ShanghaiTech University

CS240 Algorithm Design and Analysis

Fall 2021

Problem Set 1

Due: 23:59, Sept. 30, 2021

1. Submit your solutions to Gradescope (www.gradescope.com).
2. In "Account Settings" of Gradescope, set your FULL NAME to your Chinese name and enter your STUDENT ID correctly.
3. If you want to submit a handwritten version, scan it clearly. Camscanner is recommended.
4. When submitting your homework, match each of your solution to the corresponding problem number.

Problem 1:

Order the following functions so that $f_i \in O(f_j)$ when $i \leq j$.

1. $f_1(n) = 1.1^n$
2. $f_2(n) = \log_2(n) * \log_5(n^9)$
3. $f_3(n) = 9999999999$
4. $f_4(n) = n * 83970^{\log_2(n)}$
5. $f_5(n) = \sqrt{n} + n$
6. $f_6(n) = 666^{\sqrt{n}}$
7. $f_1(n) = n^{\frac{2}{3}}$

Solution:

3 2 7 5 4 6 1

Problem 2:

1. Prove that if $f(n) \leq O(g(n))$ and $g(n) \leq O(h(n))$, then, $f(n) \leq O(h(n))$
2. Give the time complexity of the following code in $\Theta(\cdot)$ notation.

Algorithm 1 While Loop

```
$i \leftarrow n$
while $i > 1$ do
    $j = i$
    while $j < n$ do
        $k \leftarrow 0$
        while $k < n$ do
            $k = k + 2$
        end while
        $j \leftarrow j * 2$
    end while
    $i \leftarrow i/2$
end while
```

Solution:

1. By definition of big-O, there exists constants C_1 and C_2 such that for all n we have
$$f(n) \leq C_1 g(n)$$
and
$$g(n) \leq C_2 h(n)$$
Multiplying the second one by C_1 then gives:
$$C_1 g(n) \leq C_1 \cdot C_2 h(n)$$
which together with the first condition gives:
$$f(n) \leq C_1 g(n) \leq C_1 \cdot C_2 h(n)$$

2. The innermost loop takes $\Theta(n)$ whenever it is called. The outer most loop takes $\Theta(\log n)$ steps, and between steps $\log n/2$ and $\log n$ (a total of $\Theta(\log n)$ of steps), we have $i < n^{1/2}$ In each such step, j gets doubled $\Theta(\log n)$ times before it reaches n. So we get a total of $\Theta\left(\log^2 n\right)$ iterations, each costing $\Theta(n)$, for a total of $\Theta\left(n \log^2 n\right)$.

Problem 3:

There is a straight country road with houses along it. The goverment wants to place some phone base stations at certain points along the road, so that every house is within four miles of one of the base stations. Give an efficient algorithm to achieve this goal with as few base stations as possible. Prove your answer is correct.

Solution:

Think of the houses as points $H = \{H_1, ..., H_n\}$on a line in that order from left to right, and the base stations as some other k points $P_1, ..., P_k$ on the same line. A solution is feasible iff every point H_i is within 4 miles of some point P_j.Our greedy strategy is to put P_1 exactly four miles to the right of H_1, remove all houses covered by P_1, i.e. with in 4 miles of P_1, and then recursively solve the sub-problem containing the rest of the houses. Let's refer to the rest of the houses as H. Let $P = (P_1, ..., P_k)$ be the solution returned by our algorithm. We will show by induction on n that our solution is optimal. The base case is trivial. Suppose our algorithm is correct for any set of at most $n-1$ houses. Consider n houses $H_1, ..., H_n$.

Claim 1: there exists an optimal solution O which puts a base station at P_1.

Let $O = \{P_1, ...P_m\}$ be any optimal solution. Clearly P_1 cannot be to the right of P_1, or else H_1 is not covered. Thus, the set of houses P_1 covers (all houses with in 8 miles to the right of H_1) contains the set of houses P_1 covers. Consequently, $O = \{P_1, P_2, ..., P_m\}$ is a feasible solution which is optimal since it has as many base stations as O.

Claim 2: let $O = \{P_1, P_2, ...P_m\}$ be an optimal solution which contains P_1, then $O = \{P_2, ..., P_m\}$ is optimal for the sub-problem H containing houses that P_1 does not cover.

If there is a better solution to the sub-problem H, then that solution along with P_1 would be feasible (all of H are covered) and would be better than O, violating the optimality of O.

Conclusion. The cost of our solution is $1 + (k-1) = 1 + OPT(H) = 1 + cost(O) = cost(O) = OPT(H)$. The first equality follows from the induction hypothesis that our algorithm is optimal for H. The second equality follows from Claim 2. The third follows from the definition of O in Claim 2. If the H_i are already sorted, the algorithm runs in time $O(n)$. If not, $O(nlgn)$is needed.

Problem 4:

There are some programs that need to be run on a computer. Each program has a designated start time and finish time and cannot be interrupted once it starts. Programs can run in parallel even if their running time overlaps. You have a 'check' program which, if invoked at a specific time point, can get information of all the programs running on the computer at that time point. The running time of the 'check' program is negligible. Design an efficient algorithm to decide the time points at which the 'check' program is invoked, so that the 'check' program is invoked for as few times as possible and is invoked at least once during the execution of every program. Argue that your algorithm is correct.

Solution:

Sort all the programs by their start times. If the start time of $A \geq B$, then B is behind A in the sorted sequence. In this sequence, we put the "check" time at the first program's start time, then delete the programs which are running at this "check" time. After this step, the remaining programs are all finished before "check" time, and the new first program in the sequence is started latest among remaining programs. Repeat previous step until no program remained. Sorting needs O(nlogn) and recursion the main need O(nlogn), so the time complexity is O(nlogn)

Prove: Assume there is a truly method to start "check" program at a1, a2, ... ,an. And our greedy algorithm gives b1, b2, ..., bn . Assume a1 = b1, a2 = b2, ..., ai-1 = bi-1 and a1 $\neq$ bi. As the theory of greedy algorithm, bi ¡ ai.
Due to other programs start before bi, so we can use bi to instead of ai, the number of programs keep starting at bi can not less than at ai(exclusive of the programs which have been checked before). Thus, greedy algorithm is true, time complexity is O(nlogn)

Problem 5:

Given a circularly sorted integer array consisting n numbers, find the total number of times the array is rotated. Assume there are no duplicates in the array, and the rotation is in the anti-clockwise direction, after n times rotation, the array can be sorted from smallest to largest.Please design an algorithm done in O(log$n$).

For example, array = [3,4,5,0,1,2], then number of rotation is 3.

Solution:

we can see that the total number of rotations is equal to the total number of elements before the minimum element.

We have already reduced the problem to find out the first element of the sorted sequence. The first element (Pivot) has one special property (let's call it the pivot's property) – both the next and previous element of the pivot element are greater than it. No other array element will have this property except the pivot element. Since the array is circularly sorted,If the pivot is the last element, then the first element will be considered its next element.If the pivot is the first element, then the last element will be considered its previous element.

We know that the middle element always divides the array into two subarrays, and the pivot element can lie only in one of these halves. It is worth noticing that at least one of these subarrays will always be sorted. If middle element happens to be the point of rotation (minimum element), then both left and right subarrays are sorted. Still, in any case, one half (subarray) must be sorted, and we will use this property to discard the left half or the right half at each iteration of the binary search.

Problem 5:

You have been hired to plan the flights and you are going to provide service to n cities. This airline will only fly you East. You need to enable all your passengers to travel from any city to any other city (to the East) with a single flight. Brute force requires Ω(n2) different routes. Devise a set of routes which requires no passenger have more than a single connection (i.e. must take at most two flights), and requires no more than O(nlogn) routes. Prove that your set of routes satisfies these requirements.

Divide Step. Describe the divide step of your algorithm here, making sure to mention what subproblems are produced. Include the total number of routes created during the divide step.

Combine Step. Describe the combine step of your algorithm here. Include the total number of routes created during this combine step

Algorithm. Describe your whole algorithm here. You do not need to repeat the procedures for divide and combine, you're welcome to simply reference them here.

Correctness. Prove the correctness of your algorithm. Namely, show that the selected routes have the property that one can travel from any city to an eastward city with at most a single connection.

Number of Routes. Express the number of routes your algorithm selects as a recurrence relation. Describe, using the master theorem, how you know that this recurrence is O(nlogn).

Solution:

We can arrange cities from west to east. Chose the city in the middle as a transfer station to connect west half and east half cities. Do the same thing in west half and east half recursively. There are O(logn) levels of transfer stations and each level will contain O(n) routes.

Problem 6:

Given a n by n wall where n is of form 2^k where $k \geq 1$. The wall has one socket (of size 1 x 1). Bespread the wall (except the socket) using L shaped planks consisting of three 1 x 1 square. Planks are not allowed to cross. You just need to describe and analyze the algorithm.

Solution:

Each step we tile one plank in the center of the considering region: we divide the considering region into four subregions and the empty part of the L shaped plank will be located the subregion which contains the socket.