/**

*Title: Algorithm Efficiency and Sorting

*Author: Melis Atun

*ID: 21901865

*Section: 1

*Assignment: 1

*Description: Answers to questions 1, 2 and 3

*/

QUESTION 1

- $T(N) = 3T(n/3) + n$; where $T(1) = 1$ and n is an exact power of 3

  $T(N) = 3T(n/3) + n$ **(first equation)**

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

  Substituting T(n/3) to the first equation yields

  $T(N) = 3[3T(n/3^2)+ n/3] + n$

  $T(N) = 3^2T(n/3^2)+ n + n$ **(second equation)**

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

Substituting $T(n/3^2)$to the second equation yields

$T(N) = 3^2[3T(n/3^3) + n/3^2] + 2n$

$T(N) = 3^3T(n/3^3) + n + 2n$ **(third equation)**

Therefore, $T(N) = 3^kT(n/3^k) + kn$

$T(n/3^k) = T(1) \Rightarrow n/3^k = 1 \Rightarrow n = 3^k \Rightarrow k = \log n$

Therefore, $T(N) = 3^kT(1) + kn$

$T(N) = n.1 + n\log n$

$T(N) = n + n\log n$

Consequently, the answer is $O(n\log n)$.

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

  $T(N) = 2T(n - 1) + n^2$ **(first equation)**

  Therefore, $T(n - 1) = 2T(n - 2) + (n - 1)^2$

  Substituting $T(n - 1)$ to the first equation yields

  $T(N) = 2[2T(n - 2) + (n - 1)^2] + n^2$

  $T(N) = 2^2T(n - 2) + 2(n - 1)^2 + n^2$ **(second equation)**

Therefore, $T(n - 2) = 2T(n - 3) + (n - 2)^2$

Substituting T(n - 2) to the second equation yields

$T(N) = 2^2[2T(n - 3) + (n - 2)^2] + 2(n - 1)^2 + n^2$

$T(N) = 2^3T(n - 3) + 2^2(n - 2)^2 + 2(n - 1)^2 + n^2$ **(third equation)**

Therefore, $T(N) = 2^kT(n - k) + 2^{k-1}(n - (k - 1))^{k-1} + 2^{k-2}(n - (k - 2))^{k-1} + n^{k-1}$

$T(n - k) = T(1) \Rightarrow n - k = 1 \Rightarrow k = n - 1$

$T(N) = 2^{n-1}T(1) + 2^{n-2}(n - n)^{k-1} + 2^{n-3}(3)^{k-1} + n^{n-2}$

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

Consequently, the answer is $O(n^n)$ for n > 3.

- $T(N) = 3T(n/4) + n\log n$, where T(1) = 1 and n is an exact power of 4

$T(N) = 3T(n/4) + n\log n$ **(first equation)**

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

Substituting T(n/4) to the first equation yields

$T(N) = 3[3T(n/4^2) + n/4\log n(n/4)] + n\log n$

$T(N) = 3^2T(n/4^2) + 3n/4\log(n/4) + n\log n$ **(second equation)**

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

Substituting $T(n/4^2)$ to the second equation yields

$T(N) = 3^2[3T(n/4^3) + n/4^2 \log(n/4^2)] + 3n/4\log(n/4) + n\log n$

$T(N) = 3^3 T(n/4^3) + 3^2 n/4^2 \log(n/4^2) + 3n/4\log(n/4) + n\log n$ **(third equation)**

Therefore, $T(N) = 3^k T(n/4^k) + 3^{k-1} n/4^{k-1} \log(n/4^{k-1}) + 3n/4\log(n/4) + n\log n$

$T(n/4^k) = T(1) \Rightarrow n/4^k = 1 \Rightarrow k = \log n$

$T(N) = 3^{logn} + n\log n$

Consequently, the answer is $O(n\log n)$.

- $T(N) = 3T(n/2) + 1$, where $T(1) = 1$ and n is an exact power of 2

  $T(N) = 3T(n/2) + 1$ **(first equation)**

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

  Substituting $T(n/2)$ to the first equation yields

  $T(N) = 3[3T(n/2^2) + 1] + 1$

  $T(N) = 3^2 T(n/2^2) + 3 + 1$ **(second equation)**

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

Substituting $T(n/2^2)$ to the second equation yields

$T(N) = 3^2[3T(n/2^3) + 1] + 2^2$

$T(N) = 3^3T(n/2^3) + 3^2 + 2^2$ **(third equation)**

Therefore, $T(N) = 3^kT(n/2^k) + 3^{k-1} + 2^{k-1}$

$T(n/2^k) = T(1) \Rightarrow n/2^k = 1 \Rightarrow k = log_2 n$

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

$3^{log_2 n} = n^{log_2 3}$

Consequently, the answer is $O(n^{log_2 3})$.

- [5, 6, 8, 4, 10, 2, 9, 1, 3, 7]
    - ***Bubble Sort***

5	6	8	4	10	2	9	1	3	7

5	6	8	4	10	2	9	1	3	7

5	6	8	4	10	2	9	1	3	7

5	6	4	8	10	2	9	1	3	7

5	6	4	8	10	2	9	1	3	7

5	6	4	8	2	10	9	1	3	7

5	6	4	8	2	9	10	1	3	7

5	6	4	8	2	9	1	10	3	7

5	6	4	8	2	9	1	3	10	7

5	6	4	8	2	9	1	3	7	10

5	6	4	8	2	9	1	3	7	**10**

5	6	4	8	2	9	1	3	7	**10**

5	4	6	8	2	9	1	3	7	**10**

5	4	6	8	2	9	1	3	7	**10**

5	4	6	2	8	9	1	3	7	**10**

5	4	6	2	8	9	1	3	7	**10**

5	4	6	2	8	1	9	3	7	**10**

5	4	6	2	8	1	3	9	7	**10**

5	4	6	2	8	1	3	7	9	**10**

5	4	6	2	8	1	3	7	**9**	**10**

4	5	6	2	8	1	3	7	**9**	**10**

4	5	6	2	8	1	3	7	**9**	**10**

4	5	2	6	8	1	3	7	**9**	**10**

4	5	2	6	8	1	3	7	**9**	**10**

4	5	2	6	1	8	3	7	**9**	**10**

4	5	2	6	1	3	8	7	**9**	**10**

4	5	2	6	1	3	7	8	**9**	**10**

4	5	2	6	1	3	7	**8**	**9**	**10**

4	5	2	6	1	3	7	**8**	**9**	**10**

4	2	5	6	1	3	7	**8**	**9**	**10**

4	2	5	6	1	3	7	**8**	**9**	**10**

4	2	5	1	6	3	7	**8**	**9**	**10**

4	2	5	1	3	6	7	**8**	**9**	**10**

4	2	5	1	3	6	7	**8**	**9**	**10**

4	2	5	1	3	6	**7**	**8**	**9**	**10**

2	4	5	1	3	6	**7**	**8**	**9**	**10**

2	4	5	1	3	6	**7**	**8**	**9**	**10**

2	4	1	5	3	6	**7**	**8**	**9**	**10**

2	4	1	3	5	6	**7**	**8**	**9**	**10**

2	4	1	3	5	**6**	**7**	**8**	**9**	**10**

2	4	1	3	5	**6**	**7**	**8**	**9**	**10**

2	4	1	3	5	**6**	**7**	**8**	**9**	**10**

2	1	4	3	5	**6**	**7**	**8**	**9**	**10**

2	1	3	4	5	**6**	**7**	**8**	**9**	**10**

2	4	1	3	**5**	**6**	**7**	**8**	**9**	**10**

2	4	1	3	**5**	**6**	**7**	**8**	**9**	**10**

2	4	1	3	**5**	**6**	**7**	**8**	**9**	**10**

2	1	4	3	**5**	**6**	**7**	**8**	**9**	**10**

2	1	3	**4**	**5**	**6**	**7**	**8**	**9**	**10**

2	1	3	**4**	**5**	**6**	**7**	**8**	**9**	**10**

1	2	3	**4**	**5**	**6**	**7**	**8**	**9**	**10**

1	2	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

2	1	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

- ***Selection Sort***

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**

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

- Recurrence relation of quick sort algorithm

Worst-case of quicksort: Array is sorted and pivot is one of the corner elements

Therefore, T(N) = T(n - 1) + n **(first equation)**

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

Substituting T(n - 1) to the first equation yields

T(N) = T(n - 2) + (n - 1) + n **(second equation)**

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

Substituting T(n - 3) to the second equation yields

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

Therefore,

T(N) = T(1) + 2 + 3 + 4 + … + (n - 1) + n

T(N) = 1 + 2 + 3 + 4 + … + (n - 1) + n

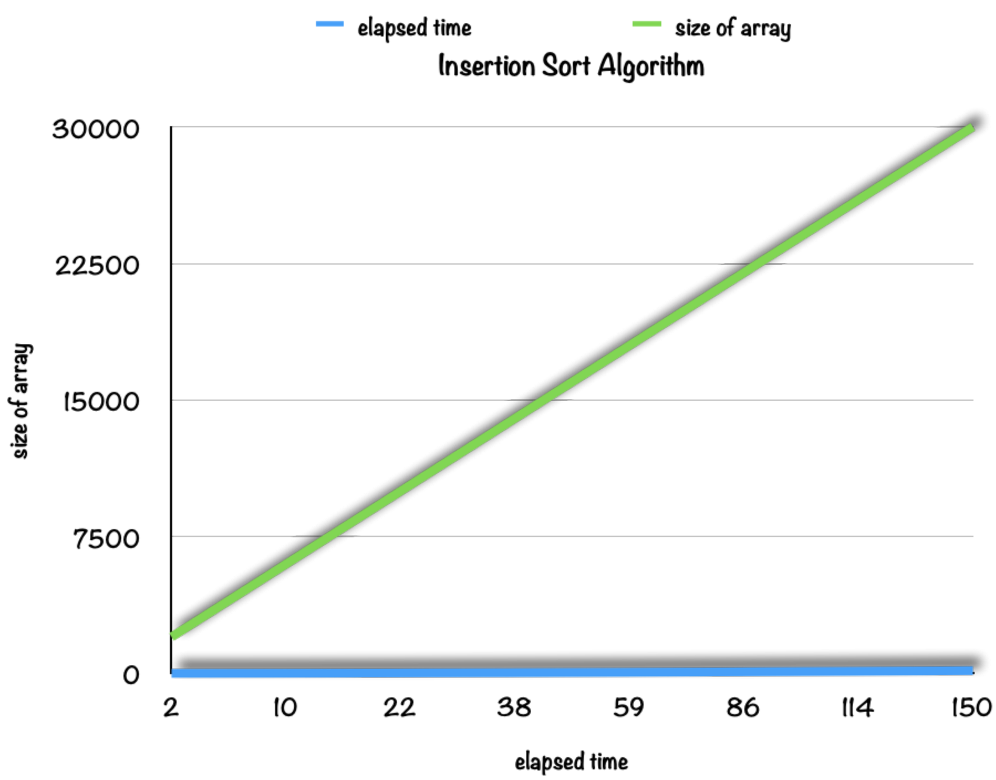
T(N) = [n(n +1) / 2] - 1 = [(n^2+ n) / 2] - 1

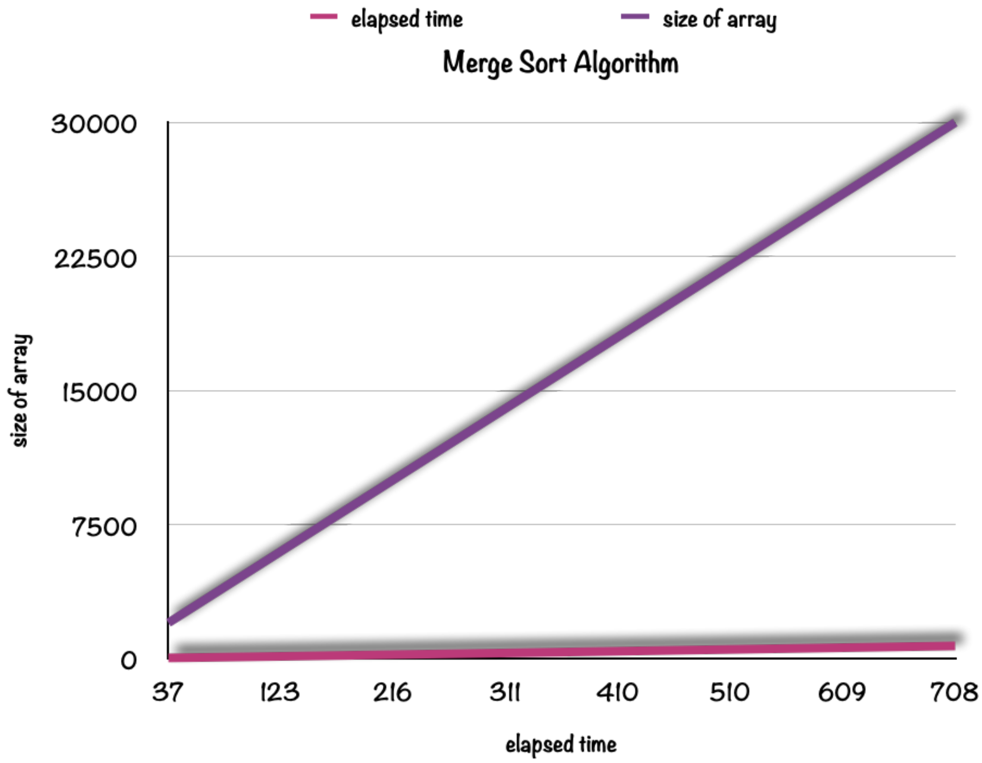
Consequently, the answer is O(N^2) for the worst-case of quick sort algorithm.

QUESTION 2

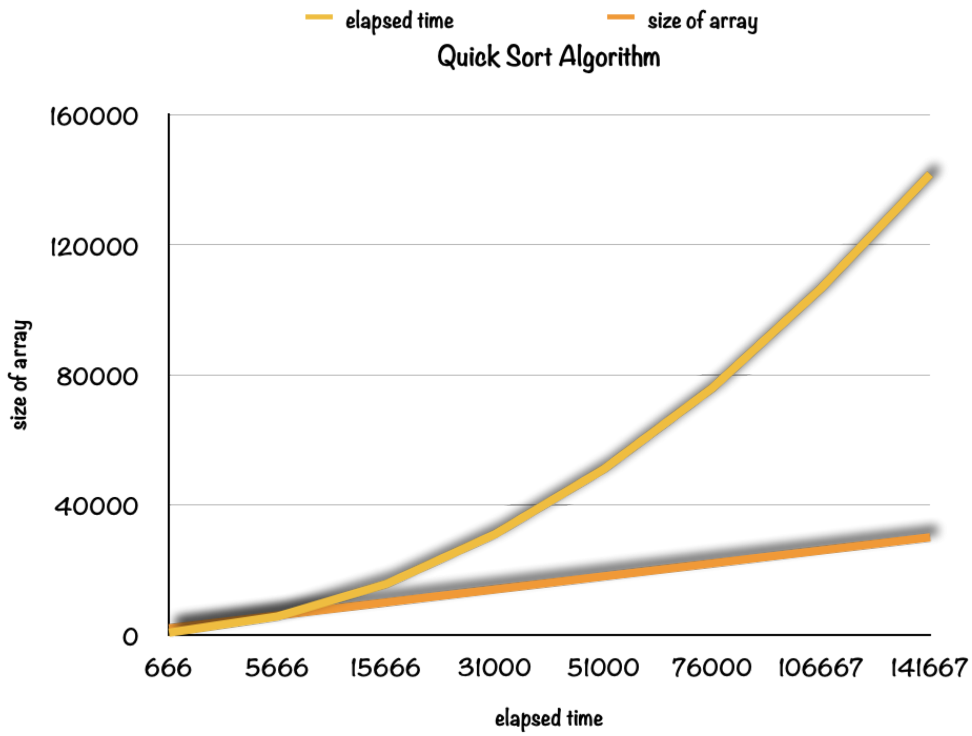
```
melo — ssh melis.atun@dijkstra.ug.bilkent.edu.tr — 148×54
Last login: Mon Oct 25 22:16:24 2021 from 139.179.221.144
-bash: warning: setlocale: LC_CTYPE: cannot change locale (UTF-8): No such file or directory
-bash-4.2$ ls
main.cpp  melo  sorting.cpp  sorting.h
-bash-4.2$ ./melo
0 1 2 3 4 5 6 7 8 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              2 ms        1999000        1022336
6000             10 ms        5999000        9030821
10000             22 ms        9999000       25371907
14000             38 ms       13999000       48522793
18000             59 ms       17999000       80910142
22000             86 ms       21999000      121556859
26000            114 ms       25999000      168209663
30000            150 ms       29999000      225583458
-----------------------------------------------------
Part b - Time Analysis of Merge Sort
Array Size   Time Elapsed      compCount      moveCount
2000             37 ms       82474583       57426861
6000            123 ms      283515697      197581899
10000            216 ms      499096478      347852826
14000            311 ms      720375328      502945776
18000            410 ms      948370252      662432084
22000            510 ms     1187014895      828646965
26000            609 ms     1421169840      993365280
30000            708 ms     1654505254     1157810418
-----------------------------------------------------
Part c - Time Analysis of Quick Sort
Array Size   Time Elapsed      compCount      moveCount
2000        666.667 ms        8049658          37450
6000        5666.67 ms       72169443         112525
10000        15666.7 ms      200323480         187418
14000           31000 ms      392484586         262402
18000           51000 ms      648628437         337393
22000           76000 ms      968792065         412055
26000          106667 ms     1352935845         487557
30000          141667 ms     1801097989         562079
-----------------------------------------------------
Part d - Time Analysis of Radix Sort
Array Size   Time Elapsed
2000             33 ms
6000            101 ms
10000            169 ms
14000            292 ms
18000            377 ms
22000            461 ms
26000            544 ms
30000            629 ms
-bash-4.2$
```

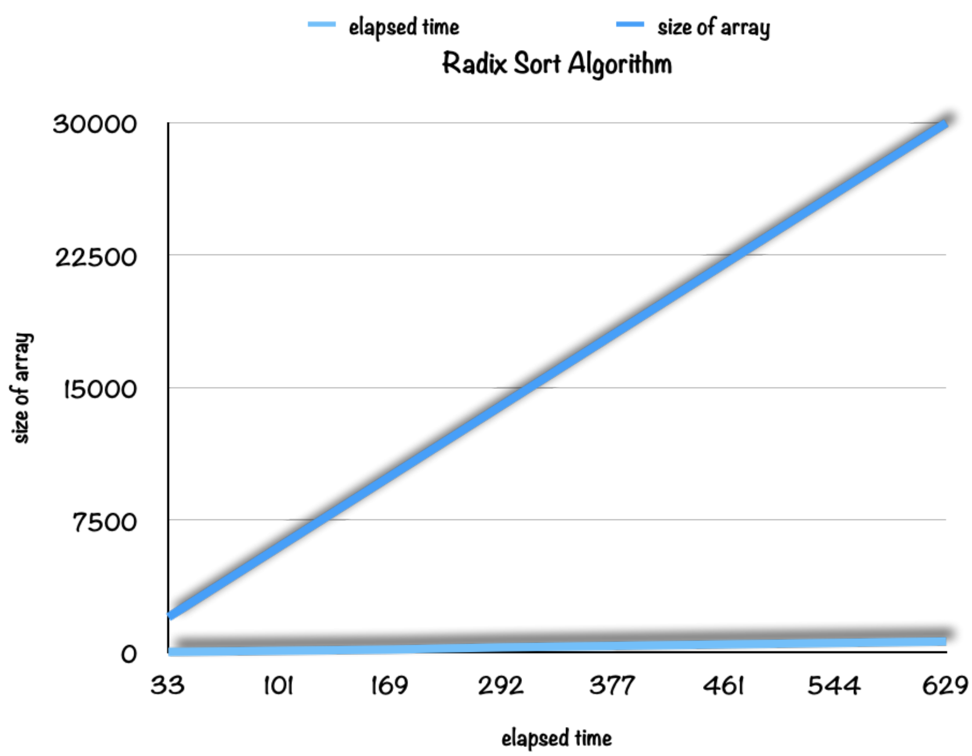
QUESTION 3

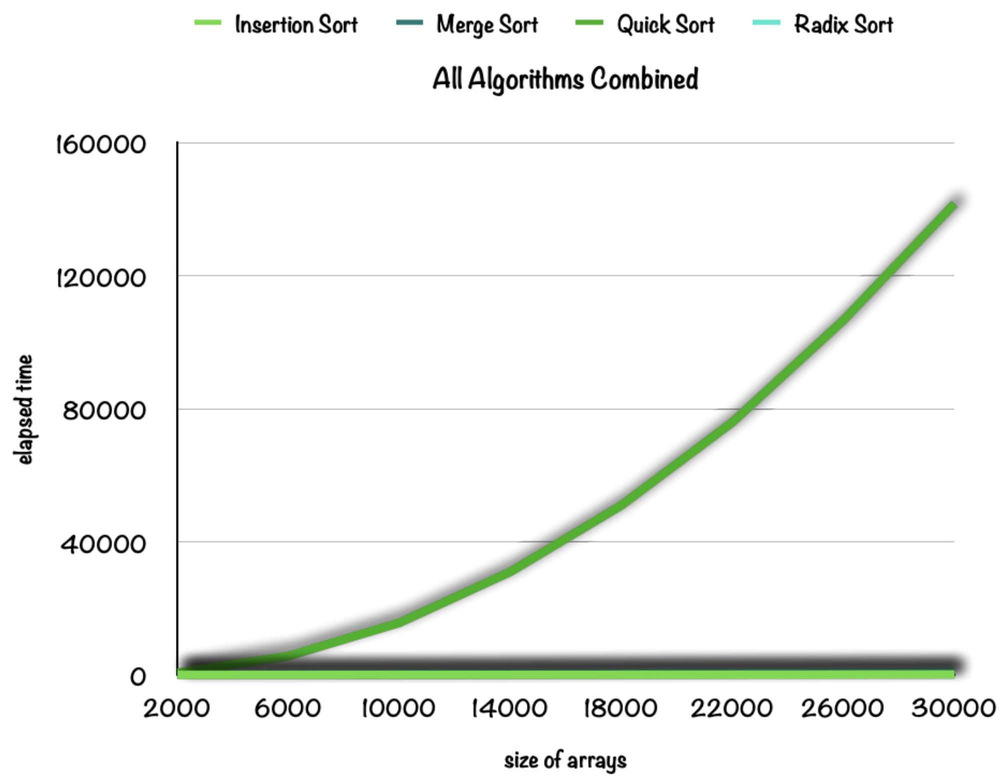

elapsed time
size of array
Insertion Sort Algorithm
30000
22500
15000
7500
0
size of array
2
10
22
38
59
86
114
150
elapsed time



elapsed time
size of array
Merge Sort Algorithm
30000
22500
15000
7500
0
size of array
37
123
216
311
410
510
609
708
elapsed time


elapsed time
size of array
Quick Sort Algorithm
160000
120000
80000
40000
0
size of array
666
5666
15666
31000
51000
76000
106667
141667
elapsed time


elapsed time
size of array
Radix Sort Algorithm
30000
22500
15000
7500
0
size of array
33
101
169
292
377
461
544
629
elapsed time



Questions:

- **Interpret and compare your empirical results with the theoretical ones. Explain any differences between the empirical and theoretical results, if any.**

Empirical results and theoretical results have similarities but they also have inconsistencies. Since the observed time complexities can change from computer to computer, the data that is obtained from observing the algorithms' time complexities may have differences compared with the theoretical data. Hence, it can be observed that theoretical and empirical results agree but there are little differences between them due to experimental errors. Also, insertion sort, merge sort and radix sort algorithms are faster compared to quick sort for big array sizes. Hence, observing their time complexities is more difficult. Last of all, since every computer has different qualifications, some differences may occur between the results.

- **How would the time complexity of your program change if you applied the sorting algorithms to an array of increasing numbers instead of randomly generated numbers?**

  The time complexity would certainly decrease because sorting the algorithms and then finding the time complexity takes more time compared to finding the time complexity of an already sorted array.