CS3230: Design and Analysis of Algorithms
Semester 2, 2020-21, School of Computing, NUS

Midterm Test

Date: 6th March, 2021, Time Allowed: 2 hours

Instructions

- This paper consists of FIVE questions and comprises of TWELVE (12) printed pages, including this page.
- Answer ALL the questions.
- Write ALL your answers in this examination book.
- This is an OPEN BOOK examination.
- Write your Student Number (that starts with "A") on the top of every page.

Student Number:

Tutorial Group Number:

Question	Maximum	Score
Q1	5	
Q2	15	
Q3	25	
Q4	15	
Q5	20	
Total	**80**	

Question 1 [5 marks]: Rank the following functions in increasing order of growth; that is, the function $f(n)$ is before function $g(n)$ in your list, only if $f(n) = O(g(n))$.

- $n^{n+4} + n!$
- $n^{11\sqrt{n}}$
- $2^{4n \log n}$
- $4^{n^{1.1}}$
- $n^{11+\frac{\log n}{n}}$

To simplify notations, we write $f(n) < g(n)$ to mean $f(n) = o(g(n))$ and $f(n) \approx g(n)$ to mean $f(n) = \Theta(g(n))$. E.g., the four functions n^2, n, $2021n^2 + n$ and n^3 could be written in increasing order of growth as follows: $n < n^2 \approx (2021n^2 + n) < n^3$.

Note, all the logarithms are of base 2. No need to write down the proofs for this problem.

[1] In the following, we take $f < g < h$ to be the correct series

Question 2 [15 marks]: Solve the following recurrence relations.

(a) $T(n) = T(n/10) + \log n$.

(b) $T(n) = 27T(n/3) + n^3/\log^2 n$.

(c) $T(n) = 3T(n/5) + \log^2 n$.

(d) $T(n) = 2T(n/3) + n \log n$.

(e) $T(n) = T(n-2) + \log n$.

Among the above recurrences, for each recurrence solvable by master theorem, please indicate which case it belongs to (either case 1, 2, or 3) with proper reasoning and state the (time) complexity. For the recurrences that are not solvable by master theorem, solve them using any of the methods taught in the class. (Express your answers using $\Theta()$ notation.)

Question 3 [7+8+10=25 marks]: Consider a *circular array*, where the first and the last cells are neighbors. More specifically, in a circular array $A[1, \cdots, n]$ of size n, $A[1]$ and $A[n]$ are neighbors of each other. An element in the array is called a *peak* if it is greater than or equal to its neighbors. Now we would like to find a peak in a circular array. (Just to clarify, the array we consider in this question is one dimensional and unsorted.)

(a) Suppose we know the values of $A[1], A[i], A[i+1]$ and $A[n]$ for some $1 < i < n$. Consider the maximum value among these four cells. Then depending on the four possible maximum, in which of the following two circular sub-arrays $A[1, \cdots, i]$ and $A[i+1, \cdots, n]$, are you guaranteed to find a peak? (Note, you have to give answers for the four possible cases. Provide your answer with proper explanation.)

(b) Design a divide-and-conquer algorithm that finds a peak in a circular array of size n in time $O(\log n)$. (Provide the running time analysis of your algorithm.)

(c) Prove a lower bound of $\Omega(\log n)$ on the running time of any comparison-based algorithm that finds a peak in a circular array of size n. (Assume, each comparison among two numbers x, y has two possible outcomes: Either $x \leq y$ or $x > y$.)

Question 4 [15 marks]: Suppose you are given an array A of n pairs of positive integers (p_i, q_i), where $p_i < n$ and $q_i < n^3$ for all $i \in \{1, 2, \cdots, n\}$. Let us define *r-value* of a pair (p, q) as the real number $\sqrt{p} + q\sqrt{r}$. Unfortunately, you **cannot** (exactly) compute the r-value of an arbitrary pair (due to various reasons). Now, design an $O(n)$ time algorithm that sorts (in non-decreasing order) the pairs in A by their r-values for $r = n$.

Question 5 [20 marks]: Consider the insertion sort algorithm (perhaps, one of the first sorting algorithms we learned). The algorithm proceeds through the array and puts each element into place one at a time by comparing it to all the preceding items in the array. The details are not important for this question. However, for your reference, let us provide the pseudocode of the insertion sort algorithm. (Note, the pseudocode considers 0 as the starting index of an array. So the input array is $A[0, \cdots, n-1]$. However, for this question, it is not important.)

```
InsertionSort(Array A, integer n)
    for i=1 to (n-1) do
        item = A[i];
        int slot = i;
        while (slot > 0) and (A[slot] > item) do
             A[slot] = A[slot-1];
             slot = slot-1;
        A[slot] = item;
}
```

Figure 1: Pseudocode of the insertion sort algorithm

The key property that you need to consider is that the running time of InsertionSort depends on the number of *inversions* in the permutation being sorted. Given a sequence of distinct integers $\{b_1, b_2, \cdots, b_n\}$, an *inversion* is a pair (b_i, b_j) such that $i < j$ but $b_i > b_j$. E.g., the sequence $\{2, 3, 8, 5, 4\}$ contains only three inversions $(8, 5), (8, 4)$ and $(5, 4)$.

Without loss of generality, assume that the input to the InsertionSort is an (arbitrary) permutation of $\{1, 2, \cdots, n\}$. The following result relates the running time of InsertionSort to the number of inversions in the input sequence: If a permutation S of $\{1, 2, \cdots, n\}$ contains k inversions, then InsertionSort(S, n) runs in time $\Theta(n + k)$. (No need to prove this result. You can assume it.)

Now, analyze the average-case performance of InsertionSort. More specifically, let S be a permutation of $\{1, 2, \cdots, n\}$ chosen uniformly at random from the set of all permutations of $\{1, 2, \cdots, n\}$. Show that the expected running time of InsertionSort on S in $\Theta(n^2)$.

—End of Paper—