Indian Institute of Technology Kharagpur
CS21203 Algorithms-I, Autumn 2022

Class Test 1

31-August-2022 | 6.30pm - 7.30pm | Full Marks: 20

Name: ______________________ **Roll Number:** ______________________

Write your answers in the question paper itself. Be brief and precise. Answer all questions.

1. (5 marks) Solve the following recurrence relation:

$$T(n) = 297^2 \ . \ T(\sqrt[297]{n}) + 297 \ . \ (\log n)^2$$

Solution:

Let $n = 2^m$. Then $T(2^m) = 297^2 \ . \ T(2^{\frac{m}{297}}) + 297 \ . \ m^2$
Now let $S(m) = T(2^m)$. Then $S(m) = 297^2 \ . \ S(\frac{m}{297}) + 297 \ . \ m^2$
Comparing with the general form of recursion $T(n) = a \ T(\frac{n}{b}) + f(n)$, we get $a = 297^2$ and $b = 297$.
So $n^{\log_b a} = n^{\log_{297} 297^2} = n^2$. Hence $f(n)$ and $n^{\log_b a}$ are asymptotically same. So by case 2 of Master Theorem we obtain, $S(m) = \Theta(m^2 \log m)$. Changing back from $S(m)$ to $T(n)$ we obtain,

$$T(n) = T(2^m) = S(m) = \Theta(m^2 \log m) = \Theta((\log n)^2 \log \log n)$$

2. (5 marks) Work out the computational complexity of the following piece of code.

```
for ( i=1; i < n; i *= 2 ) {
  for ( j = n; j > 0; j /= 2 ) {
    for ( k = j; k < n; k += 2 ) {
      sum += (i + j * k );
    }
  }
}
```

Solution:

Running time of the inner, middle, and outer loop is proportional to n, $\log n$, and $\log n$, respectively. Thus the overall Big-Oh complexity is $O(n(\log n)^2)$.

More detailed analysis: Let $n = 2^k$. Then the outer loop is executed k times, the middle loop is executed $k+1$ times, and for each value $j = 2^k, 2^{k-1}, \cdots, 2, 1$, the inner loop has different execution times as follows:

j	Inner iterations
2^k	1
2^{k-1}	$(2^k - 2^{k-1})\frac{1}{2}$
2^{k-2}	$(2^k - 2^{k-2})\frac{1}{2}$
$\cdots$	$\cdots$
2^1	$(2^k - 2^1)\frac{1}{2}$
2^0	$(2^k - 2^0)\frac{1}{2}$

In total, the number of inner/middle steps is:

$$\begin{aligned}
1 + k.2^{k-1} - (1 + 2 + \cdots + 2^{k-1})\frac{1}{2} &= 1 + k.2^{k-1} - (2^k - 1)\frac{1}{2} \\
&= 1.5 + (k-1).2^{k-1} \equiv (\log_2 n - 1)\frac{n}{2} \\
&= O(n \log n)
\end{aligned}$$

Hence the total complexity is, $O(n(\log n)^2)$.

3. (5 marks) Algorithms A and B spend exactly $T_A(n) = 0.1\ n^2 \log_{10} n$ and $T_B(n) = 2.5\ n^2$ microseconds respectively, for a problem of size n. Choose the algorithm, which is better in the Big-Oh sense, and find out a problem size n_0 such that for any larger size $n > n_0$ the chosen algorithm outperforms the other. If your problems are of the size $n \leq 10^9$, which algorithm will you recommend to use?

Solution:

In the Big-Oh sense, the algorithm B is better. It outperforms the algorithm A when $T_B(n) \leq T_A(n)$, that is, when $2.5\ n^2 \leq 0.1\ n^2 log_{10} n$. This inequality reduces to $log_{10} n \geq 25$, or $n \geq n_0 = 10^{25}$. If $n \leq 10^9$, the algorithm of choice is A.

4. (5 marks) Let $A[1 \cdots n]$ be an array of n distinct numbers. A is unimodal, i.e., for some $i, 1 \leq i \leq n$, $A[1] < \cdots < A[i]$ and $A[i] > A[i+1] > \cdots > A[n]$. Design and analyze an efficient algorithm to find i. It will be okay to write the idea in plain text instead of writing in formal algorithm format.

Solution:

Probe at the middle of the array, i.e. $A[\frac{n}{2}]$ and compare it with $A[\frac{n}{2}-1]$ and $A[\frac{n}{2}+1]$. Now consider the following cases:

<u>Case-1</u>: $(A[\frac{n}{2}-1] < A[\frac{n}{2}] < A[\frac{n}{2}+1])$: The mode will be between $A[\frac{n}{2}+1]$ and $A[n]$ and we can recurse on the part between $\frac{n}{2}+1$ and n.

<u>Case-2</u>: $(A[\frac{n}{2}-1] > A[\frac{n}{2}] > A[\frac{n}{2}+1])$: The mode will be between $A[1]$ and $A[\frac{n}{2}-1]$ and we can recurse on the part between 1 and $\frac{n}{2}-1$.
<u>Case-3</u>: (otherwise:) So, here $A[\frac{n}{2}]$ is greater than both $A[\frac{n}{2}-1]$ and $A[\frac{n}{2}+1]$, and hence $A[\frac{n}{2}]$ is the peak element.

In all of the above (exclusive as well as exhaustive cases), we perform at most three comparisons and reduce the problem to a size of at most half of the original problem. Thus, we have an $O(\log n)$ time algorithm.