Title: Algorithm Efficiency and Sorting
Author: Deniz Çalkan
ID: 21703994
Section: 1
Assignment: 1
Description: Answers of questions 1, 2 and 3.

Q1a:

• T(n) = 5T(n/3) + n.logn, where T(1) = 1 and n is an exact power of 3.

Substitute T(n/3) = 5T(n/9) + n/3.logn/3

T(n) = 25T(n/9) + 5n/3.logn/3 + n.logn

Substitute T(n/9) = 5T(n/27) + n/9.logn/9

T(n) = 125T(n/27) + 25n/9.logn/9 + 5n/3.logn/3 +nlogn

.
.
.

T(n) = 5^k T(n/3^k) + 5^{k-1} n/3^{k-1} .logn/3^{k-1} + 5^{k-2} n/3^{k-2} .logn/3^{k-2} +...+nlogn

Since T(1) = 1, n/3^k = 1 , n = 3^k, log_3^n = k

T(n) = $5^{log_3^n}$ T(1) + ($3.5^{log_3^n}$ .n/5. $3^{log_3^n}$).log3n/$3^{log_3^n}$ + ($9.5^{log_3^n}$ .n/25. $3^{log_3^n}$).log9n/$3^{log_3^n}$ +...+nlogn

Since $5^{log_3^n}$ = $n^{log_3^5}$ and $3^{log_3^n}$ = n

T(n) = $n^{log_3^5}$ + ($3.5^{log_3^n}$ / 5).log3 + ($9.5^{log_3^n}$ / 25). log9 + … + nlogn

Ignore constants and low order terms

T(n) =O($n^{log_3^5}$)

Since n is an exact power of 3

T(n) = O $(n/3)^5$

Ignore constants

ANSWER: O(n^5)

• T(n) = T(n−1) + n^2 , where T(1) = 1.

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

T(n) = T(n−2) + $(n - 1)^2$ + n^2

Substitute T(n - 2) = T(n - 3) + $(n - 2)^2$

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

.
.
.

T(n) = T(n − k) + $(n - k)^2$ + $(n - (k - 1))^2$ + … + n^2

Since T(1) = 1, n - k = 1, k = n - 1

T(n) = 1 + 1^2 + 2^2 + …+ n^2

T(n) = 1 + n.(n+1).(2n+1) / 6

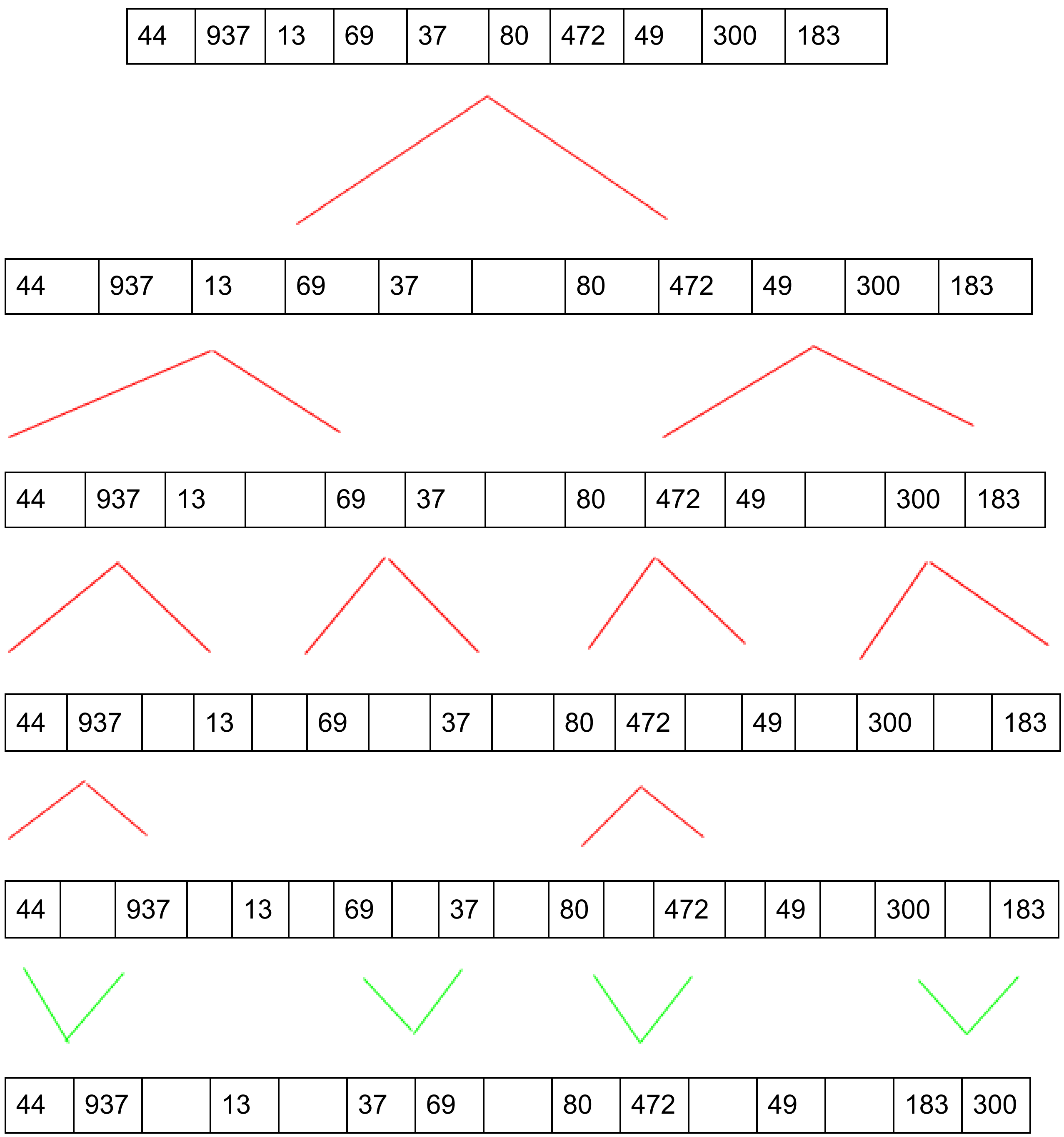
Ignore constants and low order terms

T(n) = O(n^3)

ANSWER: O(n^3)

Q1b:

Merge Sort



13	44	937		37	69		49	80	472		183	300

13	37	44	69	937		49	80	183	300	472

13	37	44	49	69	80	183	300	472	937

Red indicates divide

Green indicates merge

Insertion Sort

Original List

44	937	13	69	37	80	472	49	300	183

After pass 1

44	937	13	69	37	80	472	49	300	183

After pass 2

13	44	937	69	37	80	472	49	300	183

After pass 3

13	44	69	937	37	80	472	49	300	183

After pass 4

13	37	44	69	937	80	472	49	300	183

After pass 5

13	37	44	69	80	937	472	49	300	183

After pass 6

13	37	44	69	80	472	937	49	300	183

After pass 7

13	37	44	49	69	80	472	937	300	183

After pass 8

13	37	44	49	69	80	300	472	937	183

After pass 9

13	37	44	49	69	80	183	300	472	937

Blue indicates sorted

Pink indicates the key

Green indicates unsorted

Q1c:

For quicksort the worst case happens when the pivot is the largest or the smallest item in the array so we will make a recursive call for an array with size n - 1 and for an array with size 0 so no recursive calls are made.

```
void quicksort(DataType theArray[], int first, int last) {
    int pivotIndex;

  if (first < last) {

      partition(theArray, first, last, pivotIndex);   -> n

      quicksort(theArray, first, pivotIndex-1);     -> T(0)
      quicksort(theArray, pivotIndex+1, last);     -> T(n - 1)
  }
}
```

T(n) = T(n - 1) + n where T(0) = 0

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

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

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

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

.
.
.

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

Since T(0) = 0, n = k

T(n) = 0 + 1 + 2 +...+ n

T(n) = n.(n + 1) / 2

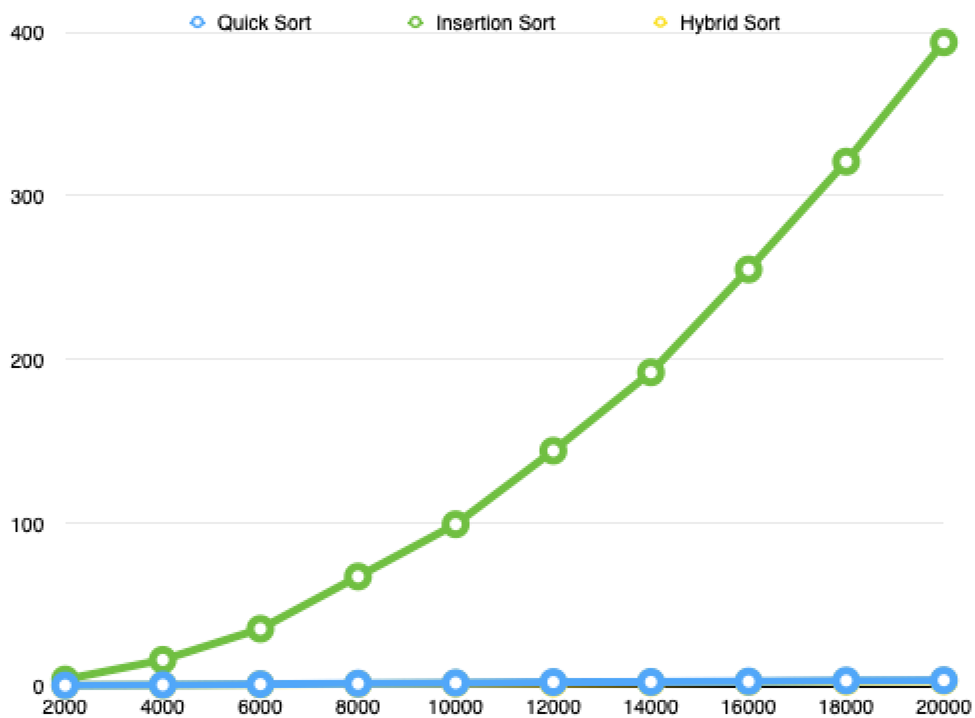
T(n) = O(n^2)

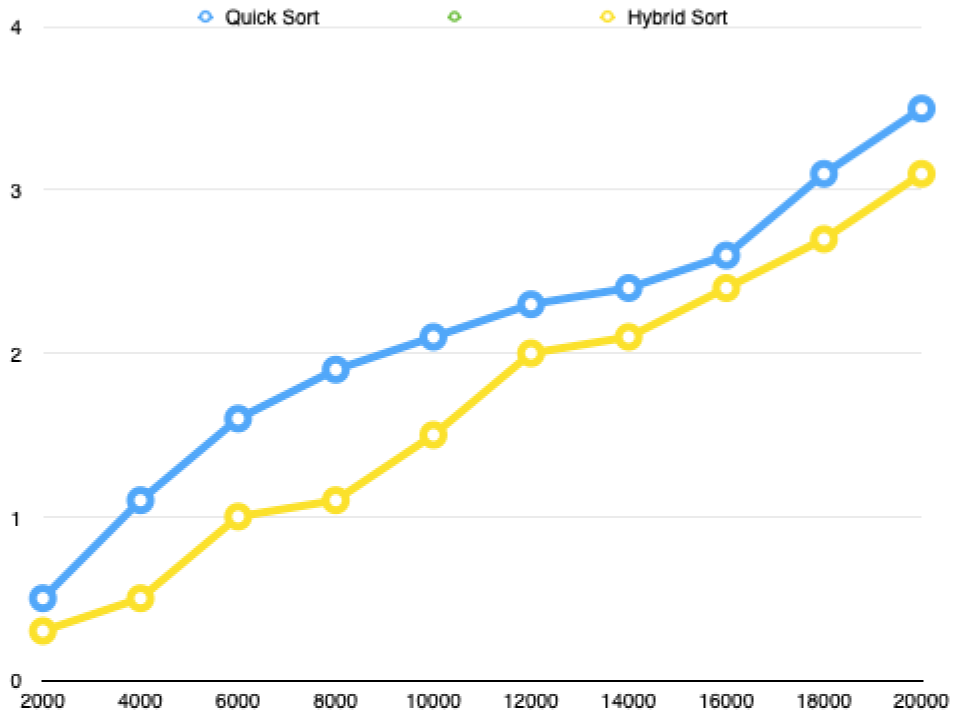
ANSWER: O(n^2)

Q2:

```
Quick Sort Test
[1, 17, 20, 43, 57, 58, 92, 93, 99, 100]
Insertion Sort Test
[1, 17, 20, 43, 57, 58, 92, 93, 99, 100]
Hybrid Sort Test
[1, 17, 20, 43, 57, 58, 92, 93, 99, 100]
------------------------------------------------------------
Part a - Time analysis of Quick Sort
Array size   Time Elapsed      compCount      moveCount
2000          0.560000 ms          24766          39000
4000          1.162000 ms          53255          87111
6000          1.644000 ms          88997         159798
8000          1.942000 ms         127129         193439
10000         2.152000 ms         156775         259710
12000         2.293000 ms         190924         291011
14000         2.400000 ms         246584         370554
16000         2.525000 ms         258244         444493
18000         3.145000 ms         293308         502709
20000         3.533000 ms         328431         520946
------------------------------------------------------------
Part b - Time analysis of Insertion Sort
Array size   Time Elapsed      compCount      moveCount
2000          4.546000 ms        1019579        1023577
4000         16.158000 ms        4046105        4054103
6000         35.264000 ms        8939931        8951929
8000         67.189000 ms       16180153       16196151
10000        99.851000 ms       24884171       24904169
12000       144.910000 ms       36280761       36304759
14000       192.225000 ms       48898620       48926618
16000       255.361000 ms       63937486       63969484
18000       321.279000 ms       81042551       81078549
20000       394.089000 ms       99315759       99355757
------------------------------------------------------------
Part c - Time analysis of Hybrid Sort
Array size   Time Elapsed      compCount      moveCount
2000          0.303000 ms          25230          32443
4000          0.551000 ms          54179          73465
6000          1.005000 ms          90146         139544
8000          1.195000 ms         129208         166638
10000         1.570000 ms         159288         226300
12000         2.012000 ms         194366         250466
14000         2.196000 ms         250121         323134
16000         2.517000 ms         262218         390499
18000         2.734000 ms         297806         441527
20000         3.119000 ms         332810         453171
------------------------------------------------------------
Program ended with exit code: 0
```

Q3:


Quick Sort
Insertion Sort
Hybrid Sort
400
300
200
100
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000



It can be seen from the graph that insertion sort is the slowest algorithm and hybrid sort is the fastest algorithm. In theory insertion sort is also slower than quick sort because on average case running time , insertion sort is O(n^2) whereas quick sort is O(nlogn). In the experiment random filled arrays are used. So results are similar to theoretical results. Also it can be seen from the graph that insertion sort acts as n^2 and quick sort and hybrid sort act as nlogn.

It can be seen from the graph that hybrid sort is slightly faster than quick sort. The reason behind this situation is that when very small sized arrays are sorted insertion sort is better than quick sort because it uses less moves and comparisons. In this experiment when the partition size is less than or equal to 10, hybrid sort uses insertion sort. That's why hybrid sort is slightly more efficient than quick sort.

Note: In the first graph, growths of quick sort and hybrid sort are not very visible because their growth rates are much smaller compared to insertion sort. Also their growth rates are very similar so in the first graph only quick sort is visible because they are merged together. So to be able to observe their running times and compare them better, a second graph is given.