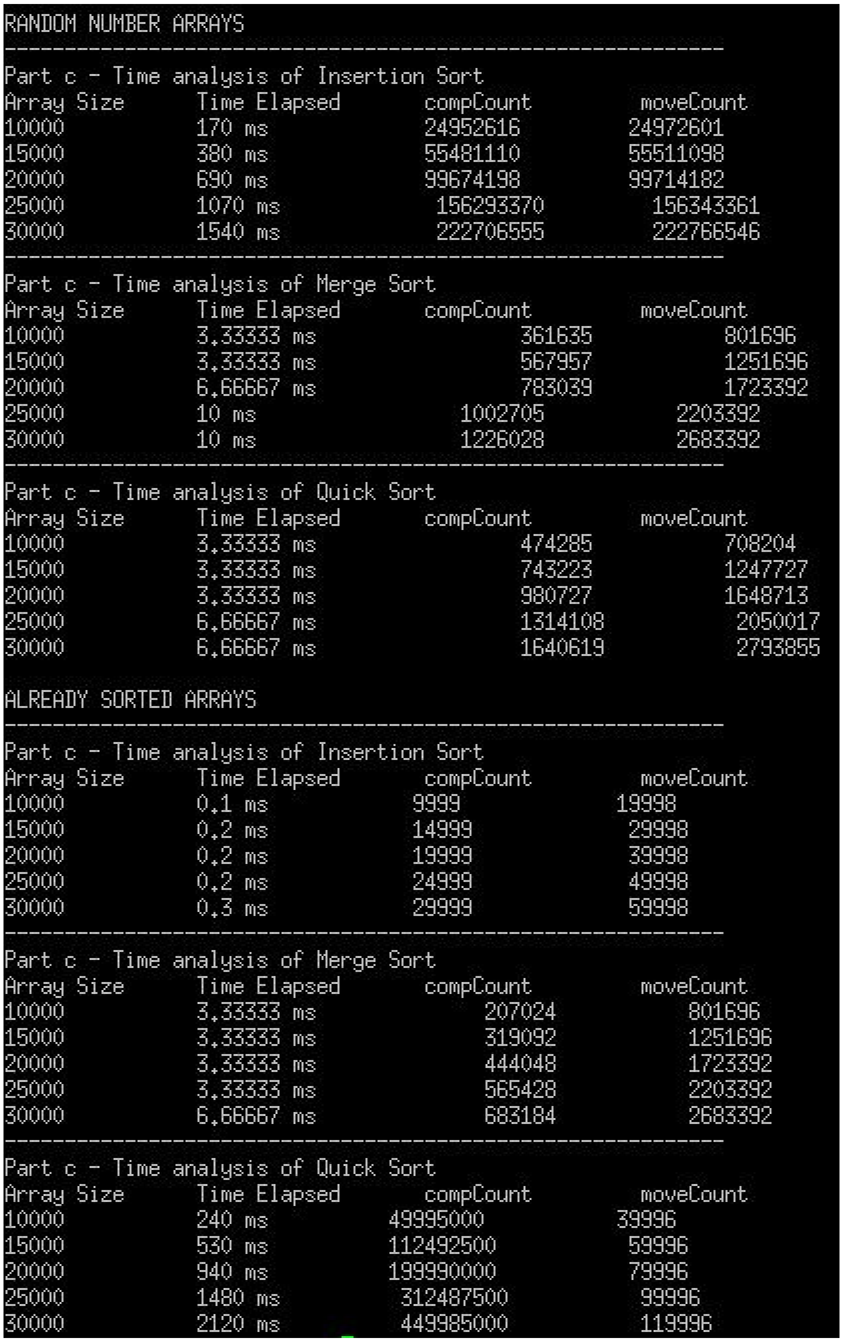
Question 2

b.



```
osama@osama-MacBookAir:~/Desktop/CS202/HW1$ ./hw1
Insertion Sort
0       2       3       5       6       7       8       9       9       11      11      14      15      16      17      18
Number of moves: 89
Number of comparisons: 62
Merge Sort
0       2       3       5       6       7       8       9       9       11      11      14      15      16      17      18
Number of moves: 128
Number of comparisons: 46
Quick Sort
0       2       3       5       6       7       8       9       9       11      11      14      15      16      17      18
Number of moves: 128
Number of comparisons: 46
```

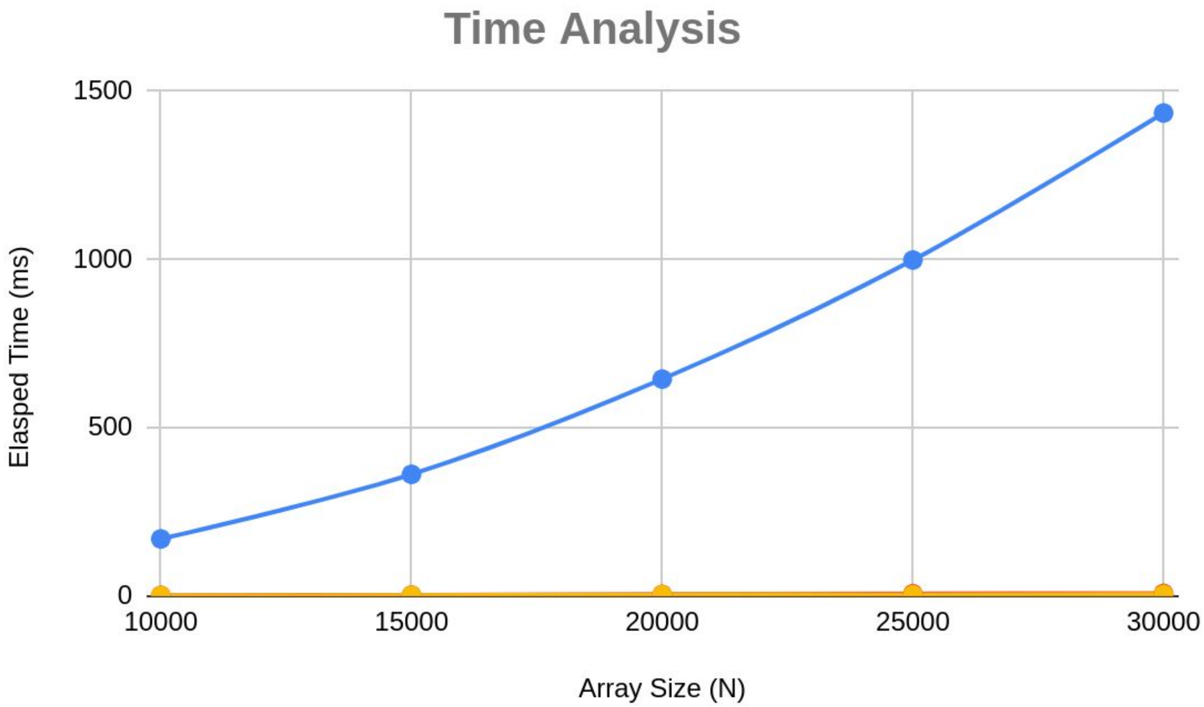
c.



```
RANDOM NUMBER ARRAYS
------------------------------------------------------------
Part c - Time analysis of Insertion Sort
Array Size      Time Elapsed       compCount         moveCount
10000           170 ms             24952616         24972601
15000           380 ms             55481110         55511098
20000           690 ms             99674198         99714182
25000           1070 ms             156293370         156343361
30000           1540 ms             222706555         222766546
------------------------------------------------------------
Part c - Time analysis of Merge Sort
Array Size      Time Elapsed       compCount         moveCount
10000           3.33333 ms                 361635           801696
15000           3.33333 ms                 567957           1251696
20000           6.66667 ms                 783039           1723392
25000           10 ms                 1002705           2203392
30000           10 ms                 1226028           2683392
------------------------------------------------------------
Part c - Time analysis of Quick Sort
Array Size      Time Elapsed       compCount         moveCount
10000           3.33333 ms                 474285           708204
15000           3.33333 ms                 743223           1247727
20000           3.33333 ms                 980727           1648713
25000           6.66667 ms                 1314108           2050017
30000           6.66667 ms                 1640619           2793855

ALREADY SORTED ARRAYS
------------------------------------------------------------
Part c - Time analysis of Insertion Sort
Array Size      Time Elapsed       compCount         moveCount
10000           0.1 ms            9999             19998
15000           0.2 ms            14999             29998
20000           0.2 ms            19999             39998
25000           0.2 ms            24999             49998
30000           0.3 ms            29999             59998
------------------------------------------------------------
Part c - Time analysis of Merge Sort
Array Size      Time Elapsed       compCount         moveCount
10000           3.33333 ms              207024           801696
15000           3.33333 ms              319092           1251696
20000           3.33333 ms              444048           1723392
25000           3.33333 ms              565428           2203392
30000           6.66667 ms              683184           2683392
------------------------------------------------------------
Part c - Time analysis of Quick Sort
Array Size      Time Elapsed       compCount         moveCount
10000           240 ms          49995000           39996
15000           530 ms          112492500           59996
20000           940 ms          199990000           79996
25000           1480 ms          312487500           99996
30000           2120 ms          449985000           119996
```

d.



Blue Line - Insertion Sort
Yellow Line - Merge Sort
Red Line - Quick Sort

According to the graph above, insertion sort shows a time complexity of O(N^2). According to theoretical results, insertion sort has a time complexity of O(N^2) as well. The graph of insertion sort has a parabolic trend indicating this relationship. However, only 5 different values for N were compared thus the entire trend could not be visualized in the graph. But the bending of the graph indicates that insertion sort increases by N^2. Analyzing the graphs for merge sort and quicksort, we can see that the graphs hardly even rise for values of N this low. This indicates a logarithmic trend. However, the average time complexity for merge sort and quicksort is O(NlogN). We can see the graphs not increasing very sharply compared to N^2 but due to the usage of only 5 values of N, it is viable to claim that the graph should not increase very much. If more values of N were used the graph would have followed the NlogN trend, which we can not be sure of according to the graph above. Hence, we can conclude that to capture an accurate estimate of the time complexity, we need to use more values of N.

When already sorted arrays are used, insertion sort turns out being the best algorithm to use according to its time complexity. This change in time complexity happens because the inner loop is never executed. Therefore, the time complexity tends to be O(N) for insertion sort. However, quicksort tends to do a lot worse. Quicksort tends to have a time complexity of O(N^2). This is because this is the worst case for quicksort. The pivot picked is the first element, which is the smallest element. Hence, comparisons are made based on this pivot. Merge sort exhibits the same complexity as before of O(NlogN). In conclusion, insertion sort is the best algorithm with the lowest time complexity, followed by merge sort, and then quicksort.

Question 3

k	**Time (ms)**		
	Insertion Sort	**Merge Sort**	**Selection Sort**
0	0	0	2
1000	1	0	5
2000	5	0	11
3000	13	0	21
4000	23	0	44
5000	34	1	65
6000	52	1	96
7000	71	1	112
8000	98	2	121
9000	126	1	137
10000	157	2	145

The table represents the time taken by each algorithm for different values of k, given a value of N (i.e. 10,000 in this case). In case of insertion sort, we can see that it takes a very small amount of time for it to completely sort an array. The inner loop is not executed because most of the array is already sorted. However, as we increase the value of k, we can see that the time for insertion sort increases. This is because the inner loop now sorts the array, for now it does not have a sorted array to begin with as before. Therefore, the time for insertion sort increases. As we increase the value of N, we note that as the value of k increases the time taken for insertion sort to completely sort an array also increases.

In case of merge sort, the algorithm is divide and conquer. Merge sort works on a time complexity of O(NlogN) in the best, average, and worst cases. It can be seen by the time taken for merge sort to sort the array that the time required for merge sort increases very slowly, as the value of k increases. Therefore, we can say that no matter how far a number is from the target address, merge sort has the same average time complexity for these different positions of numbers.

In case of quicksort, the algorithm works by taking the first element as its pivot. As we iterate through the sorted array, as in case when k is 0, the worst case of quicksort occurs. This is because the pivot selected is the smallest value, in case of k = 0. But as we increase the value of k, there are more chances that the pivot selected is not the smallest and is probably random. Therefore, the time required for quicksort decreases as we increase the value of k.

The most efficient solution to this problem would be merge sort. This is because merge sort exhibits a time complexity of O(nlogn) in the best, average, and worst cases. Therefore, no matter what the value of k is, in accordance with the value of n, merge sort tends to work best. This is however not true, in case of insertion sort. When the value of k increases to values close to the value of N, the time required increases significantly. Therefore, insertion sort is not a suitable option. Compared to quicksort, merge sort does use a considerable amount of space, but in the worst case of quicksort, quicksort displays a behavior of O(N^2). Merge sort, therefore, is the best efficient solution to this problem.