CS21003 Algorithms–I, Spring 2018–2019

Class Test 1

12–February–2019 F116/F142, 07:00pm–08:00pm Maximum marks: 20

Roll no: ____________________ **Name:** ____

[*Write your answers in the question paper itself. Be brief and precise. Answer all questions. If you use any algorithm/result/formula covered in the class, just mention it, do not elaborate.*]

1. Consider the following function.

```
unsigned int f ( unsigned int n )
{
   unsigned int i = 1, z = 0;

   while (i <= sqrt(n)) {
      z = z + i;
      i = 2 * i;
   }
   return z;
}
```

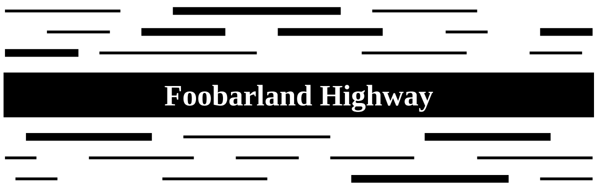
(a) Express the return value $f(n)$ as the big-theta Θ of a simple function of the input n. Prove the bound. **(4)**

Solution Let $2^t \leqslant \sqrt{n} < 2^{t+1}$. Then, the variable z accumulates the sum $z = 1 + 2 + 2^2 + \cdots + 2^t = 2^{t+1} - 1$. Thus the return value satisfies $\sqrt{n} - 1 < z \leqslant 2\sqrt{n} - 1$, which is $\Theta(\sqrt{n})$.

(b) Analyze the time complexity of the function (in the Θ notation) in terms of its input n. **(4)**

Solution The loop runs for $t + 1$ iterations with each iteration taking $O(1)$ time. We have $(\frac{1}{2}\log_2 n) - 1 < t \leqslant \frac{1}{2}\log_2 n$. Therefore the running time is $\Theta(\log n)$.

2. A set of n mobile-phone towers serves the entire stretch of a long straight highway in Foobarland. The Foobarland president finds that there are too many towers. He plans to remove as many towers as possible without sacrificing the requirement that the entire highway must still be covered. Here is an example of this situation. The thick lines stand for a set of chosen towers that can serve the total stretch of the highway.

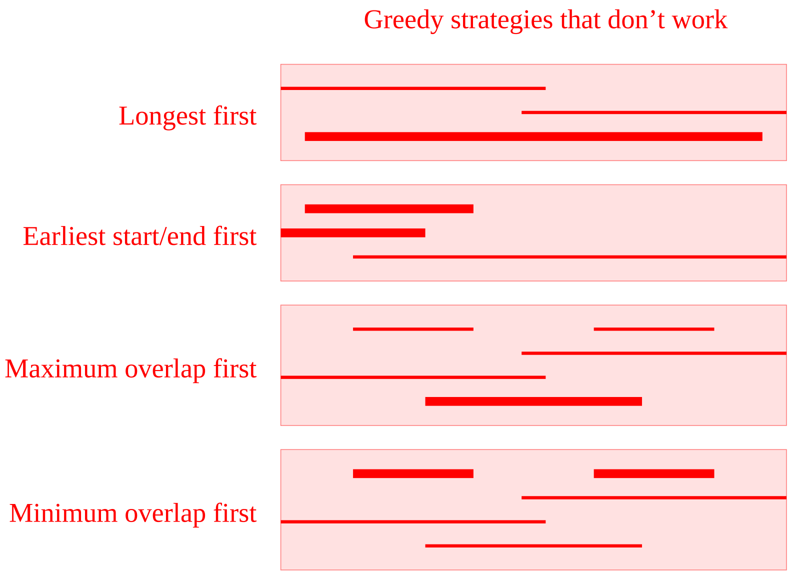


The range of a tower is supplied as an interval $[a_i, b_i]$ with $a_i < b_i$. The highway stretches from $L = \min_i(a_i)$ to $R = \max_i(b_i)$. Your task is to find a minimum set of intervals whose union is the entire interval $[L, R]$.

(a) Propose an efficient greedy algorithm to solve this problem. **(5)**

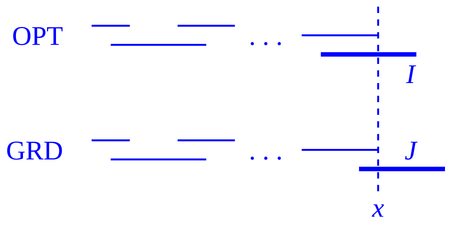
Solution Let us call an interval $[a_i, b_i]$ *active* at a point $x \in [L, R]$ if $a_i \leqslant x < b_i$. With this notation, the algorithm can be stated as follows.

```
1. Initialize $x = L$, and $S = \emptyset$.
2. While $x < R$, repeat:
    (a) Find all the intervals active at $x$.
    (b) Choose an active interval $I_j = [a_j, b_j]$ with the largest right endpoint $b_j$.
    (c) Add $I_j$ to $S$, and set $x = b_j$.
3. Return $S$.
```



(b) Prove that your algorithm correctly computes a minimum set of towers. **(5)**

Solution Let GRD be a solution produced by the greedy algorithm, and OPT an optimal solution. We sort both GRD and OPT individually in the ascending order of the left endpoints of the intervals. Assume that OPT $\neq$ GRD. Suppose that GRD and OPT match in the first k intervals (for some $k \geqslant 0$), whereas the $(k+1)$-st interval in OPT is I, and that in GRD is J with $J \neq I$. See the following figure which also shows the value of x at the time when the greedy algorithm chooses the interval J.



Let $I = [a,b]$. If $a > x$, then the region (x,a) is not covered by OPT (recall that the intervals are sorted with respect to their left endpoints, so no other following interval in OPT can cover (x,a)). If $b \leqslant x$, then the interval I can be thrown from OPT, and this reduced OPT continues to cover the entire highway $[L,R]$. Thus, we must have $a \leqslant x < b$, that is, I is an active interval at x. GRD chooses $J = [a',b']$ because $b' \geqslant b$. The region $[L,x]$ is covered by the first k intervals in both OPT and GRD. Now, if we replace I by J in OPT, the modified collection still covers $[L,R]$.

Proceeding in this way, we can convert OPT to GRD without ever increasing the number of intervals. We can therefore conclude that GRD is also *an* optimal solution.

(c) Deduce a good bound on the running time of your algorithm in the O notation. **(2)**

Solution Step 1 takes $O(1)$ time. Steps 2(a) and 2(b) require at most $O(n)$ time, whereas Step 2(c) takes $O(1)$ time. Moreover, the loop of Step 2 can run for at most n iterations. Step 3 takes $O(1)$ time (or $O(n)$ time depending on the implementation). The overall running time is thus $O(n^2)$.

Note: With careful data structuring, this problem seems to be solvable in $O(n \log n)$ time. But it is fine if the students come up with an $O(n^2)$-time algorithm.

FOR ROUGH WORK