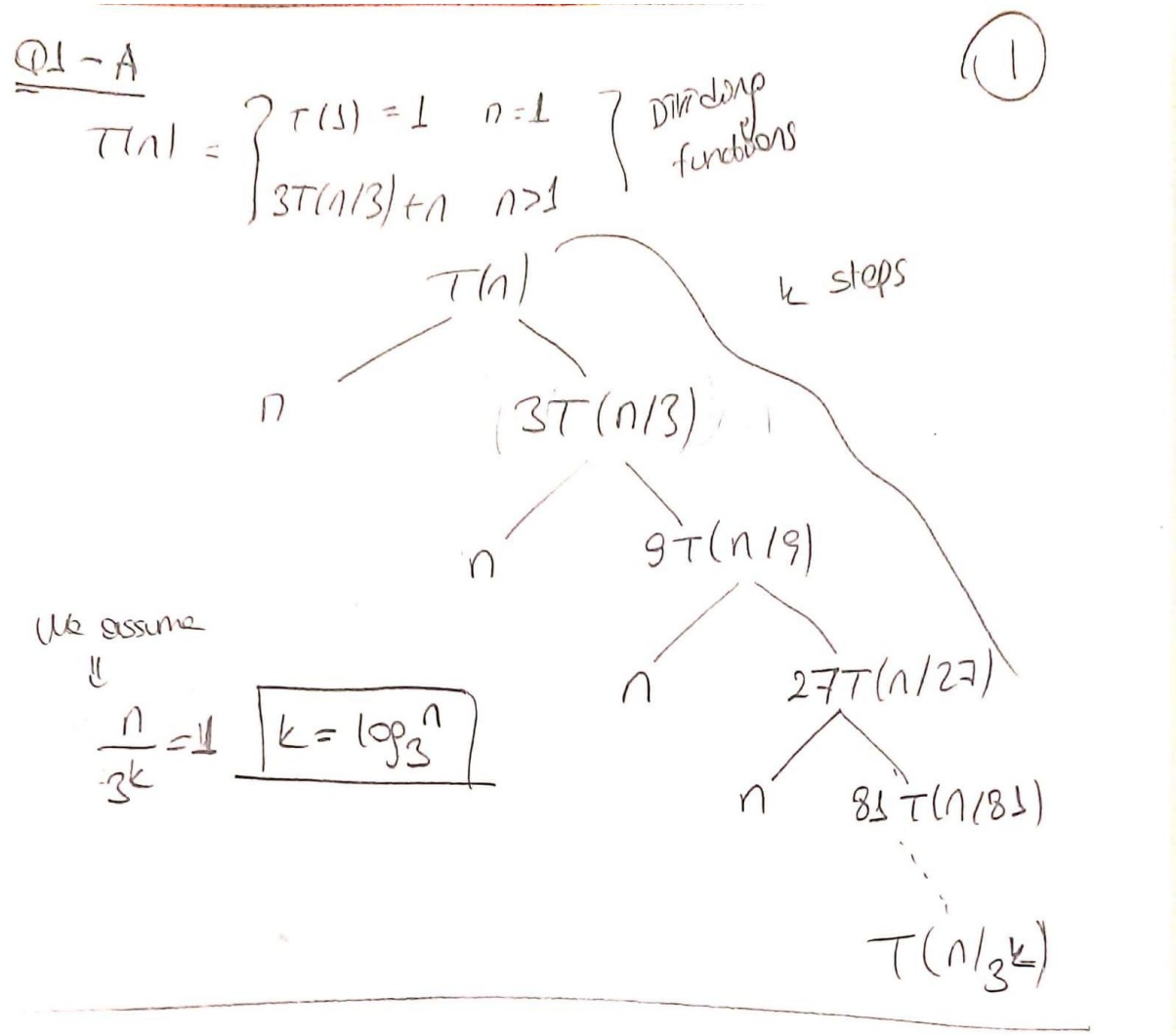
CS202 Homework 01 – Ali Eren Günaltılı 21801897

1.A-Find the asymptotic running time for.

A-)T(n) = 3T(n/3)+n, where T(1) = 1 and n is an exact power of 3



A-1

$T(n) = 3T(n/3) + n$

$T(1) = 1$

$T(n) = 3T(n/3) + n$

$T(n/3) = 3T(n/9) + n/3$

$T(n/9) = 3T(n/27) + n/9$

$T(n) = 3[3T(n/9) + n/3] + n$

$T(n) = 9T(n/9) + 2n$

$T(n) = 9[3T(n/27) + n/9] + 2n$

$T(n) = 27T(n/27) + 3n$

$T(n) = 9T(n/9) + 2n$

$T(n) = 3T(n/3) + n$

$\Rightarrow 3^k T(n/3^k) + k \cdot n$

$k = \log_3 n$

$T(n) = 3^{\log_3 n} T(n/3^{\log_3 n}) + \log_3 n \cdot n$

$T(n) = n \cdot T(1) + \log n \cdot n$

$O(\log n \cdot n)$

B-) T(N) = 3T(N/2) + 1

B-)

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

$T(1) = 1$

$$T(n) = \begin{cases} 1 & n = 1 \\ 3T(n/2)+1 & n > 1 \end{cases}$$

$T(n) = 3T(n/2) + 1$ $\quad T(n) = 3[3T(n/4)+1]+1$

$T(n/2) = 3T(n/4) + 1$ $\quad T(n) = 9T(n/4)+4$

$T(n/4) = 3T(n/8) + 1$ $\quad T(n) = 9[3T(n/8)+1]+4$

$T(n) = 27T(n/8) + 13$

$\downarrow$

$3^k T(n/2^k) + (3.2^{k-1} + 1)$

To make $T(n/2^k) = T(1)$

$2^k = n$

$\log_2^n = k$

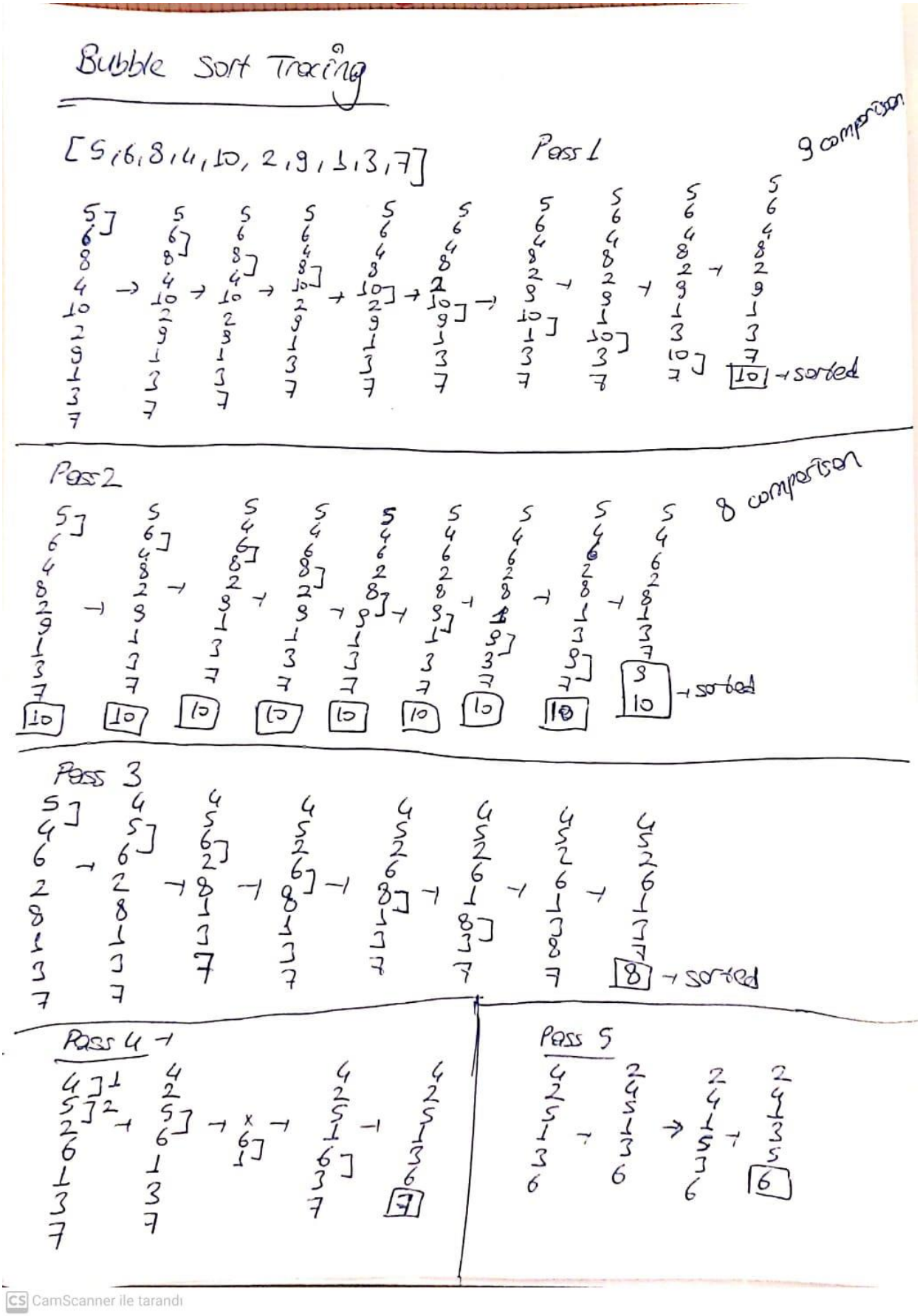
$T(n) = 3^{\log_2^n} T(n/2^k) + (3.2^{k-1} + 1)$

$= n^{\log_2^3} T(n/2^{\log_2^n}) + \left(\frac{3}{2} \cdot 2^{\log_2^n} + 1\right)$

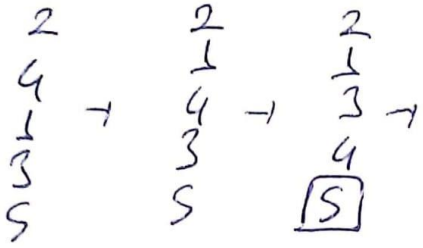
$= n^{\log_2^3} T(n/n) + \frac{3}{2}n + 1$

$\log_2^3 > 1 \rightarrow n^{1.234} \rightarrow \boxed{O(n^2)}$ //

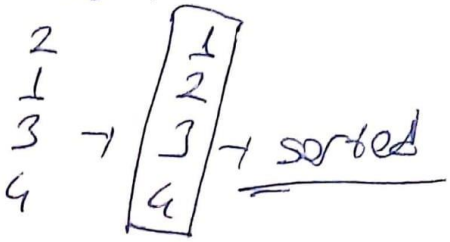
1.B-Tracing with Bubble and Selection Sort



Pass 6



Pass 7



Selection Sort Tracing

unsorted | sorted

5 6 8 4 (10) 2 9 1 3 7 |

5 6 8 4 7 2 (9) 1 3 | 10

5 6 (8) 4 7 2 3 1 | 9 10

5 6 1 4 (7) 2 3 | 8 9 10

5 (6) 1 4 3 2 | 7 8 9 10

(5) 2 1 4 3 | 6 7 8 9 10

3 2 1 (4) | 5 6 7 8 9 10

(3) 2 1 | 4 5 6 7 8 9 10

1 2 3 4 5 6 7 8 9 10 // → sorted

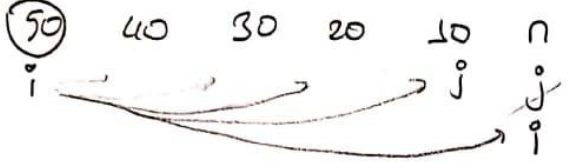
1.C-Recurrence Relation of Quick Sort at its worst case

Question 1-C

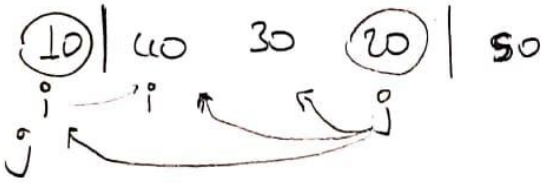
If elements are ascending or descending order, quick sort will be $O(n^2)$ which is the worst case.

50 40 30 20 10
10 20 30 40 50
→ Both of them will end with $O(n^2)$. Since partioning happens at the edge of array.

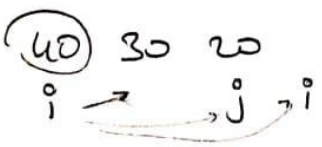
pass1 pivot



pass2

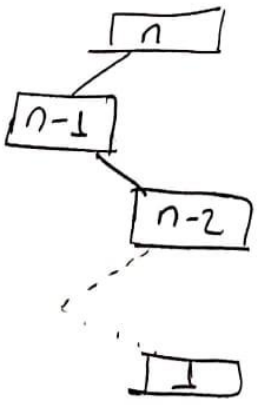


pass3



→ swap (pivot, j) = 20 30 40

No of comparison to partition quick sort



$\frac{n(n-1)}{2}$ comparison $= O(n^2)$.

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

Question 1-c

$$T(n) = \begin{cases} 1 & n=0 \\ T(n-1)+O(n) & n>0 \end{cases}$$

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

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

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

$T(n) = T(n-2)+2 \cdot O(n)$

$T(n) = T(n-3)+3 \cdot O(n)$

$\vdots$ k times

$T(n) = T(n-k)+k \cdot O(n)$

$n-k=0$

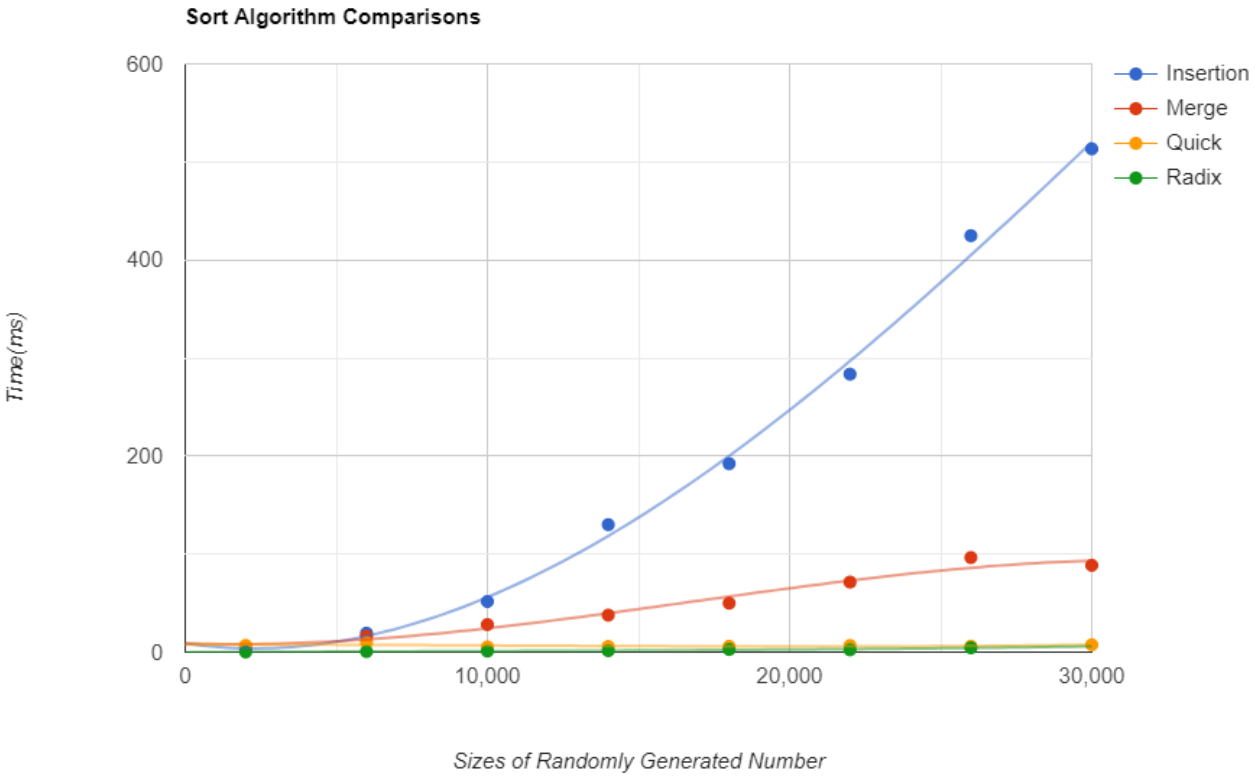
$k=n$

$T(n-n)+n \cdot O(n) =$

$T(0)+n^2 = n^2+1 = \boxed{O(n^2)}$

Performance Analysis Comparisan Between Four Sorts

Sizes	2000	6000	10000	14000	18000	22000	26000	30000
Insertion	2.41430ms	19.3366ms	51.6722 ms	130.031 ms	192.18 ms	283.524 ms	424.76 ms	513.306 ms
Merge	5.6893ms	16.188 ms	28.1506 ms	37.7412 ms	49.9368 ms	71.4521 ms	96.4917 ms	98.5735 ms
Quick	6.868ms	9.8599 ms	5.3258 ms	5.7591 ms	5.9589 ms	6.7796 ms	6.0239 ms	7.5471 ms
Radix	0.2036 ms	0.6156 ms	1.1023 ms	1.5501 ms	2.7517 ms	2.4031 ms	4.4148 ms	4.8569 ms



Insertion – Avarage = $O(n^2)$, Worst = $O(n^2)$

Merge – Avarage = O(nlog(n)), O(nlog(n))

Quick – Average = O(nlog(n)), worst = $O(n^2)$

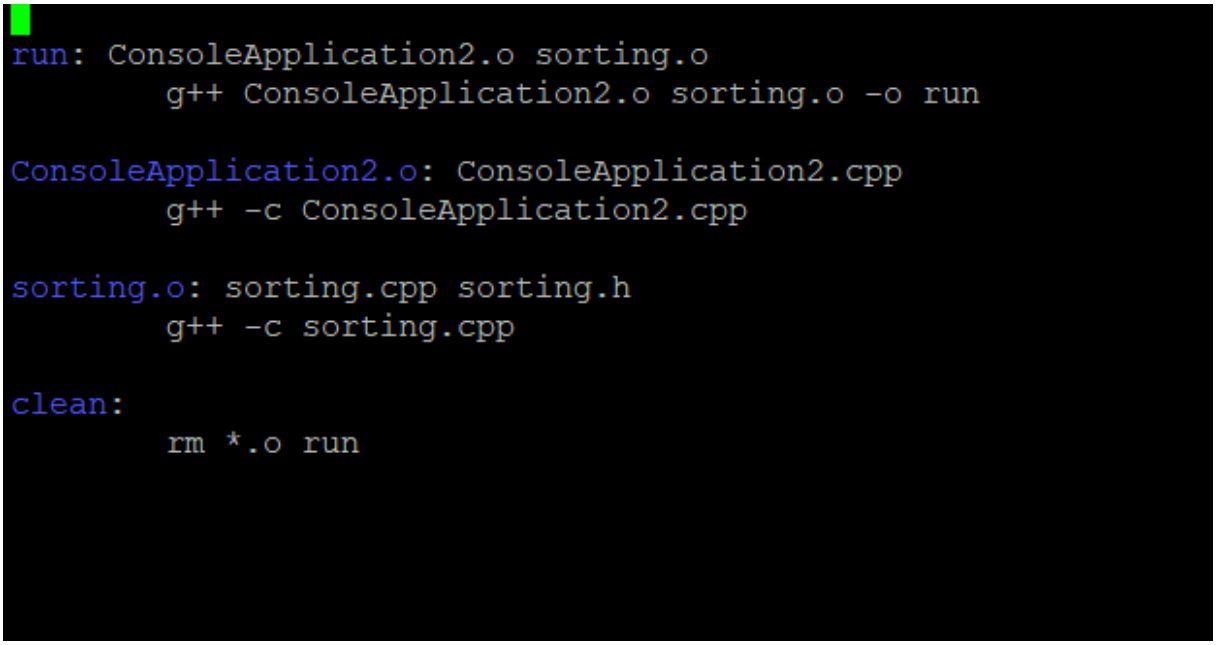
Radix – Avarage = O(d*(n+b))) where b is base, d is number of digits.

The worst sorting algorithm was insertion sort just like expected. Therefore, for this condition, empirical and theoretical values suit. The best algorithm was radix sort as it works with O(n) complexity. All sort algorithms take more time with respect to array size. However, for a few cases, there is an exception. Sometimes even though the size of the array gets bigger, the runtime gets smaller. But that is also possible since we create randomly generated arrays. Some random series of numbers may create easier sorting for quick as it is number sensitive in a way. In quick sort order of numbers and their places directly affect the runtime. However, a sort algorithm like merge is not affected by the order of numbers. Therefore, there is no fluctuation in its runtime results. Merge algorithm divides the array till it creates sub-arrays with size 1. After it divides, it starts to merge them by comparing. Thus, it is not affected by the position of values in a given random array.

Another critical outcome is the difference between the runtime increasing with respect to the array size. As all algorithms depend on the size parameter, they are increasing but the difference is getting bigger because they have different growth rates. By considering their average cases, the difference between insertion and other algorithms is understandable. Insertion grows with n2. The interesting point is why the quick sort is so fast although it has also the worst case with n2 too? Because worst case of the quick occurs when the array is sorted or sorted in descending order. As we are randomly generated our arrays, we don't see quick's worst-case scenario. Therefore, it is normal to have such a difference between quick and merge with an array size of 30000. For small sizes arrays, their differences are not that huge. The best algorithm was radix sort by considering experimental results. Radix sort controls digit by digit to sort the array. In case of the given array is sorted, merge sort won't affect it. But insertion sort is affected positively since the sorted array is its best case. After it divides the array into two as; unsorted and sorted, it doesn't have a value to put sorted side as the other unsorted side is also sorted array.

Quicksort is affected in a bad way since the sorted array is its worst case. As we choose our pivot value from the beginning, we traverse the whole array to understand our pivot is in the right position.Radix sort is also not affected since it is not value-position sensitive.

Notes: I used auto start1 = chrono::high_resolution_clock::now(), when I collecting data and filling the table given above. However, when I try to run my program on dijkstra it gives error. Therefore, I measure the runtime with clock_t.start() method. But, in newer version merge sort is always slower than insertion unlike the expectataion. I have also taken the permission of Can Hoca and TA to put my answers in handwritten for Question 1.



```
run: ConsoleApplication2.o sorting.o
	g++ ConsoleApplication2.o sorting.o -o run

ConsoleApplication2.o: ConsoleApplication2.cpp
	g++ -c ConsoleApplication2.cpp

sorting.o: sorting.cpp sorting.h
	g++ -c sorting.cpp

clean:
	rm *.o run
```

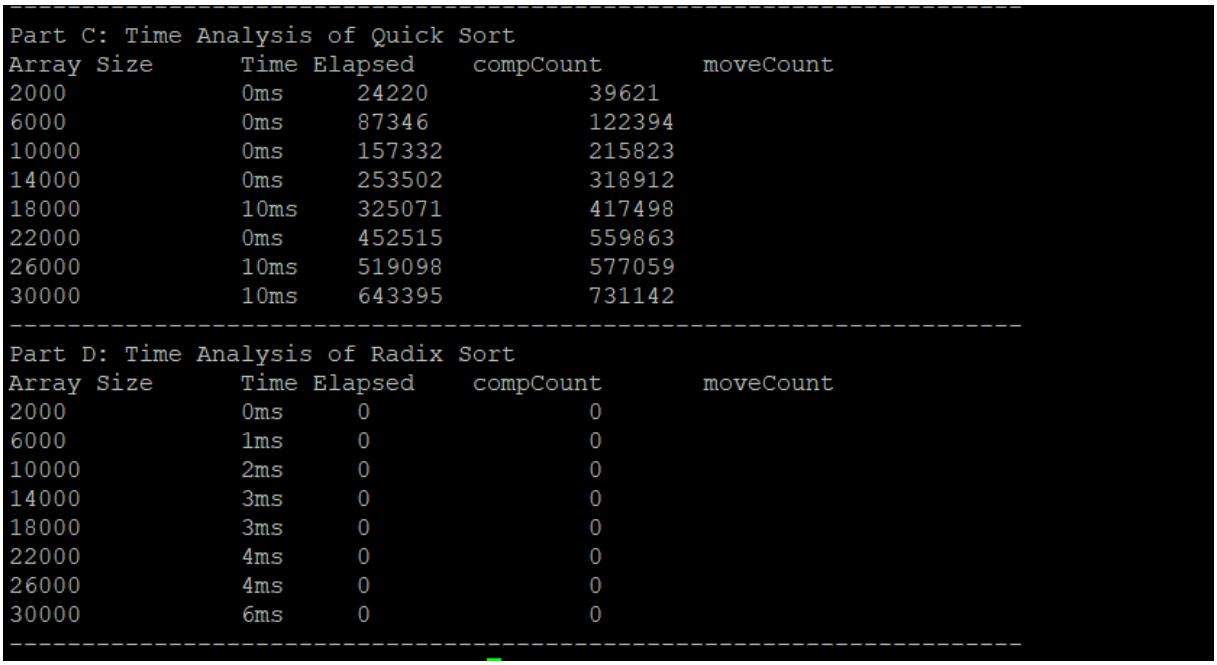
1-Makefile

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
-----------------------------------------------
Part A: Time Analysis of Insertion Sort
Array Size      Time Elapsed    compCount       moveCount
2000            0.3ms   1999000                 1064338
6000            1.9ms   17997000                9115449
10000           5.6ms   49995000                25061625
14000           11.1ms  97993000                49687562
18000           17.8ms  161991000               81023982
22000           26ms    241989000               121835995
26000           39ms    337987000               169592679
30000           47ms    449985000               224426774
-------------------------------------------------------------------
Part B: Time Analysis of Merge Sort
Array Size      Time Elapsed    compCount       moveCount
2000            5ms     115100          439040
6000            14.7ms  408670          1516160
10000           24.2ms  726520          2672320
14000           33.9ms  1057730         3872320
18000           42.6ms  1403420         5104640
22000           52.3ms  1734380         6384640
26000           61.1ms  2099940         7664640
30000           70.7ms  2462880         8944640
-------------------------------------------------------------------
Part C: Time Analysis of Quick Sort
Array Size      Time Elapsed    compCount       moveCount
2000            1ms     27841           40821
6000            2ms     88494           131652
10000           3ms     166514          220776
14000           4ms     239904          320001
18000           4ms     327125          450306
22000           6ms     413113          501600
26000           7ms     541657          671949
30000           8ms     657952          763521
-------------------------------------------------------------------
Part D: Time Analysis of Radix Sort
Array Size      Time Elapsed    compCount       moveCount
2000            0.2ms   0               0
6000            0.6ms   0               0
10000           1ms     0               0
14000           1.3ms   0               0
18000           1.7ms   0               0
22000           2.1ms   0               0
26000           2.5ms   0               0
30000           3ms     0               0
-------------------------------------------------------------------
```

2-Visual Studio Screenshot

```
[eren.gunaltili@dijkstra CS202]$ emacs Makefile
[eren.gunaltili@dijkstra CS202]$ make
g++ -c ConsoleApplication2.cpp
g++ -c sorting.cpp
g++ ConsoleApplication2.o sorting.o -o run
[eren.gunaltili@dijkstra CS202]$ ./run
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
-----------------------------------------------
Part A: Time Analysis of Insertion Sort
Array Size      Time Elapsed    compCount       moveCount
2000            1ms     1999000                 1019807
6000            5ms     17997000                9132828
10000           13ms    49995000                24988035
14000           26ms    97993000                49024247
18000           44ms    161991000               81726530
22000           65ms    241989000               121665227
26000           90ms    337987000               169924018
30000           120ms   449985000               226948089
-----------------------------------------------------------------------
Part B: Time Analysis of Merge Sort
Array Size      Time Elapsed    compCount       moveCount
2000            1ms     115150          439040
6000            1ms     407420          1516160
10000           2ms     726030          2672320
14000           2ms     1058490         3872320
18000           4ms     1398860         5104640
22000           4ms     1744760         6384640
26000           5ms     2099940         7664640
30000           6ms     2468610         8944640
-----------------------------------------------------------------------
Part C: Time Analysis of Quick Sort
Array Size      Time Elapsed    compCount       moveCount
2000            0ms     24220           39621
6000            0ms     87346           122394
10000           0ms     157332          215823
14000           0ms     253502          318912
18000           10ms    325071          417498
22000           0ms     452515          559863
26000           10ms    519098          577059
30000           10ms    643395          731142
-----------------------------------------------------------------------
```

3-Dijkstra Machine Screenshot



4-Remaining screenshot