Student ID: ____________________ **Student Name:** ____________________

CS203 Data Structure and Algorithm Analysis **Quiz 1**

Note 1: Write all your solutions in the question paper directly. You can ask additional answer paper if necessary

Note 2: If a question asks you to design an algorithm, full marks will be given if your algorithm runs with optimal time complexity

Note 3: If a question asks you to design an algorithm, you should **first** describe your ideas in general words, **then** write the pseudocode, and **end** with time complexity analysis.

Problem 1 [20 points] Binary Search Algorithm

Let S1 be an unsorted array of n integers, and S2 is another sorted array of $\log_2 n$ integers (n is a power of 2). Describe an algorithm to output the number of pairs (x, y) satisfying x ∈ S1, y ∈ S2, and x > y. Your algorithm must terminate in $O(n \log\log n)$ time. For example, if S1 = {10, 7, 12, 18} and S2 = {15, 7}, then you should output 4 because 4 pairs satisfy the required conditions: (18,15), (10, 7), (12, 7), (18, 7).

Idea: (5 points)

For every element x ∈ S1, perform binary search on S2 to find the number t_x of elements in S2 that are smaller than x. Return $\sum_{x \in S1} t_X$.

Pseudocode: (10 marks)

```
Algorithm CountPairs(S1, S2)
    n ← len(S1)
    sum ← 0 // the total number of pairs
    for i ← 0 to n-1
        sum += findPairs(S1[i], S2)
    return sum
```

Algorithm findPairs(t, S2) (suppose S2 is in descending order)

```
left ← 0, right ← len(S2)
repeat
    mid ← (left+right)/2
    if (t <= S2[mid]) then
        right ← mid - 1
    else
        left ← mid + 1
until left > right
return right >0 ? right : 0
```

Time complexity analysis: (5 marks)

There are O(n) elements in S1, for each element, the binary search on S2 costs O(log log n) time, thus, the total cost is therefore O(n log log n).

Problem 2 [20 points] Iteration/Recursion method

Given an array **A** with **n** integers, please verify whether it is sorted in ascending order or not. Please implement your algorithm via iteration and recursion method, respectively.

(a) Iteration method **(10 points)**

```
IsSorted_Iteration(array A, integer n)
{
    for each i in 1 to n-1
    {
        if(a[i] > a[i+1])
            return FALSE
    }
    return TRUE
}
```

(b) Recursion method **(10 points)**

```
IsSorted_Recursion(array A, integer idx, integer n)
{
    If(n=1 || idx = n)
        return TRUE
    else
        return (A[idx] > A[idx+1])?FALSE:IsSorted_Recursion(A, idx+1, n);
}
```

Problem 3 [30 points] Algorithm Design

Let A[1…n] and B [1…n] be two arrays, each containing n integers in ascending order. Suppose all the 2n integers are distinct. Let k be an integer between 1 and 2n. Design an O(log n)-time algorithm to find the k-th smallest of the 2n elements.

Idea (10 points): we use recursion method to find the k-th smallest element in the two sized-n sorted arrays. The key idea is that we consider the median element of each array recursively. We omit base case, for recursive case, we take the median element u of A, namely, u=A[s] where $s = \lfloor n/2 \rfloor$, and the median element v of B , namely, v = B[t] where $t = \lfloor m/2 \rfloor$. Without loss of generality, assume that v <= u (otherwise, swap the roles of A and B). We distinguish two cases:
Case 1: s+t >= k: None of the elements in A[s+1…n] can possibly be the result. We recurse by searching for the k-th smallest element the s + m elements in A[1…s] and B[1…n].
Case 2: s + t < k: None of the elements in B[1…t] can possibly be the result. We recurse by searching for the (k-t)-th smallest element the n+m-t elements in A[1…n] and B[t+1…n].

Pseudocode (10 points):

```
Algorithm Findkth(array A, integer na, array B, integer nb, integer k)
1.  if (na < nb)  swap the roles of A and B
2.  if (m ==1) // Base case
3.      if (k == n+1)
4.          return max{A[n], B[1]}
5.      else if (k <=n)
6.          return (A[k] < B[1] ) ? A[k] : max{A[k-1], B[1] }
7.  else // recursive case
8.      s ← na/2, t ← nb/2
9.      if (A[s] < B[t])   swap the roles of A and B //
10.     if (s + t >= k)
11.         Findkth (A[1…s], s, B, nb, k)
12.     else
13.         Findkth(A, na, B[t+1…nb], nb-t, k-t)
```

Without loss of generality, we assume the size of array A is larger than or equal to the size of array B, and the median element value of array A is larger than or equal to the median element value of array B during the algorithm processing (see Line 1 / Line 12).

Time complexity analysis (10 points): In any case, we spend O(1) time and shrink one array by half for the recursion. Overall, the above shrinking can happen at most O(log n) + O(log n) times before reaching the base case. It thus follows that the entire algorithm finishes in O(log n + log n) time. Therefore, the problem can be settled in O(log n) time.

Problem 4 [30 points] Filling blank questions

(a) [5 points] The time complexity of the following function is **$O(\sqrt{n})$**.

```
int foo(int n){
        i = 0, s = 0;
        while(s < n){
            i ++;
            s += i; }
    }
```

(b) [5 points] Given a node P of a linked list L. P is neither the head nor the tail of L, which option can only delete the next node of P from L: **C**.

A. P = P -> next

B. P -> next = P

C. P - > next = P -> next -> next

D. P=P -> next -> next

(c) [5 points] Let f(n) be a function of positive integer n. We know:

f(1) = 1

f(n) = 2n + 4f(⌈n/4⌉):

then f(n) = **$O(n \log n)$**, recall that ⌈x⌉ is the ceiling operator that returns the smallest integer at least x.

(d) [5 points] Which of the following functions is $O(n\log\sqrt{n})$ ( )

A. $(1.03)^n$ B. $n \cdot (\log_2 n)^{1.0001}$ **C. $358 \cdot n\log_2 n$** D. $n^{1.2}/\log^5 n$

(e) [10 points] The time complexity of the following function is :

T(n) = **2T(n/2)+1** (recursion expression) = **O(n)** (Big-O notation).

```
int func(int n){
    if(n > 1){
        print("#")
        func(n/2)
        func(n/2)
    }
 }
```