ShanghaiTech University

CS240 Algorithm Design and Analysis

Spring 2020

Problem Set 1

Due: 23:59, Mar. 23, 2020

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

Problem 1:

Take the following list of functions and arrange them in ascending order of growth rate. That is, if function $g(n)$ immediately follows function $f(n)$ in your list, then it should be the case that $f(n)$ is $O(g(n))$.

$$f_1(n) = n^{\log_2 n} \tag{1}$$

$$f_2(n) = n^2 \log_2 n \tag{2}$$

$$f_3(n) = 2^{2^n} \tag{3}$$

$$f_4(n) = 10^{10^{10^{10}}} \tag{4}$$

$$f_5(n) = 2^{\sqrt{n}} \tag{5}$$

$$f_6(n) = 2^{\frac{5}{2}\log_2 n} \tag{6}$$

Problem 2:

We build up a tower with a pile of stones. Each layer of the tower is exactly one piece of stone. We index the layers from top to bottom, so layer 1 is the top layer. The stone weight of the i-th layer is s_i. The tower will fall if there exists a layer i s.t. $\sum_{j=1}^{i-1} s_j > s_i$. For example, if the tower is built with 5 stones and the stone weight from top to bottom is $2, 3, 5, 9, 100$, then the tower would fall because $9 < 5 + 3 + 2 = 10$.

You are the designer of the tower and you have n stones. Each stone has different weight and the same height. Can you build the tower as high as possible? Give an efficient algorithm and prove that the algorithm produces the highest tower. Show the time complexity of your algorithm.

Problem 3:

Assume you are a grandparent and going to give your grandchildren some pieces of cake. However you cannot satisfy a child unless the size of cake piece he receives is no less than his expected size. Different children may have different expected sizes. Meanwhile, you can't give each child more than one piece. For example, if three children's expected sizes are $\{1, 3, 4\}$ and the sizes of pieces of cake are $\{1, 2\}$, then you could only satisfy the first child. Given the expected sizes of the children and the sizes of cake pieces that you have, how can you make the most children satisfied? Show that your algorithm is correct.

Problem 4:

There are n students and each student i has 2 scores x_i, y_i. Students i, j are friends if and only if $x_i < x_j$ and $y_i > y_j$. How many pairs of friends are there? Design an efficient algorithm. For comparison, our algorithm runs in $\mathcal{O}(n \log n)$ time. (Hint: extend merge sort.)

Problem 5:

Given n numbers $a_1, ..., a_n$ s.t. $0 \leq a_i < 2^K$ for all i. There must exist a number X s.t. $\max_{1 \leq i \leq n} (a_i \oplus X)$ is minimum, where $\oplus$ is bitwise XOR operation.

You can find the minimum value of $\max_{1 \leq i \leq n} (a_i \oplus X)$ by Algorithm 1 below where & is bitwise AND operation.

(a) Show that this algorithm can find minimum value of $\max_{1 \leq i \leq n} (a_i \oplus X)$.

(b) What is the time complex of this algorithm? Prove your answer.

Problem 6:

You are given three sequences A, B and C. The length of the three sequences is m, n and m+n respectively. In other words, the length of C is the sum of the length of A and B. Design an algorithm to check if A and B can be merged into C such that the order of all the letters in A and B is preserved.

Example 1: A=aabb, B=cba, C=acabbab, then your algorithm should return true.

Example 2: A=aabb, B=cba, C=aaabbbc, then your algorithm should return false.

Problem 7:

Given a set of numbers, $a_0, a_1, ..., a_{n-1}$, answer q questions. Each question contains two parameters l, r. You need to answer $\max_{l<i<r} a_i$ for each question. Give an algorithm to answer all the questions in $O(n \log n + q)$ time.

HINT: You need to answer every question in constant time, which means you should pre-compute something in $O(n \log n)$ time to make searching the maximum in a range easy.

Algorithm 1 Minimum Value

```
1: procedure F(INT K, ARRAY A)
2:     t ← 2^{K−1}
3:     B ← {}
4:     C ← {}
5:
6:     for all i ∈ A do
7:         if i & t > 0 then add i to B
8:         else add i to C
9:         end if
10:    end for
11:
12:    if K == 1 then
13:        if B or C is empty then return 0
14:        else return 1
15:        end if
16:    end if
17:
18:    if B is not empty then AnsB ← F(K − 1, B)
19:    else AnsB ← ∞
20:    end if
21:    if C is not empty then AnsC ← F(K − 1, C)
22:    else AnsC ← ∞
23:    end if
24:
25:    if B or C is empty then return min{AnsB, AnsC}
26:    else return t + min{AnsB, AnsC}
27:    end if
28: end procedure
```