ShanghaiTech University

CS240 Algorithm Design and Analysis

Spring 2022

Problem Set 1

Due: 23:59, Mar. 13, 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.
5. Late homeworks submitted within 24 hours of the due date will be marked down 25%, submitted within 48 hours will be marked down 50%, and will not be accepted past 48 hours.
6. You are required to follow ShanghaiTech's academic honesty policies. You are allowed to discuss problems with other students, but you must write up your solutions by yourselves. You are not allowed to copy materials from other students or from online or published resources. Violating academic honesty can result in serious sanctions.

Problem 1:

Give tight asymptotic bounds for the following recurrences and prove your answers are correct.
1. $T(n) = 2T(\sqrt{n}) + \log n$
2. $T(n) = T(n/4) + T(n/2) + cn$ where c is a constant.

Solution:

1. $R(n) = T(e^n)$, so $R(n) = 2R(n/2) + n$. Using Master Theorem, $R(n) = \theta(n \log n)$, and $T(n) = \theta(\log n \log(\log n))$
2.$T(n) \geq \sum_{i=0}^{\log_4^n} (3/4)^i cn$ and $T(n) \leq \sum_{i=0}^{\log_2^n} (3/4)^i cn \longrightarrow T(n) = \theta(n)$

Problem 2:

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

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

Solution:

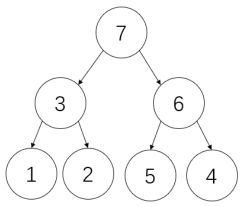
7, 6, 1, 8, 2, 3, 4, 5

Problem 3:

A *number list* is a sequence of length n containing each integer from 1 to n exactly once. For example, $[1]$, $[1, 5, 4, 3, 2]$ and $[3, 1, 2]$ are number lists, but $[1, 3, 1]$, $[5, 2, 1]$ and $[0]$ are not.
Given a number list $A[1...n]$ of length n, we will transform A to a rooted binary tree as follows:

- If A is empty, then return.
- Otherwise, set the root of the tree to be the maximum number in A.
- Let L and R be the parts of A to the left and right of the maximum number, respectively. Transform L and R into rooted binary trees T_L and T_R using the same rules, and make T_L and T_R the left and right subtrees of the root, respectively.

An example of the transformation for the number list $[1, 3, 2, 7, 5, 6, 4]$ is shown below.



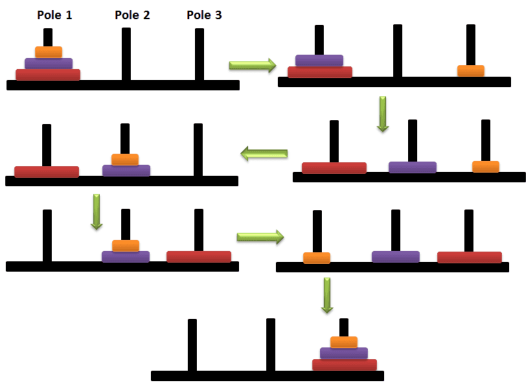
Give a divide-and-conquer algorithm to compute the depth of each node in the transformed tree, where the root has depth 0. For example, for $A = [1, 3, 2, 7, 5, 6, 4]$, the depths of the nodes are $[2, 1, 2, 0, 2, 1, 2]$.

Solution:

Every time we just scan the list and find the max number to be the root for this sub tree. Then split the list by root into left child tree and right child tree, depth+1. Next time we do it on the sub tree again. When the list has no number, end the recursion.

Problem 4:

In the Towers of Hanoi puzzle, we are given a platform with 3 rods and a number of disks of various diameters, which can slide onto any rod. The puzzle begins with the disks stacked on one rod in order of decreasing size, the smallest at the top to the largest at the bottom. The objective of the puzzle is to move the entire stack of disks to the last rod. In each step we can only move one disk, and we can never place a larger disk on top of a smaller one. For example, the figure below shows that we can solve the puzzle with 3 disks in 7 steps.



Give a recursive algorithm to solve the puzzle in the minimum number of steps, and calculate with proof the number of steps needed for moving n disks.

Solution:

Algorithm 1 Hanoi puzzle

```
function HANOI(disk, source, dest, spare)
    if disk == 1 then move disk from source to dest
    else
        MoveDisk(disk-1, source, spare, dest)
        move disk from source to dest
        MoveDisk(disk-1, spare, dest, source)
    end if
end function
```

Step number $= 2^N - 1$ $M(n) = 2M(n-1) + 1$ and $M(1) = 1$

Problem 5:

Given a list of n integers, design an efficient algorithm to determine whether the list is the result of a post-order traversal of a binary search tree.

Solution:

Algorithm 2 Binary Tree

```
function TREE(postorder, left, right)
    if left $\geq$ right then return True
    else
        start $\leftarrow$ left

        while postorder[start] < postorder[right] do start $\leftarrow$ start + 1
        end while
        subroot $\leftarrow$ start

        while postorder[start] > postorder[right] do start $\leftarrow$ starSt + 1
        end while
        return (start == right) & (Tree(postorder, left, subroot - 1)) &
(Tree(postorder, subroot, right - 1))

    end if
end function
```

Problem 6:

Given a string s of length $n = 2^m$ for some m, let s_i ($1 \leq i \leq n$) denote the i'th letter of s, and let $s[i, j]$ ($1 \leq i \leq j \leq n$) denote the substring of s from the i'th to j'th letters. We say s is *x-good* if it satisfies *at least one* of the following conditions.

- The length of s is 1, and it contains the letter x (i.e. $s_1 = x$).
- The length of s is greater than 1, and the first half of s contains only the letter x (i.e. $s[i] = x$ for $1 \leq i \leq n/2$), while the second half of s (i.e. $s[n/2+1, n]$) is an $(x+1)$-good string (where $x+1$ is the next letter after x).
- The length of s is greater than 1, and the second half of s contains only the letter x (i.e. $s[i] = x$ for $n/2+1 \leq i \leq n$), while the first half of s (i.e. $s[1, n/2]$) is an $(x+1)$-good string (where $x+1$ is the next letter after x).

For example, "jjkl", "xxyzwwww" and "aaaacdbb" are j, x and a-good strings, respectively.

Given an input string, we can always transform it to an a-good string by changing its letters one at a time. For example, we can change strings "ceaaaabb" and "bbaaddcc" to a-good strings by changing 4 and 5 letters, respectively. Design an efficient divide-and-conquer algorithm to change an input string to an a-good string using the fewest number of letter changes.

Solution:

We need to compare every two sides' result and return the minimum one. It is wrong to use greedy algorithm.

```
int dfs(int l, int r, char c, int num = 0) {
    if (l >= r) return num + (s[l] != c);
    int mid = (l + r) >> 1;
    //modify left
    int cou1 = 0, cou2 = 0;
    for (int i = l; i <= mid; ++i)
        if (s[i] != c) cou1++;
    int res1 = dfs(mid + 1, r, c + 1, num + cou1);
    //modify right
    for (int i = mid + 1; i <= r; ++i)
        if (s[i] != c) cou2++;
    int res2 = dfs(l, mid, c + 1, num + cou2);
    return min(res1, res2);
}
```

Problem 7:

Consider a set of n players. We want to design a set of games G, where in each game, we split the n players into two teams, such that the teams can have different numbers of players, but each team has at least one player. Furthermore, we want to ensure that for each pair of players, there is some game in G in which the two players play on different teams. Give an algorithm to do this using the least number of games, i.e. minimize $|G|$.

Solution:

Assume T_n is the team. We can first change the whole team into two part, eg $T_1[1..n/2], T_2[n/2+1..n]$ and play a game. Next time we can split those two team again, then we can get $T_1[1..n/4]$, $T_2[n/4+1..n/2]$, $T_3[n/2+1..3n/4]$, $T_4[3n/4+1..n]$. Then let T_1, T_3 be a team, let T_2, T_4 be a team. The next steps are similar. When $T_n[]$ only have one person, stop the algorithm. The number

of games is $\lceil logn \rceil$.
You also can use the digit of every person's id to spilt the team.