CS 577 Assignment 2 – Asymptotic Analysis & Graphs Spring 2023

Answer the questions in the boxes provided on the question sheets. If you run out of room for an answer, add a page to the end of the document.

Name: Matej Popovski Wisc id: 908 3541632

Asymptotic Analysis

1. *Kleinberg, Jon. Algorithm Design (p. 67, q. 3, 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 $O(g(n))$.

   (a) $f_1(n) = n^{2.5}$
   $f_2(n) = \sqrt{2n}$
   $f_3(n) = n + 10$
   $f_4(n) = 10n$
   $f_5(n) = 100n$
   $f_6(n) = n^2 \log n$

   $f_2, f_3, f_4, f_5, f_6, f_1$

   (b) $g_1(n) = 2^{\log n}$
   $g_2(n) = 2^n$
   $g_3(n) = n(\log n)$
   $g_4(n) = n^{4/3}$
   $g_5(n) = n^{\log n}$
   $g_6(n) = 2^{(2^n)}$
   $g_7(n) = 2^{(n^2)}$

   $g_1, g_3, g_4, g_5, g_2, g_7, g_6$

2. *Kleinberg, Jon. Algorithm Design (p. 68, q. 5).* Assume you have a positive, non-decreasing function f and a positive, increasing function g such that $g(n) \geq 2$ and $f(n)$ is $O(g(n))$. For each of the following statements, decide whether you think it is true or false and give a proof or counterexample.

(a) $\log_2 f(n)$ is $O(\log_2 g(n))$

True. $f(n) \leq c \cdot g(n)$ $\exists c, n_0 \mid \forall n > n_0 \Rightarrow T$
Since $f(n)$ is non decreasing, we can choose it to be constant c_1. $\Rightarrow$ $c_1 \leq c_2 \cdot g(n)$
$\Rightarrow \log c_1 \leq c_3 + \log(g(n))$
$\Rightarrow$ After some n, the right side is going to be larger no matter the constants.

(b) $2^{f(n)}$ is $O(2^{g(n)})$

False
Suppose $f(n) = 2n$ and $g(n) = n$
$2^{f(n)} = 2^{2n} = 4^n$ and $2^{g(n)} = 2^n$
As n increases, $4^n > c \cdot 2^n$ because true no matter c.

(c) $f(n)^2$ is $O(g(n)^2)$

True.
For all $n > n_0$, $f(n) \leq c \cdot g(n)$ So then
$f(n)^2 \leq c \cdot g(n)^2$ for some values of c.

3. *Kleinberg, Jon. Algorithm Design (p. 68, q. 6).* You're given an array A consisting of n integers. You'd like to output a two-dimensional n-by-n array B in which $B[i, j]$ (for $i < j$) contains the sum of array entries $A[i]$ through $A[j]$ — that is, the sum $A[i] + A[i + 1] + ... + A[j]$. (Whenever $i \geq j$, it doesn't matter what is output for $B[i, j]$.) Here's a simple algorithm to solve this problem.

```
for i = 1 to n
  for j = i + 1 to n
    add up array entries A[i] through A[j]
    store the result in B[i, j]
  endfor
endfor
```

(a) For some function f that you should choose, give a bound of the form $O(f(n))$ on the running time of this algorithm on an input of size n (i.e., a bound on the number of operations performed by the algorithm).

$f(n) = n^3$. The outer loop runs for exactly n iterations and the inner loop runs at most n iterations. Adding $A[i]$ through $A[j]$ takes $O(j-i+1)$ which is at most $O(n)$. So overall $O(n^3)$

(b) For this same function f, show that the running time of the algorithm on an input of size n is also $\Omega(f(n))$. (This shows an asympto tically tight bound of $\Theta(f(n))$ on the running time.)

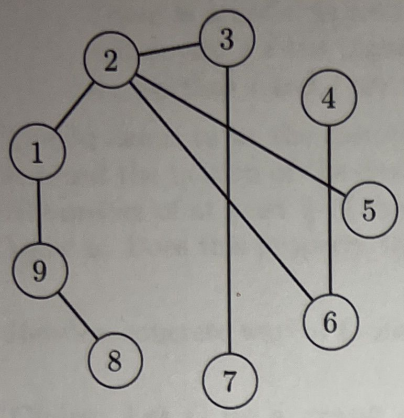
Consider the times during the execution of the algo when $i \leq n/4$ and $j \geq 3n/4$. Adding up entries $A[i]$ through $A[j]$ is at least $n/2$. There are $(n/4)^2$ pairs (i,j) with $i \leq n/4$ and $j \geq 3n/4$. The given algo enumerates over all of them, so it must perform at least $n/2$ operations. Therefore the algorit must perform at least $n/2 \cdot (n/4)^2 = n^3/32$ this is $\Omega(n^3)$ as desired.

(c) Although the algorithm provided is the most natural way to solve the problem, it contains some highly unnecessary sources of inefficiency. Give a different algorithm to solve this problem, with an asymptotically better running time. In other words, you should design an algorithm with running time $O(g(n))$, where $\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$.

For $i = 1, 2, \ldots, n$
  Set $B[i, i+1]$ to $A[i] + A[i+1]$
For $k = 2, 3, \ldots, n-1$
  For $i = 1, 2, \ldots, n-k$
    Set $j = i + k$
    Set $B[i, j]$ to be $B[i, j-1] + A[j]$
The O of this algo is $O(n^2)$

Graphs

4. Given the following graph, list a possible order of traversal of nodes by breadth-first search and by depth-first search. Consider node 1 to be the starting node.



BFS: 1, 2, 9, 3, 5, 6, 8, 7, 4

DFS: 1, 2, 3, 7, 5, 6, 4, 9, 8

5. *Kleinberg, Jon. Algorithm Design (p. 108, q. 5).* A binary tree is a rooted tree in which each node has at most two children. Show by induction that in any binary tree the number of nodes with two children is exactly one less than the number of leaves.

Let n be the num of nodes with 2 children and ℓ be the num of leaves

Base Case: $n=1$. Root by itself is a leaf with no children so statement holds.

1. Adding a node to an existing node with no children does not change n or L. The node that was added becomes a leaf, but its parent is not a leaf anymore making the num of ℓ the same.
2. Adding a node to one that has one child increases ℓ by one, and also increases n by one - so it holds.
3. As stated you cant add a node to a full node.

1, 2, 3 show that all scenarios, adding a node maintains $n=\ell-1$ shows that adding a node doesnt change n and ℓ. 2 shows n and ℓ increment together

Hence $n=\ell-1$ holds for all cases

6. *Kleinberg, Jon. Algorithm Design (p. 108, q. 7).* Some friends of yours work on wireless networks, and they're currently studying the properties of a network of n mobile devices. As the devices move around, they define a graph at any point in time as follows:

> There is a node representing each of the n devices, and there is an edge between device i and device j if the physical locations of i and j are no more than 500 meters apart. (If so, we say that i and j are "in range" of each other.)

They'd like it to be the case that the network of devices is connected at all times, and so they've constrained the motion of the devices to satisfy the following property: at all times, each device i is within 500 meters of at least $\frac{n}{2}$ of the other devices. (We'll assume n is an even number.) What they'd like to know is: Does this property by itself guarantee that the network will remain connected?

Here's a concrete way to formulate the question as a claim about graphs.

Claim: Let G be a graph on n nodes, where n is an even number. If every node of G has degree at least $\frac{n}{2}$, then G is connected.

Decide whether you think the claim is true or false, and give a proof of either the claim or its negation.

Assume G is not connected. example: there are two unconected components C_1 and C_2
Since C_1 and C_2 are sub graphs, a node in C_1 and C_2 must have a degree of at least $n/2$.
This means C_1 and C_2 must have at least $n/2+1$ nodes each.
The total number of nodes is $\left(\frac{n}{2}+1\right)+\left(\frac{n}{2}+1\right)=n+2$
This contradicts the fact that G has n nodes, so the claim is true.