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: Zhuoyan Yu Wisc id: zyu 444

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

   $f_2(n)$, $f_3(n)$, $f_4(n)$, $f_5(n)$, $f_6(n)$, $f_1(n)$

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

   $n\log n \le n^{\frac{4}{3}} \le n^{\log n} \le 2^{\log n} = 2^n \le 2^{n^2} \le 2^{2^n}$

   $g_3(n)$ $g_4(n)$ $g_5(n)$ $g_1(n)$ $g_2(n)$ $g_7(n)$ $g_6(n)$

2. *Kleinberg, Jon. Algorithm Design (p. 68, q. 5).* Assume you have positive 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 $O(\log_2 g(n))$

true $\exists\ c, N$ s.t. $f(n) \le c\,g(n)$ as $n \ge N$

for $n \ge N$, we have $\log_2 f(n) \le \log_2 c + \log_2 g(n)$

then $\log_2 f(n) = O(\log_2 g(n))$

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

true $\exists\ c, N$ s.t. $0 \le f(n) \le c\,g(n)$ as $n \ge N$

for $n \ge N$, we have $2^{f(n)} \le \left(2^{g(n)}\right)^c$

then $2^{f(n)} = O\left(2^{g(n)}\right)$

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

true $\exists\ c, N$ s.t. $0 \le f(n) \le c\,g(n)$ as $n \ge N$

for $n \ge N$, we have $f(n)^2 \le g(n)^2$

then $f(n)^2 = O\left(g(n)^2\right)$

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
```

c_1 n
c_2 $\sum_{i=2}^{n} t_i$
c_3 $\sum_{i=2}^{n} t_i$
c_4 $\sum_{i=2}^{n} t_i$

(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).

let $T(n)$ be the time complexity.

$T(n) \leq c_1 \cdot n + c_2 \sum_{i=2}^{n} t_i + c_3 \sum_{i=2}^{n} t_i + c_4 \sum_{i=2}^{n} t_i \leq c \cdot n \cdot (n+1) = O(n^2)$

$f(n) \in O(n^2)$

(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.)

$T(n) \geq c_1 \cdot n + c_2 \cdot \sum_{i=2}^{n} (t_i - 1) + c_3 \left(\sum_{i=2}^{n} t_i - 1 \right) + c_4 \sum_{i=2}^{n} (t_i - 1)$

$\geq \tilde{c} \cdot n \cdot (n-1) = \Omega(n^2)$

$f(n) \in \Omega(n^2)$ then we have $f(n) \in \Theta(n^2)$

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

45 1.b

```
merge(L1, L2): // merge 2 sorted array
 index1 = 0; index2 = 0;
 res = []; add2 = false
if index1 == L1.length && index2 == L2.length:
    return res;
while true:
     add2 = false;
if index1 == L1.length:
    add2 = true;
else if index2 != L2.length && L1(index1) ≤ L2(index2):
    add2 = true;
if (!add2):
    res.append(L1(index1));
    index1++;
else:
```

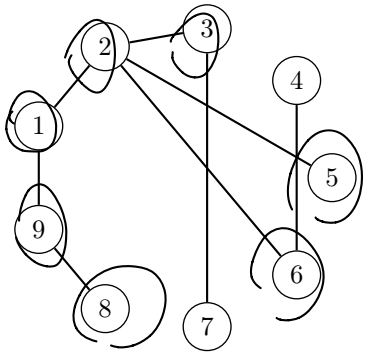
```
mergeSort(L):
 bin = ⌊L.length/2⌋
L1 = L[0 : bin]
L2 = L[bin : L.length]
L1 = L1.mergeSort;
L2 = L2.mergeSort;
 return merge(L1, L2)
```

the complexity of mergeSort is $O(n \log n) \leq O(n^2)$

res . append L2(index2);
index2 ++;

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.



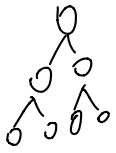
DFS: 1 2 3 7 5 6 4 9 8

BFS: 1 2 9 3 5 6 8 4 7

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 denote the number of nodes with 2 children.

m denote the number of leaves.



if $n=0$, $m=1$

then consider a binary tree T with: $n \geq 1$, $m \geq 2$

① at least one parent of a leave has 2 children (special case: full binary tree)

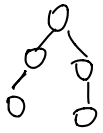
then we choose any one of such parents, consider it's node c

then we trim c, we get T' with $n' = n-1$, $m' = m-2+1 = m'-1$

we have $n' = m' - 1$

$n = n'+1 = m' \Rightarrow m = m'+1 = n+1$ □

② all parents of leaves has only one children (which is leaf)



then we trim all those parents, got T' with n', m'

we have $n=n'$, $m=m'$.

we do this repeatly until we have condition 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?

   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.

The claim is true.

we prove it by contradiction:

Consider G_1 with n nodes is not connected. G_1 has separated connected graph G_1, G_2

since every node in G_1 has at least degree $\frac{n}{2}$.

for $\forall C \in G_1$, C is connected to $\frac{n}{2}$ nodes, we know G_1 must contain at least $\frac{n}{2}+1$ nodes.

similiarly, we have same conclusion in G_2.

then the sum of nodes of G_1, G_2 is $\frac{n}{2}+1+\frac{n}{2}+1 = n+2$

contradiction.

we have G_1 must be connected

7. *Kleinberg, Jon. Algorithm Design (p. 110, q. 9).* There's a natural intuition that two nodes that are far apart in a communication network—separated by many hops—have a more tenuous connection than two nodes that are close together. There are a number of algorithmic results that are based to some extent on different ways of making this notion precise. Here's one that involves the susceptibility of paths to the deletion of nodes.

   Suppose that an n-node undirected graph $G = (V, E)$ contains two nodes s and t such that the distance between s and t is strictly greater than $\frac{n}{2}$.

   (a) Show that there must exist some node v, not equal to either s or t, such that deleting v from G destroys all $s-t$ paths. (In other words, the graph obtained from G by deleting v contains no path from s to t.)

we prove by contradiction.

Consider there exist another node u. s.t. the path connect s, u and u, t do not involved v

we denote the path from s to t go through v is path 1

the path go through u is path 2.

Since the distance strictly larger than $\frac{n}{2}$.

the sum of length of 2 path strictly larger than n

len 1 + len 2 $\geq n+1$, we need at least $n+1$ nodes to satisfy this property, contradiction! then we prove the result ∎

   (b) Give an algorithm with running time $O(m + n)$ to find such a node v.

Consider BFS from s and from t, then they will meet at v

① start BFS from s, we have queue Q_s

start BFS from t, we have Q_t

② u = dequeue(Q_s), explore the neighbors n_1 of u

v = dequeue(Q_t), explore the neighbors n_2 of v

where n_1, n_2 are sets,

if : $n_1 \cap n_2 \neq \phi$:

select $v \in n_1 \cap n_2$ is the outputted node

① - ② - ④

③

Coding Question

8. Implement depth first search in either C, C++, C#, Java, or Python. Be efficient and implement it in $O(n+m)$ time, where n is the number of graph nodes, and m is the number of graph edges. Remember to submit a makefile along with your code, just as with week 1's coding question.

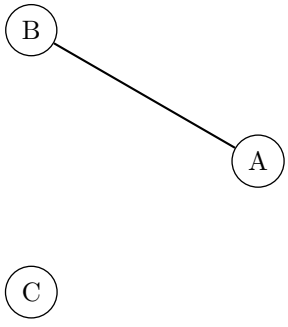
   Input format: the input will start with a positive integer, giving the number of instances that follow. For each instance, there will be a positive integer, giving the number of graph nodes. For each node there will be a line of space-delimited input. The first string in a line will be the node name. All following strings in a line will be the names of the adjacent nodes to the first node in the line.

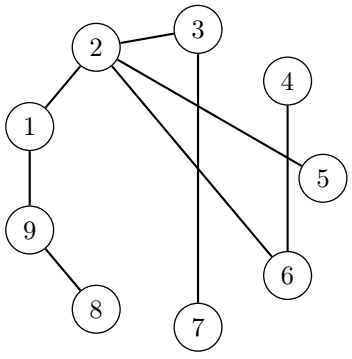
   You can assume the order the nodes will be listed is in increasing lexographic order (0-9, then A-Z, then a-z), both in each list of adjacent nodes, as well as the order nodes' lines are listed.

   A sample input is the following:

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





   Target output: for each instance, your program should output the names of nodes visited in depth first traversal of the graph, *with ties between nodes visiting the first node in lexicographic ordering.* Start your traversal with the first node in lexicographic order. Each instance's traversal should be on a separate line. Each output line should be terminated by a newline. The correct output to the sample input would be:

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