CSCI261 Analysis of Algorithms, Fall 2020/21 Homework 1

Due Friday, September 4, 2020, 11:59pm

Problem 1

Rank the following functions by order of growth; that is, find an arrangement $g_1(n), g_2(n), \ldots,$ $g_{21}(n)$ of functions satisfying $g_i(n) = O(g_{i+1}(n))$ for every $i \in \{1, \ldots 20\}$. Partition your list into equivalence classes such that $f(n)$ and $g(n)$ are in the same class if and only if $f(n) = \Theta(g(n))$. You do not have to prove your answers.

$$\begin{array}{ccccccc}
(\log n)^3 & 3 & \log n & (\log n)^{\log n} & n^3 & n \log n & n/\log n \\
\log_3 n & n! & n^{1/3} & n^2 \log n & 10^{100} n^2 + n^3 & 2^{3n} & n^2 \\
n^n & 8^{n-1} & 2^{n^3} & 8^{\log n} & n - \log n & \log(n^3) & 3^{2n}
\end{array}$$

Remarks:

- In this class we use $\log n$ to denote the logarithm base 2.
- The Stirling's formula is helpful when dealing with $n!$. The Stirling's formula is:

$$n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n (1 + O(\frac{1}{n}))$$

- Use also this fact: for any constants $b_1, b_2 > 0$:

$$\log^{b_1} n = O(n^{b_2}) \quad \text{and} \quad n^{b_2} \neq O(\log^{b_1} n)$$

  In words, logarithm of n raised to any power grows slower than any power of n.

Problem 2

We have a computer network with n computers. For each pair of computers we know whether this pair is directly connected by a cable. The cables are bi-directional, that is for a cable connecting computer A to computer B, we can route a packet from A to B, or from B to A.

To protect this network, we would like to monitor the traffic through all the cables. We will designate some of the computers as "trusted" — a software engineer will be assigned to each trusted computer and they will monitor the traffic through all the cables the trusted computer is directly connected to. However, such monitoring is expensive. Therefore, we want to designate as few computers as trusted as possible while still monitoring every single cable in the network.

Professor Smart came up with a brilliant plan: Let's designate as trusted the computer that is directly connected to the largest number of other computers (if there are more such computers, choose one of them). Then pretend to remove this computer and all its cables from the network, getting a smaller network. Repeat this process, designating the next trusted computer, while there are still cables in the network.

For example, if our network has four computers A, B, C, and D, and the pairs of computers directly connected by cables are (A,B), (A,C), (B,C), and (C,D), then Professor Smart's algorithm would designate computer C as trusted, since it is connected to three other computers. After removing C and its cables, we are left with a single cable connecting A and B, so the algorithm would designate one of these computers as trusted — maybe it chooses A. We would have two designated computers, A and C, which is the smallest possible number of computers covering all the cables in this computer network.

- Give a pseudo code for Professor Smart's algorithm. In particular: Specify the variables and/or data structure(s) in which you store the input data. Give names to the main data structures and variables in your pseudo code. Write the pseudo code using indentation, while- and/or for-loops, if-conditions, and other typical pseudo code keywords.
- Estimate the algorithm's running time using asymptotic notation as a function of n and reason your estimate. Aim for a low polynomial estimate (it does not have to be optimal).
- Does the algorithm work? That is, does it always produce a smallest set of computers that cover all the cables? If not, provide a counterexample. That is, (a) draw a computer network on which the algorithm fails, (b) trace the algorithm on your example and show the algorithm's output (the set of computers it chose — it is ok to assume that the algorithm made a "bad choice" when having multiple computers to choose from), and (c) highlight an optimal solution (a smallest set of computers that cover all the cables).

Note: This problem is known as the Vertex Cover. We will learn more about it later in the term.

Problem 3

We have p house plants in planters of various sizes $s_1, s_2, \ldots, s_p$. The plants are getting too big and each plant has to be replanted to a larger planter than the one it currently has. The good news is that we have r extra planters of sizes $t_1, t_2, \ldots, t_r$. Is it possible to replant the plants, one at a time, always putting the plant to a larger currently empty planter? Let $n = p + r$. Design an $O(n \log n)$ algorithm which decides whether such replanting is possible.

For example, for $p = 4$ and $r = 2$ and planters $s_1, \ldots, s_4 = 1, 4, 3, 2$ and $t_1, t_2 = 5, 1$ we can first move the plant in the planter of size 4 to the empty planter of size 5, then move the plant in the planter of size 3 to the now-empty planter of size 4, then move the plant in the planter of size 2 to the now-empty planter of size 3, and finally move the plant in the planter of size 1 to the now-empty planter of size 2. (We will be left with two empty planters of size 1 each.) Thus, for the input, the answer is YES. On the other hand, for input $p = 3$ and $r = 1$ and planters $s_1, s_2, s_3 = 2, 2, 2$ and $t_1 = 3$ the answer is NO.

Problem 4

For two sets of numbers A and B we define the operation $A + B = \{a + b \mid a \in A, b \in B\}$. Design an $O(n^2 \log n)$ algorithm that, for a given set S of n numbers, outputs the size (that is, the number of elements) of $S + S$. For example, if $S = \{1, 2, 4\}$, then $S + S = \{2, 3, 4, 5, 6, 8\}$ and its size is 6.