CS240 HW1

ZiJia Dai 2022233158

Sept. 20, 2022

Problem 1:

Sort the following functions in ascending order of growth.
$f_1(n) = n^{\sqrt{n}}$

$f_2(n) = n^2$

$f_3(n) = (\log n)^{\log n}$

$f_4(n) = (\log n)^n$

$f_5(n) = n^{2/3}$

$f_6(n) = 666^{\sqrt{n}}$

$f_7(n) = 2^n$

$f_8(n) = 100^{100}$

Solution:

Result: $f_8 < f_5 < f_2 < f_3 < f_6 < f_1 < f_7 < f_4$

- By observing and comparing the magnitude of the exponential of the function, the following obvious size relationships can be obtained: $f_8 < f_5 < f_2, f_2 < f_1, f_3 < f_4, f_6 < f_1, f_6 < f_4, f_6 < f_7$
- $f_4 > f_7, \because logn > 2$, when $n > c$ (c is a constance)
- $f_7 > f_1, \because log(f_7) = nlog(2) = \sqrt{n} \cdot \sqrt{n} \cdot log(2) > log(f_1) = \sqrt{n} \cdot logn$
- $f_3 > f_2, \because f_3 = (logn)^{logn} > c^{logn} > n^2 = f_2$(c is a constance)
- $f_6 > f_3, \because f_3 = (logn)^{logn} < n^{logn}, log(n^{logn}) = logn \cdot logn < n^{\frac{1}{4}} \cdot n^{\frac{1}{4}} \cdot log(666) = log(f_6)$

Problem 2:

Give the time complexity of the following code and explain the reason.

(1)
```
for ( i=1; i < n; i *= 2 ) {
    for ( j = n; j > 0; j /= 2 ) {
        for ( k = j; k <= n; k += 2 ) {
            res += (i * j + j * k );
        }
    }
}
```

(2)
```
for ( i = n; i > 0; i -= 1 ) {
    for ( j = 1; j < n; j *= 2 ) {
        for ( k = 0; k < j; k += 1 ) {
            res += (i * j + j * k );
        }
    }
}
```

Solution:

(a) For i-loop is $O(logn)$, for j-loop is $O(logn)$, for k-loop is $O(n)$, so the total complexity is $O(nlog^2n)$.

(b) For i-loop is $O(n)$, for jk-loop is $O(n)$, because

$$1 + 2 + 4 + ... + \frac{n}{2} = \frac{2(1 - 2^{logn})}{1 - 2} = 2n$$

. So the total complexity is $O(n^2)$.

Problem 3:

Given a connected graph $G = (V, E)$ with at most $n + c$ edges where c is a constant and $n := |V|$. Find the MST of G in $O(n)$ running time. You should not only give a brief explanation of your algorithm but also prove the correctness and time complexity.

Solution:

Way: Set a list as v where the nodes have been visited will be recorded. Save the edges that have been chosen in list e. Then start from recording a random node v_1, visit the connected nodes and choose the next node which has the shortest edge to record as v_2. After that obtain v_i from $v_{i-1}(i = 1, .., n)$ in the same way, choosing the one connected with and not in v which has shortest edge. Repeat till all nodes are in v.

Correctness pf: Assume that all edge have different weights, and the tree produced by algorithm denotes as T.

(1) Produce a tree. The visited nodes won't be put in v twice, so the T remains acyclic through the entire algorithm. The algorithm will run till all nodes are in v, and all of them are found by the shorest edges which connect them with the prior node, so T is connected.

(2) Each edge in T is in every MST. Separate G into S and T, If there exist $e' = (S, T)$ with $w(e') < w(e)$ and has the same end node with e, e' must be found before e because the algorithm always choose the shortest edge.

(3) Tree T is a MST. From (1), it produces a tree T with $|V| - 1$ edges. From (2), each edge in T is in every MST. But, every MST contains exactly $|V|-1$ edges so every MST must be T.

Time Complexity: The algorithm visit the whole edges only once and there are most $n + c$ edges(c is a constant), So the time complexity will be $O(n)$.

Problem 4:

In the new semester, SIST offers some courses for students. Students are required to take only one course. We asked n students "how many other students took the same course as you?" and collected the answers in an integer array answer where answer[i] is the answer of the i-th student.

Given an array answer, return the minimum number of students that could be in SIST.

Solution:

The minimum number of students comes from when some students have the same answer we consider they are in the same class. But if too many students have the same answer that is mean the number of them lager than their answer, there might be more than one same class. According to the analysis above, the pseudocode of solution is given:

Algorithm 1 Problem 4

Input: Array answer, n
Output: The minimum number of student sum
A = Quicksort(answer)
$same = 1, sum = 0$
foreach *item in A* **do**
  if $A_i = A_{i+1}$ **then**
    1 $same = same + 1$
  else
    2 $sum = sum + (1 + [\frac{same-1}{A_i}]) \cdot A_i$, $same = 1$
  end
end
return sum

Problem 5:

Suppose you are a security guard at ShanghaiTech University and now troubled by fraud detection. Since criminals may make some student cards to help thieves sneak into the campus, you have confiscated n student cards. Each card is a small plastic sheet with a magnetic stripe which encodes data, and belongs to a unique student. The cards have the same content on the surface because the student service department wants to save cost.

As a security guard, you have to find out whether there is someone colluding with the criminals and copying his student card. You 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 $O(n\ logn)$ invocations of the equivalence tester.

Solution:

We can use divide and conquer.

- Divide. Separate the problem P into P_1 and P_2, where they all have the same number of cards and target, determining whether there is a set of more than half of cards that are equivalent in n cards. Repeat this step until there only two cards in a problem.
- Solve. Compare two cards in each unit problem to determine whether they are equivalent. Differet card will be given different tags.
- Conquer. Conquer the divided two problems by testing cards between two problem groups. For example, select first card from P_1 and test cards in P_2 one by one, if they are equivalent, make their tags same but different to others. Then repeat slecting next not equivalent card in P_1 to test in P_2 until two problems are combine to bigger one.

At last, we just need to count the number of equivalent cards which is showed in their tags.