CS202
Tuğrul DAYAR
Section 3

Homework
-1-

Burak Erdem VAROL
CS / 21202534

Question 1a)

$f_4(n) < f_9(n) < f_8(n) < f_2(n) < f_5(n) < f_3(n) < f_1(n) = f_6(n) < f_{10}(n) < f_7(n) < f_{11}(n)$

Question 1b)

i.)

$T(n) = 9\,T(n/3) + n^2,\ T(1) = 1$
$T(n/3) = 9\,T(n/3^2) + (n/3)^2$

$T(n) = 9^2\,T(n/3^2) + 9\,(n^2/3^2) + n^2$
$T(n) = 9^2\,T(n/3^2) + 2n^2$
$T(n/3^2) = 9\,T(n/3^3) + n^2/3^4$

$T(n) = 9^3\,T(n/3^3) + 3n^2$

$T(n) = 9^k\,T(n/3^k) + kn^2$

$n = 3^r \rightarrow T(3^r) = 9^k\,T(3^r/3^k) + k\,9^r$

$T(3^k) = 9^k\,T(1) + k\,9^k,\ T(1) = 1$
$T(3^k) = 9^k + k\,9^k,\ n = 3^k \rightarrow k = \log_3(n)$
$T(n) = n^2 + \log_3(n)n^2 = O(n^2)$

ii.)

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

$T(n) = T(n/2^2) + 2.2$
$T(n/2^2) = T(n/2^3) + 2$

$T(n) = T(n/2^3) + 3.2$

$T(n) = T(n/2^k) + 2\,k$
$n = 2^r \rightarrow T(2^r) + T(2^r/2^k) + 2\,k$
$T(1) = 1 \rightarrow T(2^r/2^k) = 1$

$T(n) = 1 + 2\log_2 n = O(\log(n))$

Question 1c)

Bubble Sort

Original List
[4, 9, 7, 3, 5, 2, 1, 6]
1.pass
[4, 9, 7, 3, 5, 2, 1, 6]
[4, 7, 9, 3, 5, 2, 1, 6]
[4, 7, 3, 9, 5, 2, 1, 6]
[4, 7, 3, 5, 9, 2, 1, 6]
[4, 7, 3, 5, 2, 9, 1, 6]
[4, 7, 3, 5, 2, 1, 9, 6]
[4, 7, 3, 5, 2, 1, 6, 9]
2.pass
[4, 3, 7, 5, 2, 1, 6, 9]
[4, 3, 5, 7, 2, 1, 6, 9]
[4, 3, 5, 2, 7, 1, 6, 9]
[4, 3, 5, 2, 1, 7, 6, 9]
[4, 3, 5, 2, 1, 6, 7, 9]
3.pass
[3, 4, 5, 2, 1, 6, 7, 9]
[3, 4, 2, 5, 1, 6, 7, 9]
[3, 4, 2, 1, 5, 6, 7, 9]
4.pass
[3, 2, 4, 1, 5, 6, 7, 9]
[3, 2, 1, 4, 5, 6, 7, 9]
5.pass
[2, 3, 1, 4, 5, 6, 7, 9]
[2, 1, 3, 4, 5, 6, 7, 9]
6.pass
[1, 2, 3, 4, 5, 6, 7, 9]

Insertion Sort

Original List
[4, 9, 7, 3, 5, 2, 1, 6]
1.pass
[4, 7, 9, 3, 5, 2, 1, 6]
2.pass
[3, 4, 7, 9, 5, 2, 1, 6]
3.pass
[3, 4, 5, 7, 9, 2, 1, 6]
4.pass
[2, 3, 4, 5, 7, 9, 1, 6]
5.pass
[1, 2, 3, 4, 5, 7, 9, 6]
6.pass
[1, 2, 3, 4, 5, 6, 7, 9]

2. I used algorithms and codes from lecture slides and lesson book. Some part of the code changed by me because lecture slides functions definition and homework function definition was different. I gave information about the order for every program calling on first line.

// Arrays created randomly – no order.

R1K	Time	Movement	Comparisons
Selection Sort	1.356	999	507351
Merge Sort	0,149	19952	8683
Quick Sort	0,141	16753	13596

R6K	Time	Movement	Comparisons
Selection Sort	61.617	5999	18054542
Merge Sort	1.445	151616	67836
Quick Sort	1.06	163232	93288

R12K	Time	Movement	Comparisons
Selection Sort	209.187	11999	72089067
Merge Sort	2.734	327232	147760
Quick Sort	1.912	321248	201671

R18K	Time	Movement	Comparisons
Selection Sort	468.047	17999	162143111
Merge Sort	4.491	510464	231942
Quick Sort	3.52	498975	323074

// Arrays created descending order.

D1K	Time	Movement	Comparisons
Selection Sort	2.567	999	999999
Merge Sort	0,124	19952	5044
Quick Sort	1.546	3996	501499

D6K	Time	Movement	Comparisons
Selection Sort	81.07	5999	35999999
Merge Sort	1.264	151616	39152
Quick Sort	50.948	23996	18008999

D12K	Time	Movement	Comparisons
Selection Sort	318.196	11999	143999999
Merge Sort	1.365	327232	84304
Quick Sort	193.222	47996	72017999

D18K	Time	Movement	Comparisons
Selection Sort	739.46	17999	323999999
Merge Sort	2.199	510464	130592
Quick Sort	437.763	71996	162026999

// Arrays created ascending order.

A1K	Time	Movement	Comparisons
Selection Sort	2.599	999	999999
Merge Sort	0,123	19952	5044
Quick Sort	1.521	3996	501499

A6K	Time	Movement	Comparisons
Selection Sort	81.537	5999	35999999
Merge Sort	0.87	151616	39152
Quick Sort	50.697	23996	18008999

A12K	Time	Movement	Comparisons
Selection Sort	335.912	11999	143999999
Merge Sort	1.396	327232	84304
Quick Sort	197.306	47996	72017999

A18K	Time	Movement	Comparisons
Selection Sort	725.721	17999	323999999
Merge Sort	2.5	510464	130592
Quick Sort	436.787	71996	162026999

// Array first part is ascending order, last part is descending order.

M1K	Time	Movement	Comparisons
Selection Sort	1.439	999	538991
Merge Sort	0.11	19952	6864
Quick Sort	709	200384	140644

M6K	Time	Movement	Comparisons
Selection Sort	53.8	5999	20754881
Merge Sort	0,866	151616	49319
Quick Sort	63.698	6773996	18008999

M12K	Time	Movement	Comparisons
Selection Sort	291.304	11999	110979616
Merge Sort	1.591	327232	113812
Quick Sort	105.765	27332090	19973515

M18K	Time	Movement	Comparisons
Selection Sort	547.126	17999	216862966
Merge Sort	2.269	510464	130638
Quick Sort	172.622	43780936	35129413

Notes: Some time computations looks zero in Dijkstra but it is not zero, it is about Dijkstra system. Normally we can see in our IDE.

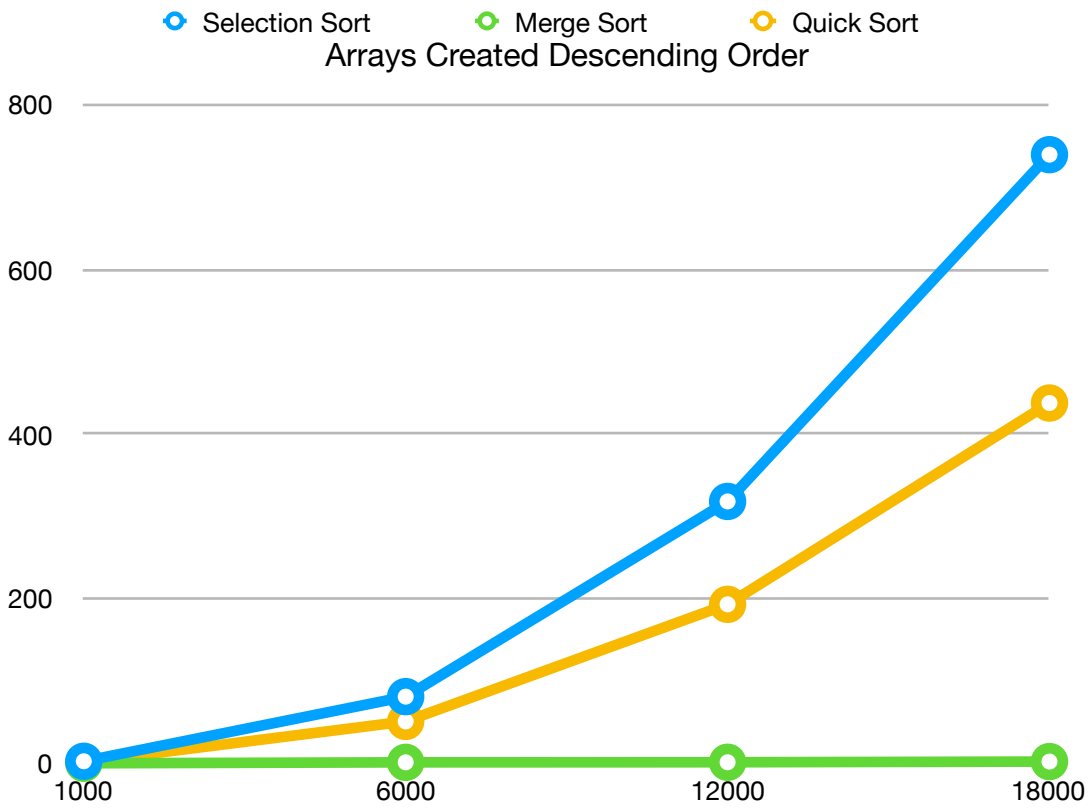
Outputs show that:

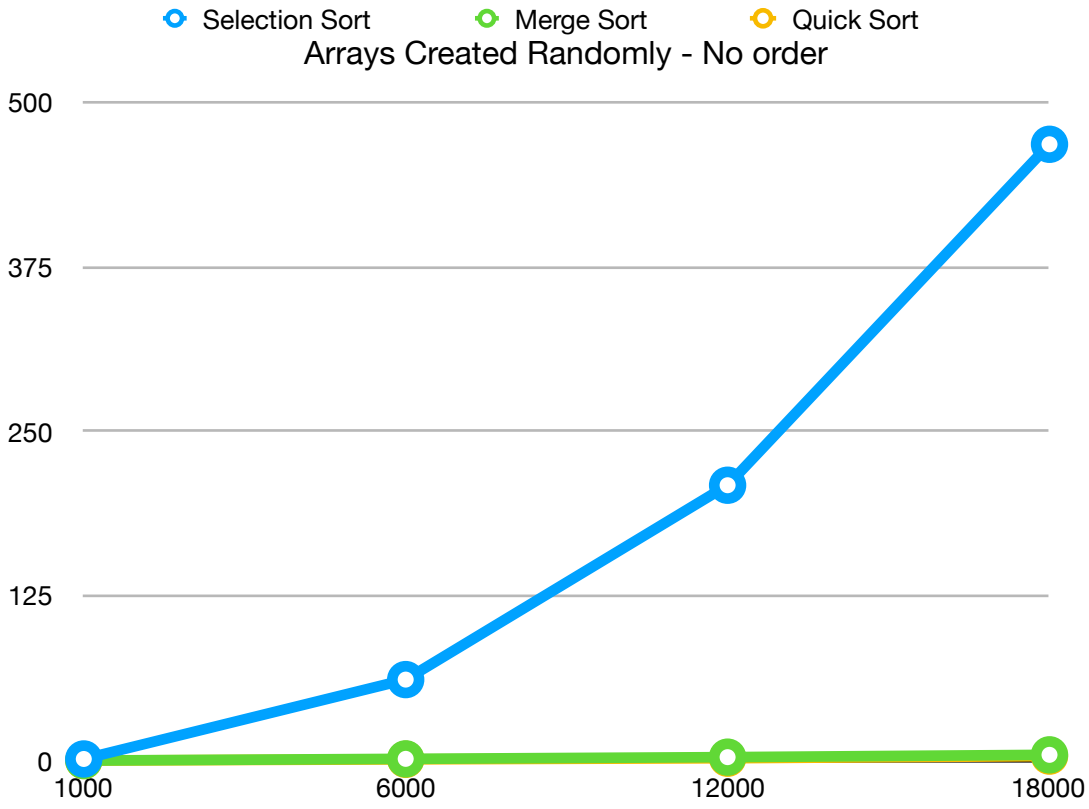
- Quick sort is slow when the array is already sorted and we choose the first element as the pivot.
- I can say that merge sort is working very well and If I have an option for choose the sorting algorithm, my choice was being merge sort because this algorithm can handle with the all possible situations.
- In the 4. Question, both ascending and descending order and I can say that quick sort takes much more time according to selection sort for 6000 elements in array.

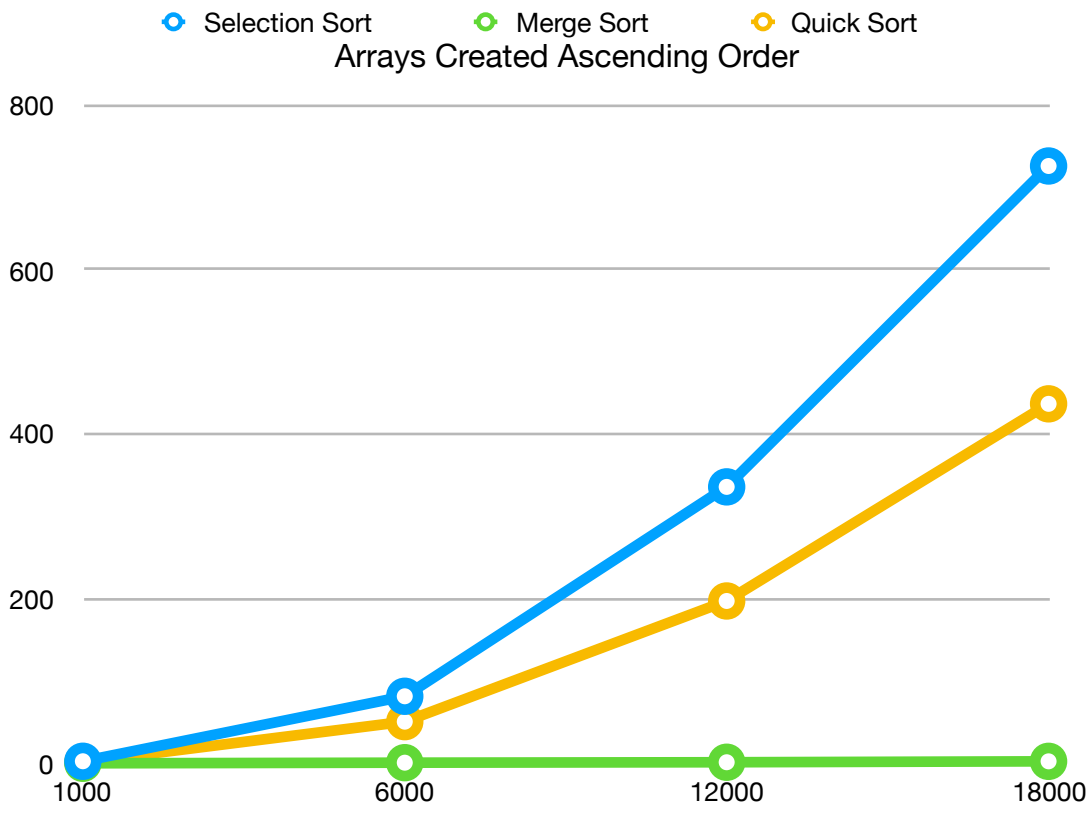
Analysis: Results show that the best handler of algorithms is merge sort, merge sort algorithms handle with all possible situations perfectly. If we have randomly created unsorted list merge sort and quick sort two of them can use by us but selection sort can not useful for 1000+ situations. The computations took too much time for computer. If given list is sorted, quick sort is not working well, because quick sort precondition is unsorted list. If the list is sorted, quick sort took too much time we can see from the tables. I encounter with interesting thing that for 6000 elements half - half ordered in array, the selection sort works faster than the quick sort but in

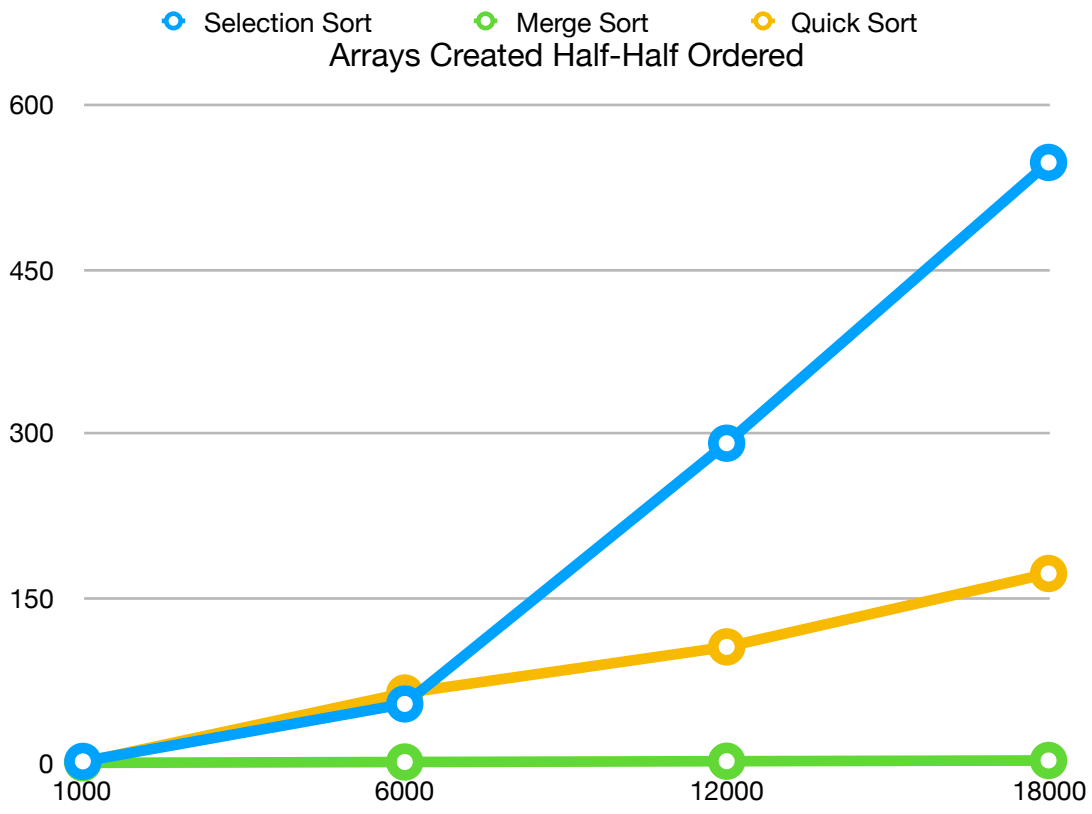
ascending and descending ordered arrays, quick sort always faster than the selection sort. It was amazing. We can see from the tables below the analysis. These are not different from the sorting algorithms time complexity. Selection sort is O(n^2), Merge sort is O(n.log(n)), quick sort is O(n.log(n)) but worst case is O(n^2) because sorted list took much time if we use quick sort. Quick sort prerequisite is unsorted list.

According to Time Comparisons Of Sort Algorithms on Graphs






Selection Sort
Merge Sort
Quick Sort
Arrays Created Ascending Order
800
600
400
200
0
1000
6000
12000
18000


Selection Sort
Merge Sort
Quick Sort
Arrays Created Half-Half Ordered
600
450
300
150
0
1000
6000
12000
18000