ID: hh2537; Name: Haoze He

Q1:

When $\frac{n}{2^i} = k$, the quick sort stops. It takes$\mathrm{O}(\mathrm{n_i}) = O\left(nlog\left(\frac{n}{k}\right)\right)$. For each subarray with size k, use insertion sort, the time complexity is $\mathrm{O}(\mathrm{k}^2)$for each one. Sum them together, it is

$$\mathrm{O}(\mathrm{nk} + \mathrm{nlog}(\frac{\mathrm{n}}{\mathrm{k}}))$$

To improve the result, using master theorem, we choose smaller k to make sure $\mathrm{O}(\mathrm{nk})$ is not dominant comparing with $\mathrm{O}(\mathrm{nlog}(\frac{\mathrm{n}}{\mathrm{k}}))$. To make sure $\mathrm{O}(\mathrm{nk}) < O(nlogn)$, K should be $\mathrm{Olog}(\mathrm{n})$.

To proof that, we select c_1 for quick sort and c_2 for insertion sort.

$$C_2 nk + C_1 nlog\left(\frac{n}{k}\right) < C_1 nlogn$$

$$C_2 k < C_1 logk$$

Thus, k = $\mathrm{O}(\mathrm{logn})$

Q2.

a)

If all elements are the same, our recurrence becomes: $\mathrm{T}(\mathrm{n}) = T(n-1) + \theta(n)$, the time complexity becomes $\Theta(\mathrm{n}^2)$

b)

The pseudocode of the modified algorithm shows below:

```
Algorithm: PARTITION'(A, p, r):
x = A[p]
lowPointer = p
highPointer = p
for j = p + 1 to r:
    if A[j] < x:
        y = A[j]
        A[j] = A[highPointer + 1]
        A[highPointer + 1] = A[lowPointer]
        A[lowPointer] = y
        lowPointer = lowPointer + 1
        highPointer = highPointer + 1
    else if A[j] == x:
        swap(A[highPointer], A[j])
        highPointer = highPointer + 1
    return (lowPointer, highPointer)
```

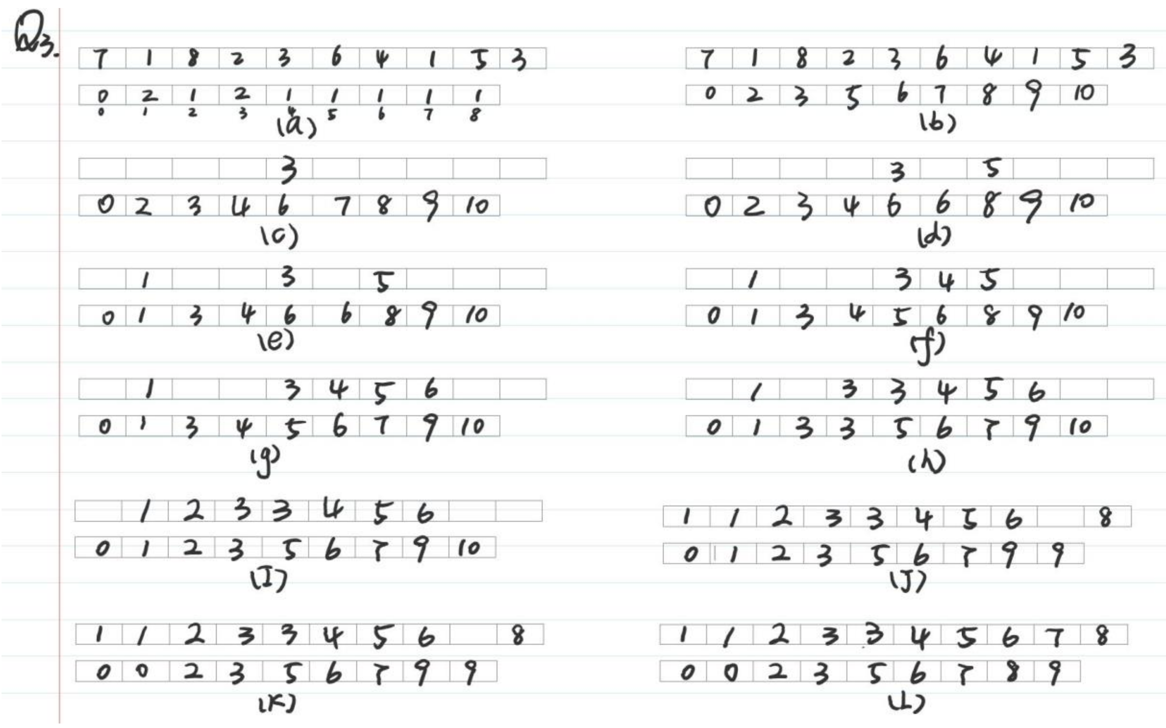
c)

```
Algorithm: QUICKSORT’(p,r):
if p < r:
    (lowPointer, highPointer) = RANDOMIZED-PARTITION'(A, p, r)
    QUICKSORT'(A, p, low - 1)
    QUICKSORT'(A, high + 1, r)
```

d)

When all input elements are distinct, he sub-problem of QUICKSORT’ is actually not larger than the sub-problem of QUICKSORT. It sometimes performs better since it’s more balanced.

Q3.



Q4.

There are two cases, n is odd or even.

Case1, n is odd:

$$\text{number of comparisons} = 1 + \frac{3(n-3)}{2} + 2 = \left(\left\lceil\frac{3n}{2}\right\rceil - \frac{1}{2}\right) - \frac{3}{2} = \left\lceil\frac{3}{2}\right\rceil - 2$$

Case2, n is even:

$$\text{number of comparisons} = 1 + \frac{3(n-3)}{2} = \frac{3n}{2} - 2 = \left\lceil\frac{3}{2}\right\rceil - 2$$

Hence, proof is done.

Q5.

a)

Sorting number running time = $O(n\log n)$, running time of listing i largest = $O(i)$.

Hence, total running time = $O(n\log n + i)$

b)

Running time of building a max-priority queue = $O(n)$, running time of each time call EXTRACT-MAX = $O(\log n)$.

Hence, total running time = $O(n + i \log n)$

c)

Running time of finding i^{th} largest number and partitioning is $O(n)$, running time of sorting the i^{th} largest number is $O(i\log i)$.

Hence, the total running time is $O(n + i \log i)$.