ShanghaiTech University

CS240 Algorithm Design and Analysis

Spring 2021

Problem Set 1

Due: 23:59, Mar. 16, 2021

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 $O(g(n))$.

According to Table 2.1 in the slide 02, when n is going to large.

$O(n) < O(n \cdot \log_2 n) < O(n^2) < O(n^3) < O(1.5^n) < O(2^n) < O(n!)$

For $f_7(n)$ and $f_8(n)$, according to the difinition of upper bounds: For all $n \geq 1$, $C = 10^{10^{10}}$, we have $10^{10^{10}} \leq C \cdot n^{\frac{1}{5}}$, so $10^{10^{10}} = O(2^{\frac{1}{5}\log_2 n})$.

For $f_5(n)$ and $f_8(n)$, $\log f_5(n) = 3^n$.

$\log f_8(n) = 2^n \cdot \log_2 3$, when $n \geq 1$, $C = \log_2 3$.

we have $2^n \log_2 3 \leq C \cdot 3^n$, so we have

$\log f_8(n) \leq C \cdot \log f_5(n)$, thus $f_8(n) \leq C\, f_5(n)$, $f_8(n) = O(f_5(n))$

$$f_1(n) = \log_{\sqrt{2}} 2^n \quad \backsim O(2^n) \tag{1}$$
$$f_2(n) = n^{\log_2 n} \quad \backsim O(2^n) \tag{2}$$
$$f_3(n) = 2^{\sqrt{n}} \quad \backsim O(2^{n^{\frac{1}{2}}}) \tag{3}$$
$$f_4(n) = n^2 \log_2 n \quad \backsim O(n^2 \cdot \log_2 n) \tag{4}$$
$$f_5(n) = 2^{3^n} \tag{5}$$
$$f_6(n) = 2^{\frac{1}{5}\log_2 n} \quad \backsim O(n^{\frac{1}{5}}) \tag{6}$$
$$f_7(n) = 10^{10^{10^{10}}} \quad \backsim O(C) \tag{7}$$
$$f_8(n) = 3^{2^n} \tag{8}$$

The final list: $f_5(n), f_8(n), f_2(n), f_3(n), f_4(n), f_1(n), f_6(n), f_7(n)$

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 $9 < 5 + 3 + 2 = 10$. s_i need to $\geq$ sum of s_1 to s_{i-1}

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. ① an algorithm ② time complexity

Algorithm thinking

Sort the stones by their weight, and we record the sum of current weight. Everytime we choose the stone that meets the requirement and has the minimum weight.

Pseudocode

```
Sort stones by weight so that
w1 ≤ w2 ≤ w3 ≤ ··· ≤ wn
                    ↙ stone selected
A ← ∅
int sum = 0
for j = 1 to n {
    if (wj ≥ sum) {
        A ← A ∪ {j}
        sum += wj
    }
}
return A
```

Pf. (by contradiction)

- Let $i_1, i_2, \cdots i_k$ denote set of jobs selected by greedy ···①
- Let $j_1, j_2, \cdots j_m$ denote set of jobs in the OPT with $i_1 = j_1, i_2 = j_2, \cdots i_r = j_r$ for the largest possible value of r. ···②
- We know greedy choose the minimum weight which meets the requirement, so $i_{r+1} < j_{r+1}$. Assume $sum_r = i_1 + i_2 + \cdots + i_r$, we know $sum_r \leq i_{r+1} < j_{r+1}$. ···③
- We can replacing j_{r+1} with i_{r+1} in the OPT, and $j_{r+2} \geq j_{r+1} + sum_r > i_{r+1} + sum_r$. ···④
- Contradicts ⑤

Problem 3:

You are a strict manager of a startup company and you have designed a time table for every employee. You want to assure that your employees work as you expect by checking everyone at least once during one's required work time. Also, you want to check as few times as possible to fit in your busy schedule. Can you design an efficient algorithm to decide the time points at which you need to check, and prove that the algorithm is optimal?

Algorithm thinking

Sort the stones by their weight, and we record the sum of current weight. Everytime we choose the stone that meets the requirement and has the minimum weight.

Pseudocode

Sort stones by weight so that $w_1 \leq w_2 \leq w_3 \leq \cdots \leq w_n$

stone selected

$A \leftarrow \emptyset$

int sum = 0

for j = 1 to n {

  if ($w_j \geq$ sum) {

    $A \leftarrow A \cup \{j\}$

    sum += w_j

  }

}

return A

Pf. (by contradiction)

- Let $i_1, i_2, \cdots i_k$ denote set of jobs selected by greedy ···①
- Let $j_1, j_2, \cdots j_m$ denote set of jobs in the OPT with $i_1 = j_1, i_2 = j_2 \cdots i_r = j_r$ for the longest possible value of r. ···②
- We know greedy choose the minimum weight which meets the requirement, so $i_{r+1} < j_{r+1}$.
  Assume $sum_r = i_1 + i_2 + \cdots + i_r$, we know $sum_r \leq i_{r+1} < j_{r+1}$. ···③
- We can replacing j_{r+1} with i_{r+1} in the OPT, and $j_{r+2} \geq j_{r+1} + sum_r > i_{r+1} + sum_r$. ···④
- Contradicts ②.

Notice that in the problem setting, the time table for each employee specifies a start time and a finish time, and the time tables of different employees may overlap. Besides, checking your employees takes negligible time and does not cause any change in their time tables.

Problem 4:

Suppose you are a security manager of a theater which is troubled by fraud detection. Since ticket scalpers may make some employee's cards to help audience sneak into the backstage, you have confiscated n employee's cards. Each card is a small plastic sheet with a magnetic stripe which encodes data, and belongs to a unique employee. The cards have the same content on the surface because the finance department wants to save cost.

As a security manager, you have to find out whether there is someone colluding with ticket scalpers and copying his employee's card. Your department has an equivalence tester, which can tell whether two cards belong to one person, but cannot tell who it is. (We say two cards are equivalent if they belong to one person.)

Assume that to invoke the equivalence tester, each time you can only take two cards and plug them into the equivalence tester. Please determine whether there is a set of more than $n/2$ cards that are equivalent in n cards, i.e. belonging to one person, with only $O(n \log n)$ invocations of the equivalence tester.

You can assume for this problem that n is a power of 2.

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

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
```

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.