/*
 * NAME: Kuangyi Yang
 * ID: A53083212
 * LOGIN: cs12fig
 */

########## Problem1 ##########
1)after Partition(A, 0, 7):

result: A = (1,3,5,7,2,4,8,9)
my work:
i = -1; p = 0; r = 7; pivot = 8:
j = 0: A[0] < 8, i = 0; A[0]<->A[0]
--> (1,3,5,7,9,2,4,8)
j = 1: A[1] < 8, i = 1; A[1]<->A[1]
--> (1,3,5,7,9,2,4,8)
j = 2: A[2] < 8, i = 2; A[2]<->A[2]
--> (1,3,5,7,9,2,4,8)
j = 3: A[3] < 8, i = 3; A[3]<->A[3]
--> (1,3,5,7,9,2,4,8)
j = 4: A[4] > 8, i = 3;
--> (1,3,5,7,9,2,4,8)
j = 5: A[5] < 8, i = 4; A[4]<->A[5]
--> (1,3,5,7,2,9,4,8)
j = 6: A[6] < 8, i = 5; A[5]<->A[6]
--> (1,3,5,7,2,4,9,8)
A[6]<->A[7]
--> (1,3,5,7,2,4,8,9)

First 2 iterations:
Partition(A, 0, 6)
result: A = (1,3,5,7,2,4,8,9)

Partition(A, 0, 5)
result: A = (1,3,2,4,5,7,8,9)
my work:
i = -1; p = 0; r = 5; pivot = 4:
j = 0; A[0] < 4, i = 0; A[0]<->A[0]
--> (1,3,5,7,2,4,8,9)
j = 1; A[1] < 4, i = 1; A[1]<->A[1]

--> (1,3,5,7,2,4,8,9)
j = 2; A[2] > 4, i = 1;
--> (1,3,5,7,2,4,8,9)
j = 3; A[3] > 4, i = 1;
--> (1,3,5,7,2,4,8,9)
j = 4; A[4] < 4, i = 2; A[2]<->A[4]
--> (1,3,2,7,5,4,8,9)
A[3]<->A[5]
--> (1,3,2,4,5,7,8,9)

########## Problem2 ##########
printInt(5):
5
4
3
2
1
printInt2(5):
1
2
3
4
5

########## Problem3 ##########

Solution1:
(2,6,4,9,1,3,5,7,8)

pivot = 1
(1l,6,4,9,2,3,5,7,8)

pivot = 2
(1,2l,4,9,6,3,5,7,8)

pivot = 3
(1,2,3l,9,6,4,5,7,8)

pivot = 4
(1,2,3,4l,6,9,5,7,8)

pivot = 5
(1,2,3,4,5l,9,6,7,8)

pivot = 6
(1,2,3,4,5,6l,9,7,8)

pivot = 7
(1,2,3,4,5,6,7l,9,8)

pivot = 8
(1,2,3,4,5,6,7,8,9)
======================

Solution2:
(2,4,6,8,9,5,3,7,1)

pivot = 9
(2,4,6,8,1,5,3,7,l9)

pivot = 8
(2,4,6,7,1,5,3,l8,9)

pivot = 7
(2,4,6,3,1,5,l7,8,9)

pivot = 6
(2,4,5,3,1,l6,7,8,9)

pivot = 5
(2,4,1,3,l5,6,7,8,9)

pivot = 4
(2,3,1,l4,5,6,7,8,9)

pivot = 3
(2,1,l3,4,5,6,7,8,9)

pivot = 2
(1,2,3,4,5,6,7,8,9)

########## Problem4 ##########
Find the mode is best implemented by first sorting the list of numbers
(a)
Algorithm:
```
for int i = 1; i < n; i++
        min = first element
        if (element(i) < min)
                min = element(i)
```

Complexity is O(n), so it's not a good idea to sort first
this is because, if we sort first, it takes at least O(n) to sort first, then takes O(1) to find the minimum, in other words, the complexity is still O(n)

(b)
Algorithm:
```
for int i = 1; i < n; i++
        max = first element
        if (element(i) > min)
                max = element(i)
```

Complexity is O(n), so it's not a good idea to sort first
this is because, if we sort first, it takes at least O(n) to sort first, then takes O(1) to find the maximum, in other words, the complexity is still O(n)

(c)
Algorithm:
```
sum = 0
for int i = 0; i < n; i++
        sum = sum + element(i)
mean = sum/n
```

Complexity is O(n), to find the mean, we only need to add all elements up, the orders does not matter, so sorting first does not improve anything

(d)

```
Algorithm:
k = A.length/2
find-kth(A, k)
  pivot = random element of A
  (L, R) = split(A, pivot)
  if k = |L|+1, return A[k]
  if k ≤ |L|  , find-kth(L, k)
  if k > |L|+1, find-kth(R, k-(|L|+1))
```

Complexity is O(n) in average but O(n^2) in the worst case. if we sort first, then it takes at least O(n) time to sort, then it takes O(1) time to find the element(A.length/2), it maybe not be a good idea to sort first

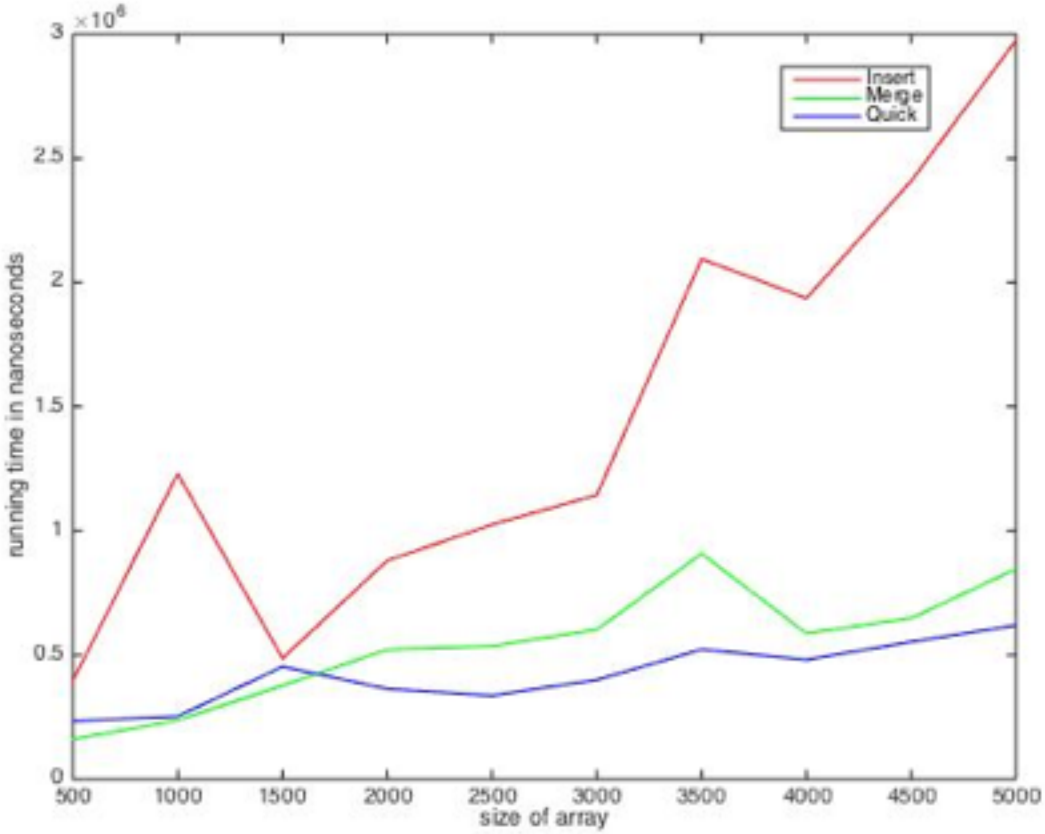
(e)
Algorithm:
sort first and then find longest subsequence, the complexity is O(nlogn)

```
step1: sort the sequence
step2: find the size of longest subsequence
count1 = 0
count2 = 0
max = 0
for i = 0; i < n; i++
        count1 = count1 + 1
        if elt(i) != elt(i+1)
                max = maximum(count1, count2)
                count2 = count1
                count1 = 0
return max
```

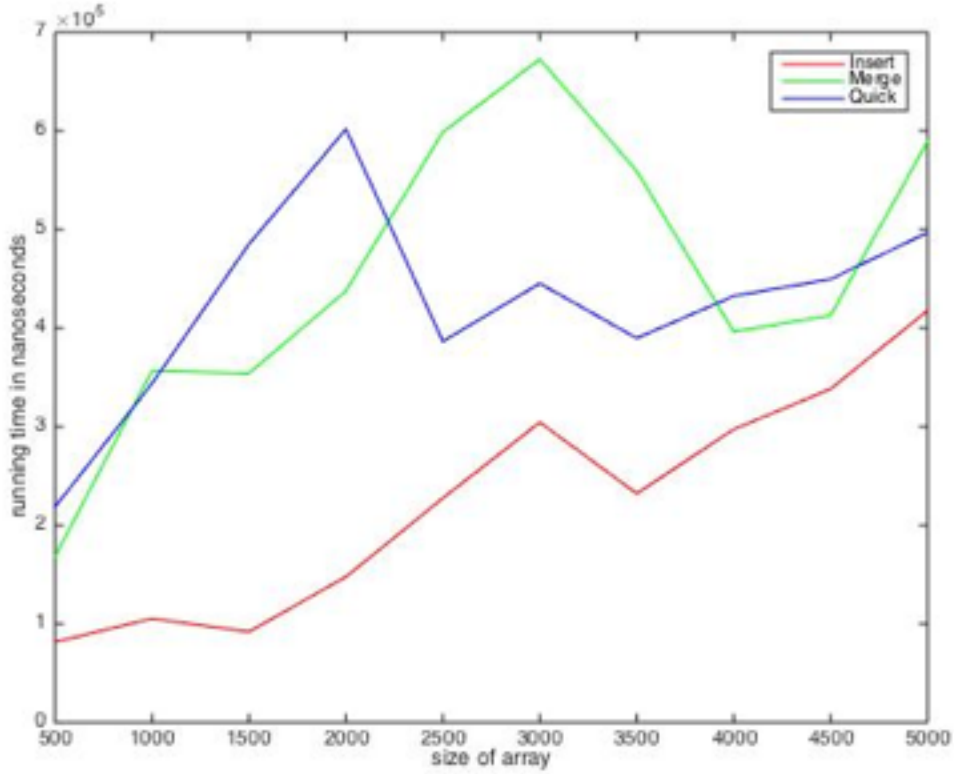
########## Problem5 ##########

```
T(n) = 4T(n/4) + c'n
    = 4(4T(n/16) + c'n/4) + c'n
    = 16T(n/16) + 2c'n
    = ...
    = 4^k*T(n/4^k) + kc'n
let k = log_4(n)
T(n) = n*T(1) + c'*n*log_4(n)
    = cn + c'nlog_4(n)
O(T(n)) = O(nlog_4(n))
```

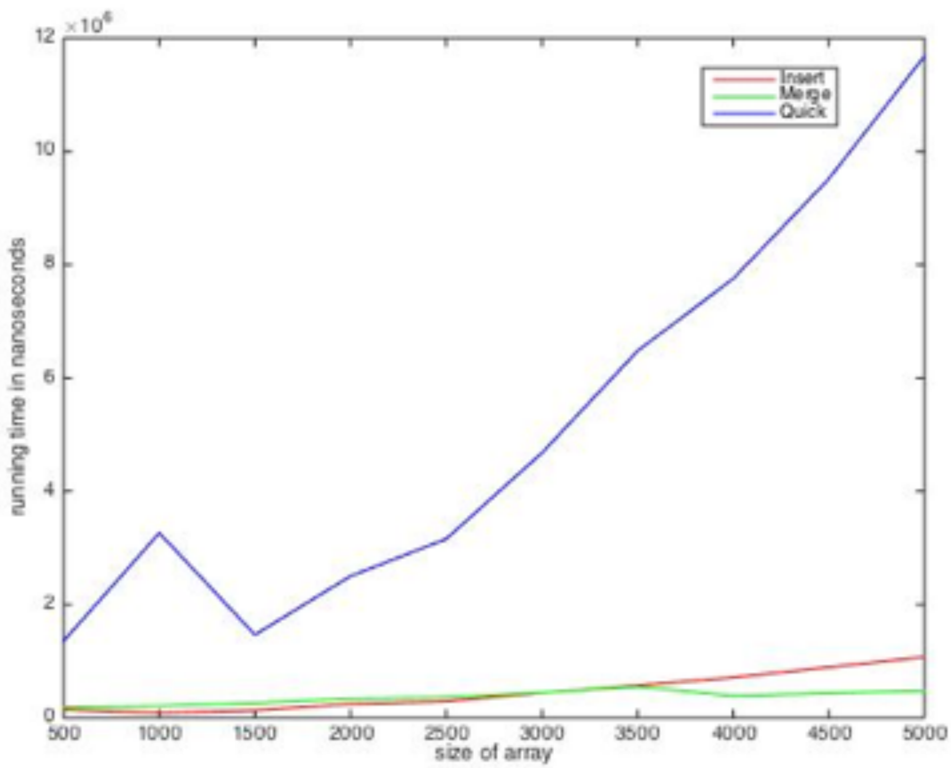
PART2:
(a) random:



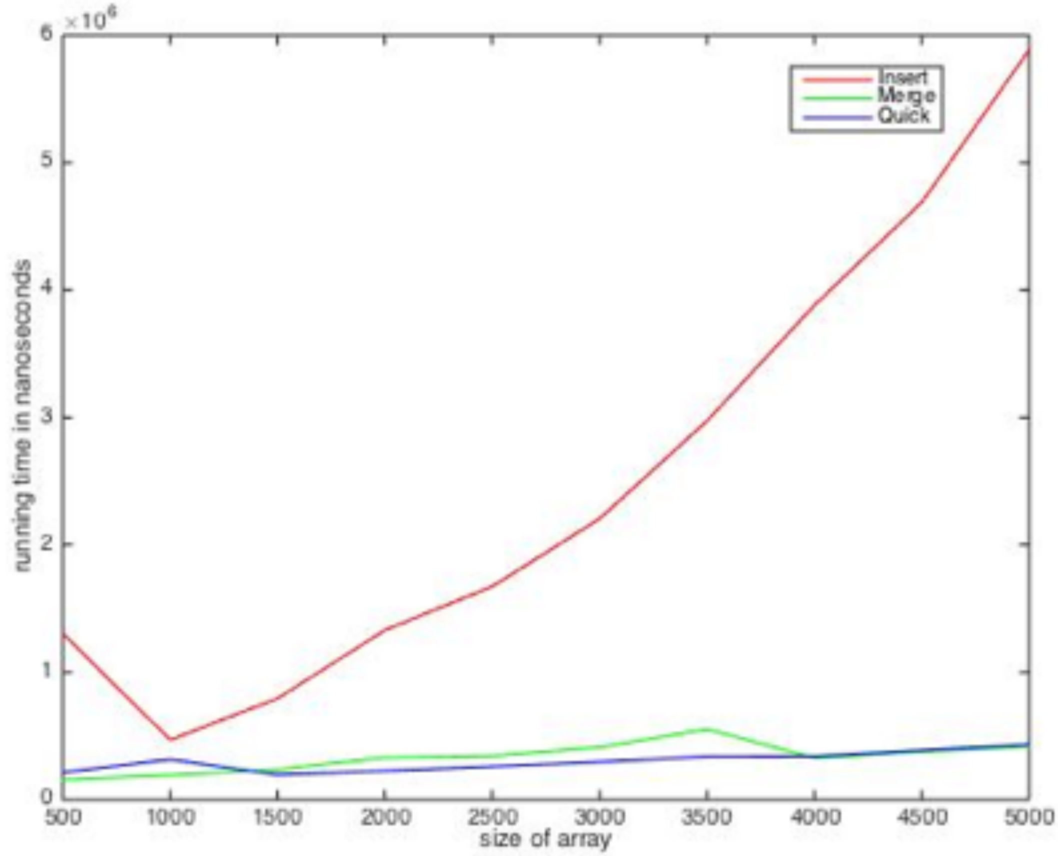
(b) Almost Sorted:



(c) Duplicated:



(d) Reverse:



findings:

(a) Random: merge sort and quick sort are quite similar(O(nlogn), and both are faster than insertion sort(O(n^2))
(b) Almost sorted: insertion sort is the fastest(O(n))
(c) Include many duplicates: merge sort is the fastest,quick sort is the slowest
(d) Reverse values: if pivot is the last element, quick sort is the lowest(O(n^2)); if pivot is random, quick sort and merge sort are almost the same(O(nlogn))

Conclusion:
The complexity of merge sort are almost the same in all situations O(nlogn)

In almost sorted situation, inserstion has the best performance O(n)

In reverse sorted situation, if pivot is the last element, quick sort is slowest O(n^2)

In random situation, quick sort and merge sort have better performance than insertion sort