Name: Murat Sevinç

ID: 21702603

HW_1

Question 1: a) $T(n) = 5T(n/3) + n.\log n$, where $T(1) = 1$

$= 5[\ 5(T/9) + (n/3).\ \log(n/3)] + n.\log n$

$= \ 5[5[(T/27) + (n/9).\ \log(n/9)] + (n/3).\log(n/3)] + n.\log n$

… (k steps after) … (k steps after)

$5^k\ .\ T(n/3^k) \quad + \quad \log_3 n\ .\ n.\log n$

T(1) is O(1), so, $\mathbf{n/3^k = 1}$ => $\mathbf{k = \log_3 n}$

$O(\ N + (\log n)^2\ .\ n) = \mathbf{O(n\ (\log n)^2)}$

b) $T(n) = T(n - 1) + (n^2)$ where $T(1) = 1$.

$= T(n\text{-}2) + (n\text{-}1)^2 + n^2$

…

(k steps after)

$= T(n\text{-}k) + (n\text{-}k\text{+}1)^2 + (n\text{-}k\text{+}2)^2 \ldots + (n\text{-}1)^2 + n^2$ } there are k terms each $O(n^2)$

Where $n - k = 1 \Rightarrow k = n\text{-}1$

Result: $O(\ (n\text{-}1)\ .\ (n^2) \Rightarrow \mathbf{O(n^3)}$

Question 1 b):

Merge Sort : [44, 937, 13, 69, 37, 80, 472, 49, 300, 183]

Note: * indicates that split in that stage

Step 1 (divide)

44	937	13	69	37	*	80	472	49	300	183

Step 2 (divide)

44	937	13	*	69	37		80	472	49	*	300	183

Step 3 (divide)

44	937	*	13		69	*	37		80	472	*	49		300	*	183

Step 4 (last divide)

44	*	937		13		69		37		80	*	472		49		300		183

Step 5 (merge) (Bold values are merged ones)

44	**937**		13		**37**	**69**		**80**	**472**		49		**183**	**300**

Step 6 (merge)

13	**44**	**937**		**37**	**69**		**49**	**80**	**472**		**183**	**300**

Step 7 (merge)

13	**37**	**44**	**69**	**937**		**49**	**80**	**183**	**300**	**472**

Step 8 (last merge)

13	**37**	**44**	**49**	**69**	**80**	**183**	**300**	**472**	**937**

Insertion Sort : [44, 937, 13, 69, 37, 80, 472, 49, 300, 183]

44	937	13	69	37	80	472	49	300	183
44	**937**	13	69	37	80	472	49	300	183
13	**44**	**937**	69	37	80	472	49	300	183
13	**44**	**69**	**937**	37	80	472	49	300	183
13	**37**	**44**	**69**	**937**	80	472	49	300	183
13	**37**	**44**	**69**	**80**	**937**	472	49	300	183
13	**37**	**44**	**69**	**80**	**472**	**937**	49	300	183
13	**37**	**44**	**49**	**69**	**80**	**472**	**937**	300	183
13	**37**	**44**	**49**	**69**	**80**	**300**	**472**	**937**	183
13	**37**	**44**	**49**	**69**	**80**	**183**	**300**	**472**	**937**

Question 1: c)

Worst case of quick sort is when the array is **already sorted.**

Quick Sort when worst case:

```
{
        partition(theArray, first, last, pivotIndex);          O(n)

        quicksort(theArray, first, pivotIndex-1);              T(0)

        quicksort(theArray, pivotIndex+1, last);               T(n-1)

}
```

Recurrence Equation: $T(n) = T(n-1) + O(n)$ when $T(0) = O(1)$

$= T(n-2) + (n-1) + n$

$= T(n-3) + (n-2) + (n-1) + n$

…

…

$= T(n-k) + (n-k+1) + (n-k+2) + \ldots(n-2) + (n-1) + n$

When $k = n$

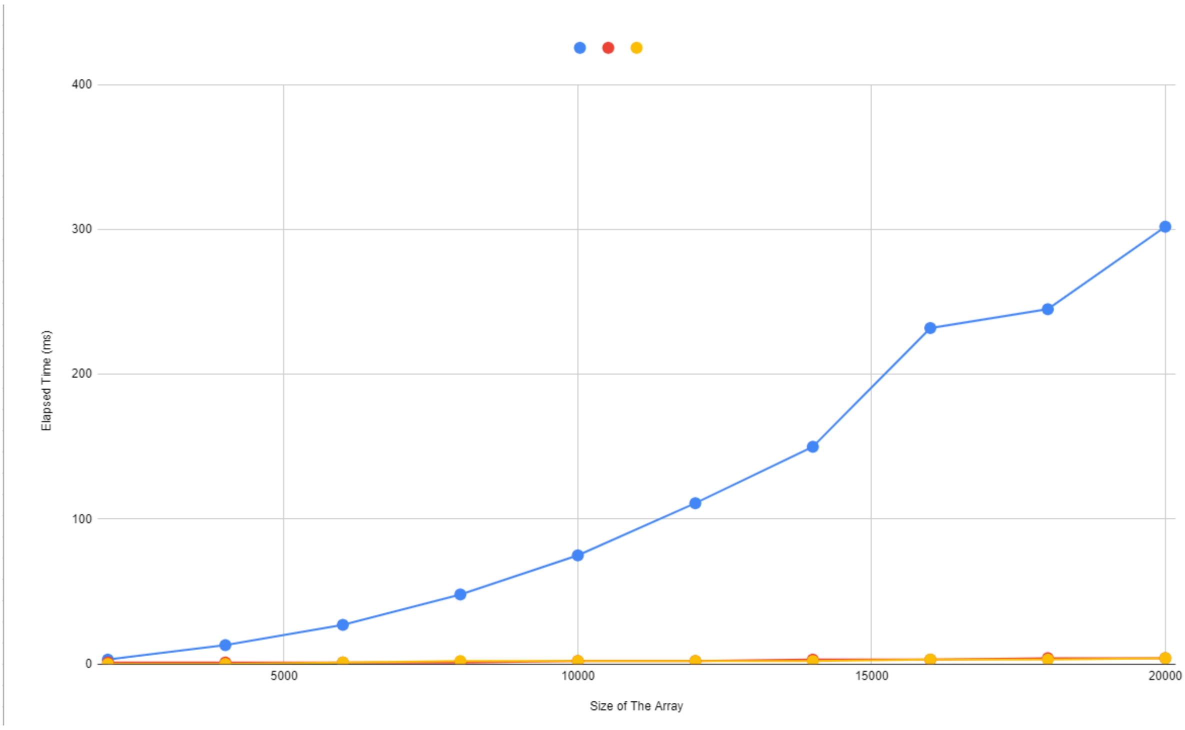
Result: $n(n-1) / 2 \Rightarrow \mathbf{O(n^2)}$

Question 2:

"C:\Users\Murat\OneDrive\Masa³st³\CS 202\Homeworks\hw1\21702603_hw1\hw1\bin\Debug\hw1.exe"

```
1       17      20      43      57      58      92      93      99      100
1       17      20      43      57      58      92      93      99      100
1       17      20      43      57      58      92      93      99      100

------------------------------------------------------------

Part a - Time analysis of Quick Sort
Array size      Time Elapsed    compCount       moveCount

2000            1               25588           40354
4000            1               54057           94551
6000            1               86439           150043
8000            1               121603          186962
10000           2               163407          270632
12000           2               192774          348130
14000           3               229313          400428
16000           3               265075          449238
18000           4               310751          503167
20000           4               330900          564323
------------------------------------------------------------

Part b - Time analysis of Insertion Sort
Array size      Time Elapsed    compCount       moveCount

2000            3               991307          995295
4000            13              4037023         4045014
6000            27              9004911         9016902
8000            48              15970752        15986745
10000           75              24685954        24705944
12000           111             35832476        35856465
14000           150             49125934        49153924
16000           232             63848548        63880541
18000           245             80920603        80956590
20000           302             99892569        99932556
------------------------------------------------------------

Part c - Time analysis of Hybrid Sort
Array size      Time Elapsed    compCount       moveCount

2000            0               25226           37314
4000            0               53383           88131
6000            1               85733           140876
8000            2               120453          174372
10000           2               161911          255206
12000           2               190614          329519
14000           2               227305          378184
16000           3               262673          424542
18000           3               308064          474809
20000           4               327650          533640
Process returned 0 (0x0)   execution time : 1.951 s
Press any key to continue.
```

Question 3:



Blue: insertion sort Red: Quick Sort Yellow: Hybrid Sort

As the table demonstrates, insertion sort is growing quadratic, there is an awkward situation for input size: 16000, yet this may be caused by our random array. It generally looks like n^2 which is theoretically correct.

Quick sort and hybrid sort are also as expected. They don't really change when input increases. This is because of their time complexities which are log (n).

Hybrid sort and quick sort were really fast. Theoretically, quick sort should be quicker than the hybrid sort. Yet, in my result there was a little difference. Hybrid sort was faster like 1 ms in some cases. This may be caused by the fact that hybrid sort needs less data moves and comparisons when compared to the quick sort. Therefore, one advantage of hybrid sort is it needs less data moves and key comparisons. But theoretically, quick sort is faster.