CS 101 Fall 2021 - Quiz 3-A

10/18/2021 - 25 Minutes **Name:** **ID number:**

(8 Points) Problem 1: **True or False:** For each statement, write "T" if this statement is correct; write "F" otherwise. Please write your answers in the box below.

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
T	F	T	T	F	T	T	T

(1) Merge sort requires $O(n)$ space complexity.

(2) In quick sort (with n distinct elements and sort in ascending order, $n \geq 2$), if we randomly select the pivot, after the first partition operation, the smallest element of the array can be anywhere.

(3) Quick sort algorithm will have $O(n^2)$ time complexity in the worst case.

(4) Insertion sort never compares the same two elements twice.

(5) Quick sort runs in best case $\Theta(\log n)$ time for certain inputs.

(6) Randomly choosing pivots can be used to reduce the probability of quick sort taking the worst case running time.

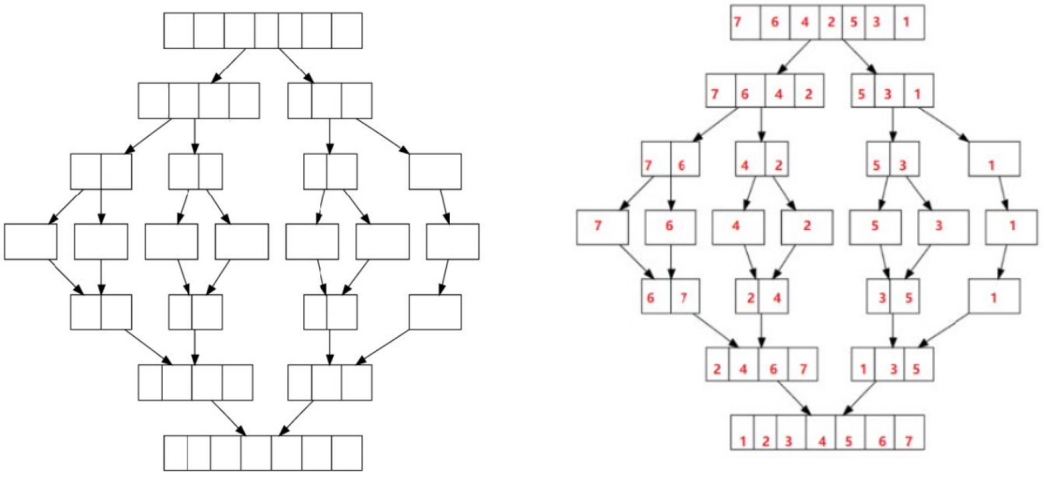
(7) Merge sort has a worst case runtime that is asymptotically better than quick sort's worst case runtime.

(8) Insertion sort (ascending) has the best time complexity on an already sorted list among all sorting methods.

(6 Points) Problem 2: Consider this array: 7, 6, 4, 2, 5, 3, 1.

(1) (4 pts) Use **mergesort** to sort this array in ascending order. Show your process in the following figure.

(2) (2 pts) How many inversions are there in the array? 18



(3 Points) Problem 3:

Tom wants to sort his favorite colors in ascending order using quicksort. The original array is:

$$red, cyan, yellow, gray, green, black, blue, white$$

After the first partitioning step, it becomes: ("red" is chosen as pivot)

$$white, cyan, yellow, gray, red, black, green, blue$$

Known that NO elements are equal, we can infer that: (Fill the blanks with ">", "<", or "?" if given informationis insufficient to judge)

(a) red____blue (b) yellow____gray (c) green____cyan

<, ?, >

CS 101 Fall 2021 - Quiz 3-A
10/18/2021 - 25 Minutes **Name:** **ID number:**

(8 Points) Problem 4: The following is a pseudo code

```
// ASSUMPTION: ALL ELEMENTS IN ARR[] ARE DISTINCT
int function(int arr[], int left, int right, int k)
{
    // If k is smaller than number of elements in array
    if (k > 0 && k <= right - left + 1) {
        // partition function moves all elements smaller than pivot to left of it
        // and greater elements to right in O(right - left + 1) time
        // and return the final position of the pivot
        int pos = partition(arr, left, right);
        // If position is same as k
        if (pos - left == k - 1)
            return arr[pos];
        // If position is bigger, recur for left subarray
        if (pos - left > k - 1)
            return funtion(arr, left, pos - 1, k);
        // Else recur for right subarray
        return function(arr, pos + 1, right, k - pos + left - 1);
    }
    // If k is bigger than number of elements in array, there is no correct result
    return INT_MAX;
}
```

- (3 Points) What is the output when $arr = [4, 3, 2, 5, 1], left = 0, right = 4, k = 4$?

5

- (3 Points) What does the function intend to do?

This function returns k'th smallest element in arr

- (2 Points) Analyze the **worst** time complexity of the function above. Please explain your answer.

The time complexity of the function is T(n). The pivot separates the array to two parts. Assume the pivot is ith smallest element in the array. Then there are i - 1 elements smaller than the pivot and n - i elements bigger than the pivot.
- if $i = 1, ..., k-1$, then the kth smallest element is in the back part, $T(n) = T(n-i) + O(n)$
- if $i = k$, then the kth smallest element is the pivot. $T(n) = O(n)$
- if $i = k+1, ..., n$, then the kth smallest element is in the front part, $T(n) = T(i-1) + O(n)$
In general, the worst case is $T(n) = T(n-1) + O(n)$.

So the average time complexity of the function is $O(n^2)$.