QUESTION 1

Part A:

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

Part B:

Equation 1:

$T(n) = 9T(n/3) + n^2$, $T(1) = 1$, let $T(n) = Cn^2\log n$ where C is a constant:

$Cn^2\log n = 9(Cn^2[\log n-\log 3] / 9) + n^2$
$Cn^2\log n = Cn^2\log n - Cn^2\log 3 + n^2$

Inserting $Cn^2\log 3 = \Theta(n^2)$, $Cn^2\log n = \Theta(n^2\log n)$, $n^2 = \Theta(n^2)$

$\Theta(n^2\log n) = \Theta(n^2\log n) - \Theta(n^2) + \Theta(n^2)$
$\Theta(n^2\log n) = \Theta(n^2\log n)$

Thus, **$T(n) = \Theta(n^2\log n)$** by mathematical induction

Equation 2:

$T(n) = T(n/2) + 2$, $T(1) = 1$, let $T(n) = 2\log_2(n) + 1$

$2\log_2(n) + 1 = 2(\log_2(n) – 1) + 1 + 2$
$2\log_2(n) + 1 = 2\log_2(n) + 1$

Thus, $T(n) = 2\log_2(n) + 1$ by mathematical induction, **$T(n) = \Theta(\log n)$**

Part C:

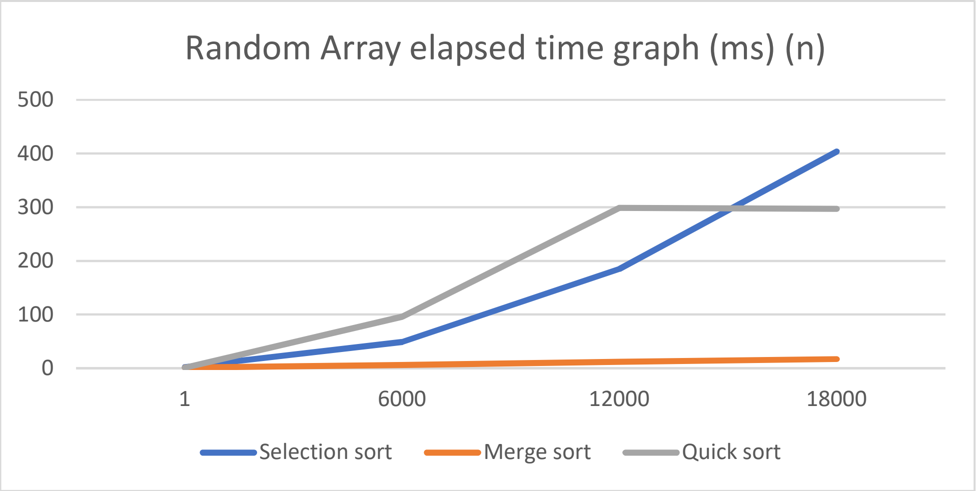
Insertion sort:

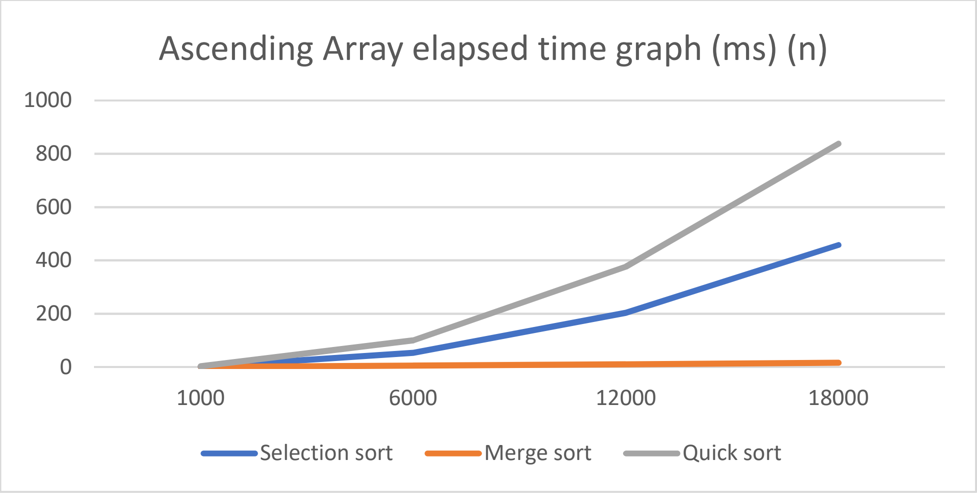
[4,9,7,3,5,2,1,6] => [4,7,9,3,5,2,1,6] => [3,4,7,9,5,2,1,6] => [3,4,5,7,9,2,1,6] => [2,3,4,5,7,9,1,6] => [1,2,3,4,5,7,9,6] => [1,2,3,4,5,6,7,9]

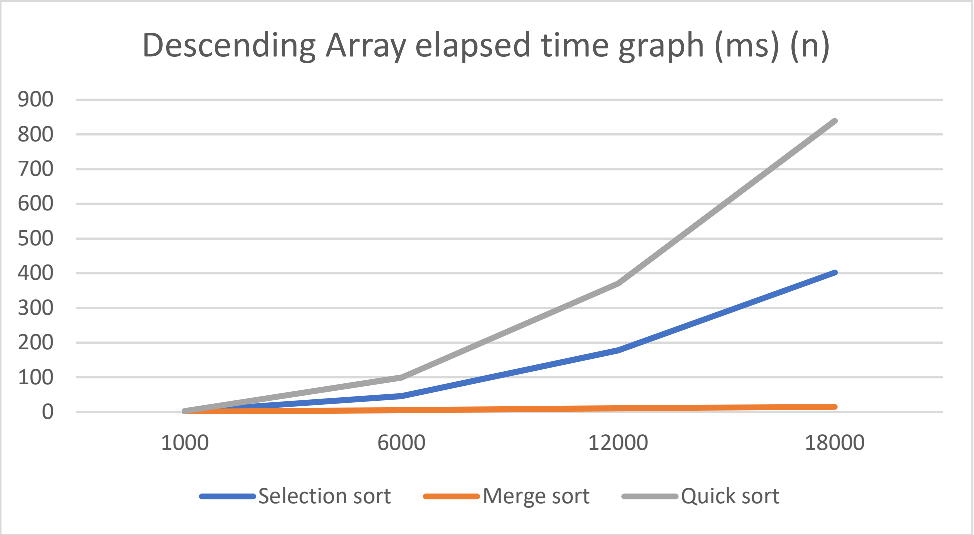
<u>Bubble sort:</u>

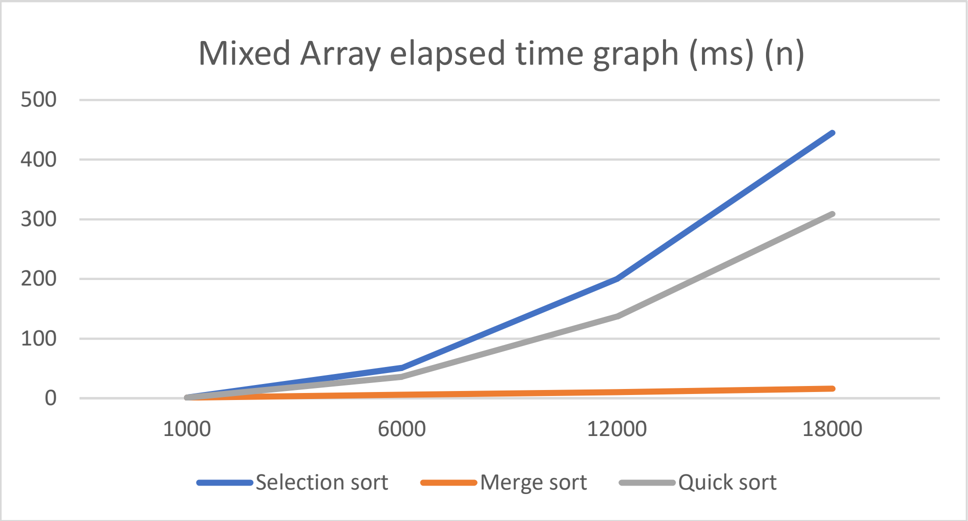
[4,9,7,3,5,2,1,6] => [4,7,9,3,5,2,1,6] => [4,7,3,9,5,2,1,6] => [4,7,3,5,9,2,1,6] => [4,7,3,5,2,9,1,6] => [4,7,3,5,2,1,9,6] => **[4,7,3,5,2,1,6,9]** => [4,3,7,5,2,1,6,9] => [4,3,5,7,2,1,6,9] => [4,3,5,2,7,1,6,9] => **[4,3,5,2,1,7,6,9]** => [4,3,5,2,1,6,7,9] => [3,4,5,2,1,6,7,9] => [3,4,2,5,1,6,7,9] => **[3,4,2,1,5,6,7,9]** => [3,2,4,1,5,6,7,9] => **[3,2,1,4,5,6,7,9]** => [2,3,1,4,5,6,7,9] => **[2,1,3,4,5,6,7,9]** => **[1,2,3,4,5,6,7,9]**

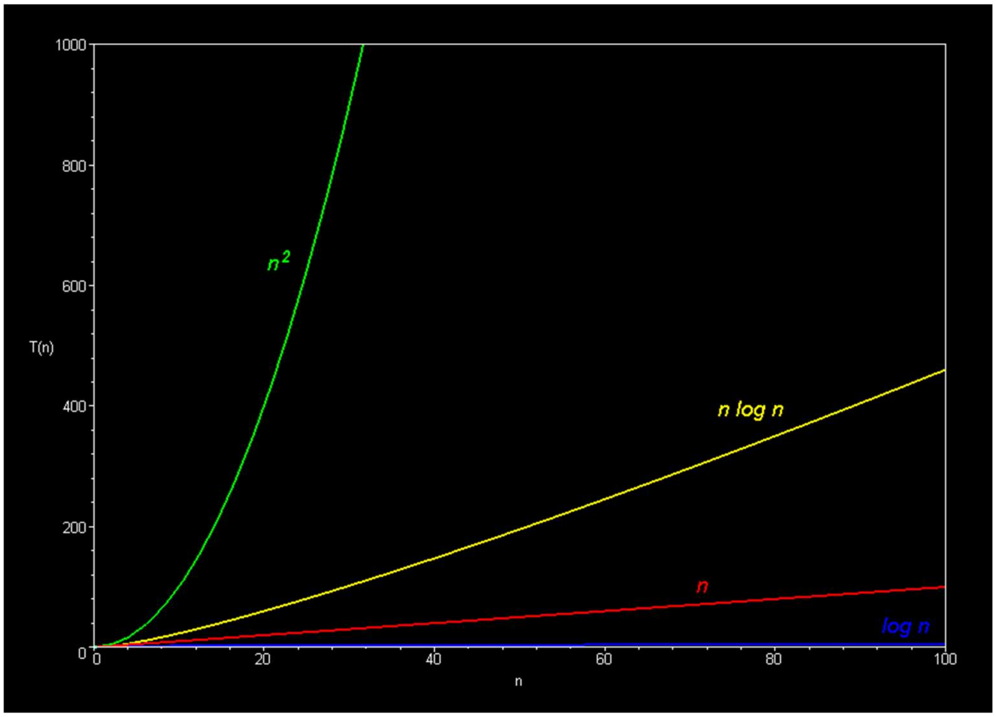
QUESTION 3





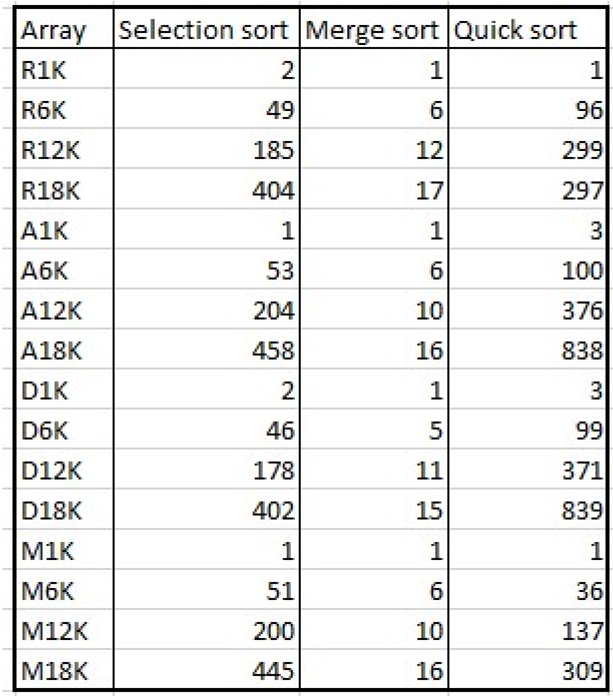

Descending Array elapsed time graph (ms) (n)
900
800
700
600
500
400
300
200
100
0
1000
6000
12000
18000
Selection sort
Merge sort
Quick sort


Mixed Array elapsed time graph (ms) (n)
500
400
300
200
100
0
1000
6000
12000
18000
Selection sort
Merge sort
Quick sort


1000
800
600
400
200
0
T(n)
n²
n log n
n
log n
0
20
40
60
80
100
n

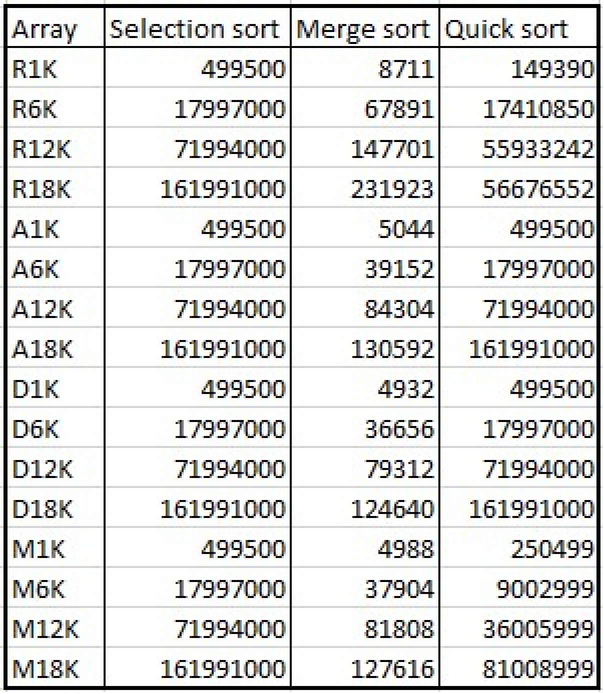
(Theoretical results source:

http://science.slc.edu/jmarshall/courses/2002/spring/cs50/BigO/index.html)



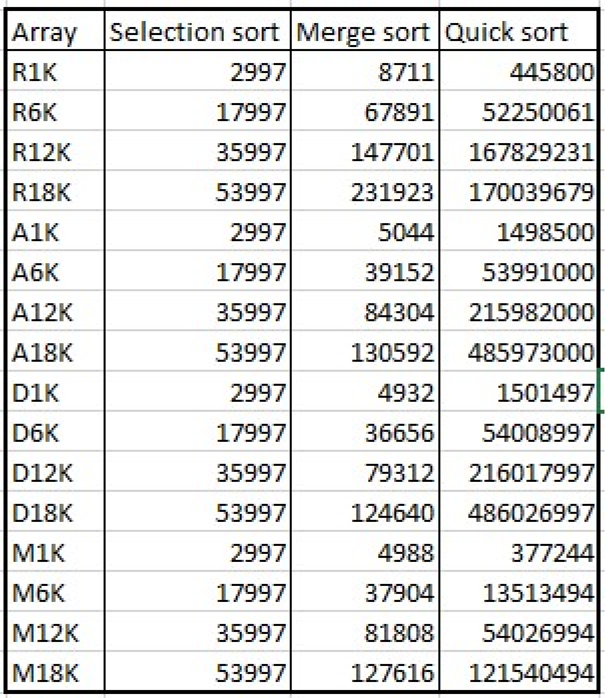
Array	Selection sort	Merge sort	Quick sort
R1K	2	1	1
R6K	49	6	96
R12K	185	12	299
R18K	404	17	297
A1K	1	1	3
A6K	53	6	100
A12K	204	10	376
A18K	458	16	838
D1K	2	1	3
D6K	46	5	99
D12K	178	11	371
D18K	402	15	839
M1K	1	1	1
M6K	51	6	36
M12K	200	10	137
M18K	445	16	309

Time elapsed (ms)



Array	Selection sort	Merge sort	Quick sort
R1K	499500	8711	149390
R6K	17997000	67891	17410850
R12K	71994000	147701	55933242
R18K	161991000	231923	56676552
A1K	499500	5044	499500
A6K	17997000	39152	17997000
A12K	71994000	84304	71994000
A18K	161991000	130592	161991000
D1K	499500	4932	499500
D6K	17997000	36656	17997000
D12K	71994000	79312	71994000
D18K	161991000	124640	161991000
M1K	499500	4988	250499
M6K	17997000	37904	9002999
M12K	71994000	81808	36005999
M18K	161991000	127616	81008999

Compare count



Array	Selection sort	Merge sort	Quick sort
R1K	2997	8711	445800
R6K	17997	67891	52250061
R12K	35997	147701	167829231
R18K	53997	231923	170039679
A1K	2997	5044	1498500
A6K	17997	39152	53991000
A12K	35997	84304	215982000
A18K	53997	130592	485973000
D1K	2997	4932	1501497
D6K	17997	36656	54008997
D12K	35997	79312	216017997
D18K	53997	124640	486026997
M1K	2997	4988	377244
M6K	17997	37904	13513494
M12K	35997	81808	54026994
M18K	53997	127616	121540494

Move count

Discussion:

The quick sort algorithm is supposed to work faster than my results, I suppose this is because of my inefficient partition implementation. It can be observed that it’s slower than selection sort for most cases, which shouldn’t happen with a proper efficient implementation. The selection sort and merge sort however, resulted as expected. Merge sort is correlated with nlogn while selection sort is correlated with n^2.

Merge sort is way more efficient than other two, at least opposing to my inefficient quick sort algorithm. However, it creates temporary arrays while merging subarrays, which can be an issue if memory is limited.

Quick sort is very inefficient for already sorted arrays, since it takes first integer in the array as a pivot. It has similar results for ascending/descending array and random arrays because it sorts in a descending form.

Selection sort have same move and compare counts for 4 of the array kinds, and wouldn’t change anyways if the size doesn’t change, because it will look all of the array/subarray everytime it runs.