Quiz

1. non-in-place

1st call quicksort ({7, 2, 8, 5, 10, 6, 9, 4, 1, 3}, 0, 9)
2nd call: quicksort ({2, 5, 6, 4, 1, 3, 7, 8, 10, 9}, 0, 5)
3rd call: quicksort ({1, 2, 5, 6, 4, 3, 7, 8, 10, 9}, 0, 0)
4th call: quicksort ({1, 2, 5, 6, 4, 3, 7, 8, 10, 9}, 2, 5)
5th call: quicksort ({1, 2, 4, 3, 5, 6, 7, 8, 10, 9}, 2, 3)
6th call: quicksort ({1, 2, 3, 4, 5, 6, 7, 8, 10, 9}, 2, 2)
7th call: quicksort ({1, 2, 3, 4, 5, 6, 7, 8, 10, 9}, 4, 3)
8th call: quicksort ({1, 2, 3, 4, 5, 6, 7, 8, 10, 9}, 5, 5).
9th call: quicksort ({1, 2, 3, 4, 5, 6, 7, 8, 10, 9}, 7, 9}.
10th call: quicksort ({1, 2, 3, 4, 5, 6, 7, 8, 10, 9}, 7, 6}
11th call: quicksort ({1, 2, 3, 4, 5, 6, 7, 8, 10, 9}, 8, 9}
12th call: quicksort ({1, 2, 3, 4, 5, 6, 7, 8, 10, 9}, 8, 7}
13th call: quicksort ({1, 2, 3, 4, 5, 6, 7, 8, 9, 10}, 9, 9}

2. ① After pivot chosen, swap it to beginning of the array
② start counters i=1 and j=N-1
③ Increment i until find element A[i] > pivot
④ Decrement j until find element A[j] < pivot.
⑤ if i<j, swap A[i] with A[j], go back to ③
⑥ Otherwise, swap first element (pivot) with A[j]

① A $\underset{}{\overset{\text{pivot}}{\underline{6}}}$ 2 8 5 11 10 4 1 9 7 3
② i=1, j=10.
③ i=2 where A[2]=8 > 6. ④ j=10 where A[10]=3 < 6.
⑤ i<j, swap A[2], A[10]
A become 6 2 3 5 11 10 4 1 9 7 8 back to ③.
A[4]=11, A[7]=1. A becomes 6 2 3 5 1 10 4 11 9 7 8
A[5]=10, A[6]=4, A beomes 6 2 3 5 1 4 10 11 9 7 8
now j<i so we have A[5] and A[0] swapped.
A becomes 4 2 3 5 1 6 10 11 9 7 8.

3. 1) if it is (1,a), (2,b), (2,a), (3,a)

after selection sort, it will be (1,a) (2,a) (2,b) (3,a), so stability is violated.

2) (2,a) (2,b) (3,a), (1,a).

after selection sort, it will be (1,a) (2,b) (2,a) (3,a), so stability is violated.

3) Bubble sort is stable. In bubble sort, we compare every adjacent elements. so if the two adjacent elements have same key, we can have them not swapped. But in selection sort, swapping can be among elements not adjacent, resulting in unstable.

4.

	Worse-Case	Average-Case	In Place	Stable
Insertion sort	$O(N^2)$	$O(N^2)$	✓	✓
Selection sort	$O(N^2)$	$O(N^2)$	✓	X
Bubble sort	$O(N^2)$	$O(N^2)$	✓	✓
Merge sort	$O(N\log N)$	$O(N\log N)$	X	✓
Quick sort	$O(N^2)$	$O(N\log N)$	Weakly	X

5 ① Allocate array C[k+1]

② Allocate array D[k+1]

③ Scan A. For 1 to N, increment C[A[i]]

④ let D[1]=1 for i=2 to k, Set D[i]=D[i-1]+C[i-1]

⑤ for i=1 to N, put A[i] in new position C[A[i]], then increment C[A[i]].

Exercise.

2. $\log n! = \log(1\cdot 2\cdot 3\cdots n)$

$= \log 1 + \log 2 + \cdots + \log n$

$\geq \log(\frac{n}{2}) + \log(\frac{n}{2}+1) + \cdots + \log n$

$\geq \log(\frac{n}{2}) + \log(\frac{n}{2}) + \cdots + \log(\frac{n}{2})$

$= \log((\frac{n}{2})^{\frac{n}{2}}) = \frac{n}{2}\log(\frac{n}{2}) = \frac{n}{2}(\log n - \log 2)$

for $n \geq 4$, $\log n \geq 2\log 2$.

$\frac{1}{4} n\log n \geq \frac{n}{2}\log 2$.

Then $\log n! \geq \frac{n}{2}\log n - \frac{1}{4}n\log n = \frac{1}{4}n\log n$.

$\Rightarrow \log(n!) = \Omega(n\log n)$. proved.

3. we can use Dselection every time we determine the pivot.

```
void quicksort ( int *a, int left, int right ).{
    int pivotat = Dselect (int A[], int n).          O(n).
    if (left >= right) return;
    partition (a, left, right).;                     O(n)
    quick sort (a, left, pivotat-1);                 T(Leftsize)
    quicksort ( a, pivotat+1, right);                T(Rightsize).
}
```

since we use Dselect to find the median, we have

$T(N) = T((N-1)/2) + T((N-1)/2) + 2O(N)$

$\Rightarrow T(N) = O(N\log N)$.

4. we can firstly use "Choose Pivot (A, n)"

steps: 1. Break A into n/5 groups of size 5 each.

2. Sort each group.

3. Copy n/5 median into new array C.

4. Recursively compute median of C.

5. Return median of c as pivot.

after we get the median it returns, we can go over the array A to find whether pivot is larger than $\lceil n/2 \rceil$ times

Time complexity for Choos Pivot is $O(N)$, go over A is $O(N)$.

thus $O(N) + O(N) = O(N)$.

5. void merge (int arr[ ], int begin, int mid, int end)

```
{
    int begin2 = mid+1
    if (arr[mid] <= arr[begin2]) return;
    while (begin <= mid && begin2 <= end){
        if(arr[begin] <= arr[begin2]  start++;
        else {
            int value = arr[begin2];
            int index = begin2;
            while ( index != begin){
                arr[index] = arr [index -1];
                index --;
            }
            arr[begin] = value;
            begin++;
            mid++;
            begin2++;
        }
    }
}
void mergesort (int arr[ ], int l, int r){
    if (l<r){
        int m=l+(r-l)/2;
        mergesort (arr, l, m);
        mergesort (arr, m+1, r);
        merge (arr, l, m, r);
    }
}
```

time complexity is $O(N^2)$, quick sort is $O(n\log n)$, which is much faster.

6. 1) let the size be N.

for i = 0 to N-1 we have a loop inside for j=i to N-1, we calculate whether A[i] + A[j] = S. Obviously, time complexity is $O(N^2)$

2) for i=0 to N-1, we use binary search to find whether j exists to satisfy A[i]+A[j] = S. Since the Array is sorted, for each i, we need only log n for find j, then the whole algorithm time complexity will be O(nlogn).

7. 1) Insertion Sort. when n is small, average time $\frac{n(n-1)}{4}$ is better than bubble sort or O(nlogn) algorithm.

2). Insertion sort. Since it is almostly sorted, its time complexity for insertion sort is O(n).

3) Counting Sort. Since we have many integers of small range, it fits the requirement of Counting Sort whose time complexity is O(N+k)

4). Insertion sort, Bubble sort, Merge sort

Insertion: elements are visited in order and equal elements are inserted after its equals. (Bubble sort also visit in order).

Merge: we can let it maintain the relative order of equal keys. " merge (a, left, mid, right)

9. 1) Since $n \to \infty$, $\log n \to \infty$, we can not find c s.t.
$n \log n \leq c \cdot n$ for all n. Thus, it's incorrect.

2) $\frac{2^n}{n!} = \cancel{\frac{2}{1}} \cdot \cancel{\frac{2}{2}} \cdot \frac{2}{3} \cdot \cancel{\frac{2}{4}} \cdot \frac{2}{5} \cdots \cdot \frac{2}{5} = \frac{2}{3} \cdot \frac{2}{5} \cdot \frac{2}{6} \cdots \cdot \frac{2}{n} < 1$

then $\lim_{n \to \infty} \frac{2^n}{n!} < 1 \neq \infty$.

Thus, $2^n = O(n!)$ proved.

3). Correct. According to definition, $f_1(n) \geq c_1 g_1(n)$ for all $n > n_1$. $f_2(n) \geq c_2 g_2(n)$ for all $n > n_2$. suppose $n_0 = \max\{n_1, n_2\}$, then for $f_1(n) f_2(n)$, it is $\geq c_1 c_2 g_1(n) g_2(n)$

so, we have $c = c_1 c_2$. thus, such c, n exist to let all $f_1(n) f_2(n) \geq c\, g_1(n) g_2(n)$ for all $n > n_0 \Rightarrow f_1(n) f_2(n) = \Omega(g_1(n) g_2(n))$

4) correct. According to definition, $\overset{\text{for } n > n_1}{C_{11} g_1(n) \leq f_1(n) \leq C_{12} g_1(n)}$, $\overset{\text{for } n > n_2}{C_{21} g_2(n) \leq f_2(n) \leq C_{22} g_2(n)}$

let $n_0 = \max\{n_1, n_2\}$. then for $n > n_0$, we have.

$C_{11} g_1(n) + C_{21} g_2(n) \leq f_1(n) + f_2(n) \leq C_{12} g_1(n) + C_{22} g_2(n)$

assume for $n > n_0$, $g_1(n) \geq g_2(n)$

$\Rightarrow C_{11} g_1(n) \leq f_1(n) + f_2(n) \leq C_{12} g_1(n) + C_{22} g_1(n)$

let $C_{11} = C_1$, $C_{12} + C_{22} = C_2$. we have

$C_1 g_1(n) \leq f_1(n) + f_2(n) \leq C_2 g_1(n)$. for all $n > n_0$. proved.