CS 577 Assignment 2 – Asymptotic Analysis & Graphs Fall 2022

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: Yong Sang Park Wisc id: 908 022 3069

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

   Solution:

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

   $\cancel{n^2}\cdot\sqrt{n} \; ? \; \cancel{n^2}\log n$

   $10^{\sqrt{n}} > n$

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

   Solution:

   $n(\log n) < n^{\frac{4}{3}} < 2^{\log n} < 2^n < n^{\log n} < 2^{n^2} < 2^{2^n}$ $\Rightarrow$ $g_1(n), g_3(n), g_4(n), g_2(n), g_5(n), g_7(n), g_6(n)$

   $\cancel{n}(\log n) \; ? \; \cancel{n}\cdot n^{\frac{1}{3}}$

   $n < \underline{10^{n^{1/3}}}$

   $n^{\log n} \; ? \; 2^{2^n}$ $\quad$ $n\log n \; ? \; 2^{n^2}$

   $(\log n)^2 < 2^n \log 2$ $\quad$ $(\log n)^2 < n^2 \log 2$

$n\log n \; ? \; 2^{\log n}$

$\underline{\log n \cdot \log n > \log n \log 2}$

2. *Kleinberg, Jon. Algorithm Design (p. 68, q. 5).* Assume you have positive, non-decreasing functions f and g such that $f(n)$ 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 $O(\log_2 g(n))$

Solution:

$f(n) \leq c \cdot g(n)$

$\log_2 f(n) \leq \log_2 (c \cdot g(n)) = \log_2 c + \log_2 g(n)$

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

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

Solution:

$f(n) \leq C \cdot g(n)$

$2^{f(n)} \leq 2^{C \cdot g(n)} = (2^{g(n)})^{C}$

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

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

Solution:

$f(n) \leq C \cdot g(n)$

$f(n)^2 \leq (C \cdot g(n))^2 = C^2 \cdot g(n)^2$

$\Rightarrow f(n)^2$ is $O(g(n)^2)$

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                                        cost   time
  for j = i + 1 to n                                  C1     n+1
    add up array entries A[i] through A[j]            C2     (n+1)-i
    store the result in B[i, j]                       C3     j-i => n
  endfor                                              C4     1
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).

Solution: $O(C_1(n+1) \cdot C_2((n+1)-i) \cdot (C_3 \cdot n + C_4))$ the big O of f(n) is 3rd degree polynomial,
so $f(n) \leq O(f(n)) = O(n^3)$

(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 asymptotically tight bound of $\Theta(f(n))$ on the running time.)

Solution:
the big Ω of f(n) is 3rd degree polynomial,
So $\Omega(f(n)) = \Omega(n^3) \leq f(n)$

(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 $\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$.

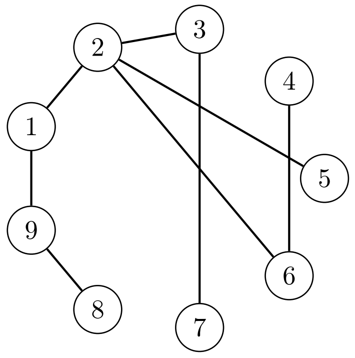
Solution:

```
for i=1 to n
  B[i,i+1] = A[i]
  for j=i+2 to n
    B[i,j] = B[i,j-1] + A[j]
  end for
end for
return B
```

the big O of g(n) is $O(g(n)) = n^2$
which means $\lim_{n\to\infty} \frac{g(n)}{f(n)} = \lim_{n\to\infty} \frac{n^2}{n^3} = 0$

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.



Solution:

breadth-first: 1-2-9-3-5-6-8-7-4

depth-first: 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.

Solution:

Basecase

For a single node with no children (root), the number of nodes with two children is exactly one less than the number of leaves.

Inductive case

Case 1. Adding a node to an existing node that has no children
It does not change the number of nodes with two children, nor the number of leaves.

Case 2. Adding a node to an existing node that has a child
It increases the number of nodes with two children by 1, and also increases the number of leaves by 1.

Since the number of nodes with two children starts 1 less than the number of leaves, and adding a node to the tree does not change them or increases them by 1, the difference is always 1.

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.

Solution:

Suppose that G is not connected with every node with degree of at least $\frac{n}{2}$

It means it has two components C_1 & C_2.

Since every node must have degree of at least $n/2$, a node in C_1 is connected to at least $\frac{n}{2}$ other nodes. That means, C_1 has $(\frac{n}{2}+1)$ nodes.

Similarily, C_2 has $(\frac{n}{2}+1)$ nodes.

It gives a total nodes (nodes of C_1) + (nodes of C_2) $= (\frac{n}{2}+1)+(\frac{n}{2}+1) = n+2$

which is a contradiction that we have total n nodes

Hence, G must be connected. ■

Coding Question

7. Implement depth-first search in either C, C++, C#, Java, or Python. Given an undirected graph with n nodes and m edges, your code should run in $O(n + m)$ time. Remember to submit a makefile along with your code, just as with week 1's coding question.

   Input: the first line contains an integer t, indicating the number of instances that follows. For each instance, the first line contains an integer n, indicating the number of nodes in the graph. Each of the following n lines contains several space-separated strings, where the first string s represents the name of a node, and the following strings represent the names of nodes that are adjacent to node s. You can assume that the nodes are listed line-by-line in lexicographic order (0-9, then A-Z, then a-z), and the adjacent nodes of a node are listed in lexicographic order. For example, consider two consecutive lines of an instance:

```
0, F
B, C, a
```

   Note that $0 < B$ and $C < a$.

   Input constraints:

   - $1 \le t \le 1000$
   - $1 \le n \le 100$
   - Strings only contain alphanumeric characters
   - Strings are guaranteed to be the names of the nodes in the graph.

   Output: for each instance, print the names of nodes visited in depth-first traversal of the graph, *with ties between nodes visiting the first node in lexicographic order.* Start your traversal with the first node in lexicographic order. The names of nodes should be space-separated, and each line should be terminated by a newline.

   Sample:

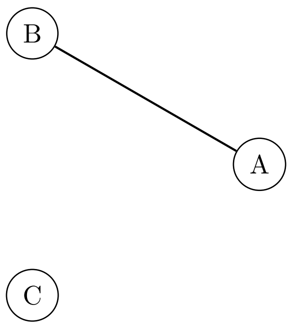
   Input:

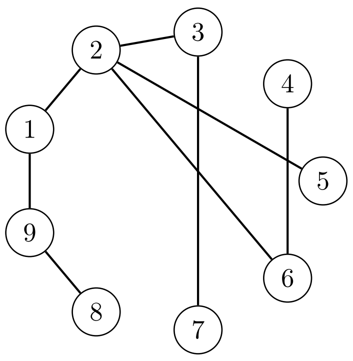
```
2
3
A B
B A
C
9
1 2 9
2 1 3 5 6
3 2 7
4 6
5 2
6 2 4
7 3
8 9
9 1 8
```

   Output:

```
A B C
1 2 3 7 5 6 4 9 8
```

   The sample input has two instances. The first instance corresponds to the graph below on the left. The second instance corresponds to the graph below on the right.


B
A
C


2
3
4
1
5
9
6
8
7