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: Mahir Khan Wisc id: mhkhan9

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) $f_1(n) = n^{2.5}$ → $n^{2.5}$ is $O(n^3)$
   $f_2(n) = \sqrt{2n}$ → $(2n)^{1/2}$ is $O(n^{1/2}) = O(\sqrt{n})$
   $f_3(n) = n + 10$ → $O(n)$
   $f_4(n) = 10n$ → $O(n)$
   $f_5(n) = 100n$ → $O(n)$
   $f_6(n) = n^2 \log n$ $f_6 = n^2 \log n \to O(n^2 \log n)$

   $f_3(n) = n+10 \to O(n)$
   $f_4(n) = 10n \to O(n)$
   $f_5(n) = 100n \to O(n)$
   $f_1(n) = n^{2.5} \to O(n^3)$
   $f_2(n) = \sqrt{2n} = (2n)^{1/2} \to O(\sqrt{n})$

   Ascending order (Lowest → Highest)
   $f_2(n), f_3(n), f_4(n), f_5(n), f_6(n), f_1(n)$
   $O(\sqrt{n})$ $O(n)$ $O(n)$ $O(n)$ $O(n^2 \log n)$ $O(n^3)$

   (b) ~~$g_1(n) = 2^{\log n}$~~ → $n^{\log 2} = n$ is $O(n)$
   $g_2(n) = 2^n$ → exponential
   ~~$g_3(n) = n(\log n)$~~ → $O(n \log n)$
   ~~$g_4(n) = n^{4/3}$~~ → $n^{4/3}$ is $O(n^2)$
   ~~$g_5(n) = n^{\log n}$~~ → $n^{\log n}$ is $O(n^2)$
   $g_6(n) = 2^{(2^n)}$ → 2^{2^n} grows exponentially, much faster than 2^n
   $g_7(n) = 2^{(n^2)}$ → grows polynomially, slower than 2^{2^n}

   Assuming base of log is 2

   Least Growth Rate → Highest Growth Rate
   $g_1(n), g_3(n), g_4(n), g_5(n), g_2(n), g_7(n), g_6(n)$
   $O(n)$ $O(n \log n)$ $O(n^2)$, $O(n^2)$, $O(2^n)$, $O(2^{n^2})$ $O(2^{2^n})$

2. *Kleinberg, Jon. Algorithm Design (p. 68, q. 5).* Assume you have a positive, non-decreasing function f and a positive, non-decreasing 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) $2^{f(n)}$ is $O(2^{g(n)})$

No, false, assume the case where $g(n) = n$ and $f(n) = 2n$. Thus $2^{g(n)} = 2^n$ and $2^{f(n)} = 2^{2n} = 4^n$

$\subset$ 4^n is not $O(2^n)$, thus statement is false

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

since $f(n)$ is $O(g(n))$, this means $f(n) \leq c \cdot g(n)$.

thus, squaring both sides, the following still holds:

$(f(n))^2 \leq c \cdot (g(n))^2$ $\longrightarrow$ Therefore statement is true.

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

True. This is because there is no scenario where $\log_2 f(n)$ is asymptotically greater than $\log_2(g(n))$

$\subset$ the only case where it is greater is if $g(n) = 1$, however, we know $g(n) \geq 2$, thus statement is true!

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).

outer loop is n operations
inner loop is n operations at worst case
Also, adding up array entries A[i] through A[j] can take up to n operations
} thus all 3 of these are n operations, ∴ f(n) is $O(n^3)$

(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.)

We know that the outer loop will always take n operations to complete, and that the inner loop still needs to go through every entry in the array, thus also requiring n operations to run.
↳ Also, once again, adding up each entry from i through j can require n operations to add them all up, thus even in the best case the function is $\Theta(n^3)$.
Thus if f(n) is $\Theta(n^3)$ and $O(n^3)$, then by definition it's average running time is $\Omega(n^3)$

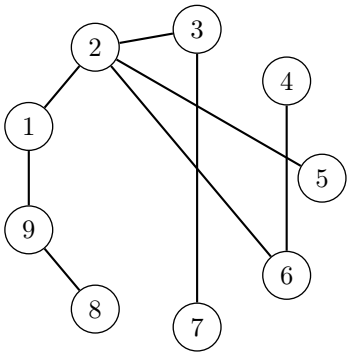
(c) Although the algorithm provided is the most natural way to solve the problem, it contains some highly unnecessary sources of inefficiency. Give a different algorithm to solve this problem, with an asymptotically better running time. In other words, you should design an algorithm with running time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

```
For i = 1 to n
    set B[i, i+1] to A[i] + A[i+1]

for j = 2 to n-1
    for k = 1 to j
        m = j + k
        set B[m, k]
```

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.



DFS

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

BFS

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

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.

At the base case, when there is only 1 node at the root of the binary tree, there are no other nodes, thus there are zero nodes with 2 children. Plus, the root has no children, therefore it is a leaf.

For the inductive hypothesis, assume that when there is a tree with $k-1$ nodes with 2 children, there are k leaves in the tree (thus the claim is true).

For the inductive step: we assumed in the inductive hypothesis that the claim holds, as in there is a tree with $k-1$ nodes with 2 children, as well as k leaves. If one of the leaves gains 2 children, then the number of nodes with 2 children increases by 1 (i.e: $(k-1)+1=k$) and the number of leaf nodes decreases by 1 (i.e: $k-1$). However, adding 2 new nodes to a leaf node means that those new child nodes are actually leaf nodes themselves. Thus, the number of leaf nodes is actually $k-1+2=k+1$, thus 1 more leaf node than there are nodes with 2 children.

Thus, we have shown by induction that in any binary tree with two children, the number of leaves is exactly 1 greater than the number of nodes with two children.

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.

I believe this claim is true, and will do a proof by contradiction to show this:

G = graph with n nodes, where n = even = $2i$, $i \in \mathbb{N}$, and for each node h, $\deg(h) \geq \frac{n}{2}$

↳ For a graph with these properties, assume that the graph is not connected. If the graph had 2 vertices, and the degree of each node is $\geq \frac{n}{2}$, where $n=2$, then the degree must be 1. Thus, the 2 nodes are connected to each other since there are no other nodes to connect to, and thus the graph is connected, going against the initial statement that the graph is not connected.

↳ thus, by proof by contradiction, we have shown that the graph with the given properties is connected.

Coding Question: DFS

7. Implement depth-first search in either C, C++, C#, Java, Python, or Rust. 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 the first 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.

   The order of the nodes in the adjacency list is important, as it will be used as the tie-breaker. For example, consider an instance

```
4
xy v0 b
b xy
v0 xy a
a v0
```

   The tie break priority is `xy` → `v0` → `b` → `a`, so your search should start at `xy`, then choose `v0` over `b` as the second node to visit. Overall, your code should produce the following output:

```
xy v0 a b
```

   Input constraints:

   - $1 \leq t \leq 1000$
   - $1 \leq n \leq 100$
   - 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 input order*. Start your traversal with the first node in input order. The names of nodes should be space-separated, the output of each instance should be terminated by a newline, and the lines should have **no trailing spaces**.

   Sample Input:

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

   Sample Output:

```
A B C
1 2 6 4 5 3 7 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.