CS211: Solutions to Problem Set 1

1. **(2.1-8, CLR) We can extend the O notation to the case of two parameters n and m that can go to infinity independently at different rates. For a given function $g(n,m)$, we denote $O(g(n,m))$ the set of functions**

$$\begin{aligned} O(g(n,m)) &= \{f(n,m) : \text{there exist positive constants } c, n_0, m_0 \text{ such that} \\ &\quad 0 \leq f(n,m) \leq cg(n,m) \text{ for all } n \geq n_0, m \geq m_0\} \end{aligned}$$

Give corresponding definitions for $\Omega(g(n,m))$ and $\Theta(g(n,m))$.

Extending the one-dimensional definitions for $\Omega(g(n))$ and $\Theta(g(n))$ to two dimensions, we get the equations

$$\begin{aligned} \Omega(g(n,m)) &= \{f(n,m) : \text{there exist positive constants } c, n_0, m_0 \text{ such that} \\ &\quad 0 \leq cg(n,m) \leq f(n,m) \text{ for all } n \geq n_0, m \geq m_0\} \end{aligned}$$

$$\begin{aligned} \Theta(g(n,m)) &= \{f(n,m) : \text{there exist positive constants } c_1, c_2, n_0, m_0 \text{ such that} \\ &\quad 0 \leq c_1 g(n,m) \leq f(n,m) \leq c_2 g(n,m) \text{ for all } n \geq n_0, m \geq m_0\} \end{aligned}$$

2. **(2.3) 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 $O(g(n))$.**

$$\begin{array}{ll} f_1(n) = n^{2.5} & f_2(n) = \sqrt{2n} \\ f_3(n) = n + 10 & f_4(n) = 10^n \\ f_5(n) = 100^n & f_6(n) = n^2 logn \end{array}$$

$f_2(n) \leq f_3(n) \leq f_6(n) \leq f_1(n) \leq f_4(n) \leq f_5(n)$

3. **(2.4) 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 $O(g(n))$.**

$$\begin{array}{ll} f_1(n) = 2^{\sqrt{logn}} & f_2(n) = 2^n \\ f_3(n) = n^{4/3} & f_4(n) = n(logn)^3 \\ f_5(n) = n^{logn} & f_6(n) = 2^{2^n} \\ f_7(n) = 2^{n^2} & \end{array}$$

$f_1(n) \leq f_4(n) \leq f_3(n) \leq f_5(n) \leq f_2(n) \leq f_7(n) \leq f_6(n)$

One way to look at the functions is to take the log of each function:

$$\begin{array}{ll} f_1(n) = \sqrt{logn} & f_2(n) = n \\ f_3(n) = \frac{4}{3} * logn & f_4(n) = logn + 3 * log(logn) < logn + 1/4logn = \frac{5}{4} * logn \\ f_5(n) = logn * logn & f_6(n) = 2^n \\ f_7(n) = n^2 & \end{array}$$

4. **Suppose that each row of an $n \times n$ array A consists of 1's and 0's such that, in any row i of A, all the 1's come before any 0's in that row. Suppose further that the number of 1's in row i is at least the number in row $i + 1$, for $i = 0, 1, ..., n - 2$. Assuming A is already in memory, describe a method running in $O(n)$ time for counting the number of 1's in the array.**

Suppose that A is square matrix with n rows and n columns. For convenience we assume a 0 indexed matrix, so A[0, 0] gives the top-left corner and A[$n - 1$, $n - 1$] gives the bottom-right corner. The idea is to keep a running total of the 1s, while moving from the top-left corner to the bottom-right corner. When you encounter a 1, move to the right, whenever you encounter a 0 or the right wall, add the column number to the total and then move down. Stop when you reach the bottom wall. The algorithm is correct because we are adding the total number of 1s in each row—the column position of the final 1 in a row indicates how many 1s precede it. The algorithm requires $O(n)$ time because we either move right or move down.

5. **(3.2) Give an algorithm to detect whether a given undirected graph contains a cycle. If the graph contains a cycle, then your algorithm should output one. (It should not output all cycles in the graph, just one of them.) The running time of your algorithm should be $O(m + n)$ for a graph with n nodes and m edges.**

Modify DFS (an O(m+n) algorithm) to keep track of the path it takes. Starting at some node (doesn't matter which in an undirected graph), when it reaches a node that it has already visited, it looks back in the path to see the last time it visited that node. If it's more than 2 "hops", it's found a cycle.