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: Kayley Seow Wisc id: kseow

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

   $f_2(n), f_3(n), f_4(n), f_5(n), f_6(n), f_8(n)$

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

   2 to the power of something → to ∞

   g_1, g_2, g_3, g_6 $g_4 < g_3$ - asymptotic limit graph

   $g_4 < g_5 < g_3$ $2^{\log n}, n\log^n, 2$ is smaller

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

   $\lim_{n\to\infty} \frac{n^{\log n}}{n^{4/3}} > 0$ - has Ω $fn \geq c \cdot g(n)$

   $\log(n(\log(n)))$ $\log(n^{\log n})$

   $\log n + \log(\log(n))$ $\log(n)\log(n)$

   $\log(n) = z$
   $z + \log(z)$ z^2

   $\sqrt{n} > \log(n)$
   $n^{\frac{1}{2}} > \log(n)$

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 $O(\log_2 g(n))$

$f(n)=O(g(n)) \rightarrow \exists n_0 > 0$
$\exists c \in \mathbb{R} > 0$

$\log_2 f(n) = \log_2(O(g(n)))$ $g n \geq 2$

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

$\log_2 f(n) \leq \log_2(c) + \log_2(g(n))$ - ran into problem b/c of addition, want $c_2 \log_2(g(n))$

$\log_2 f(n) \leq \log_2(c) + \log_2(g(n)) \leq \log_2(g(n))\,(c+1)$

Let $c^* = c+1$, $c^* \in \mathbb{R} > 0$ by closure

$\log_2 f(n) \leq \log_2(g(n)) \cdot c^*$ for $c^* \in \mathbb{R} > 0$

since we only care about $n \geq n_0$, $\log_2(f(n)) \geq 0$ at some point because $\log_2 f(n)$ is non-decreasing.

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

$2^x O($

Or $f(n) \leq c \cdot g(n)$ $\exists c > 0$

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

$f(n) > c \cdot g(n)$ disprove

$2^{f(n)} > c \cdot 2^{g(n)}$

$2^{\log_2 x^2} > c \cdot 2^{\log_2 x}$

$x^2 > cx$

$f(n) = \log_2 x^2$

$g(x) = \log_2 x$

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

$0 \leq f(n) \leq c \cdot g(n)$

$0 \leq f(n)^2 \leq c^2 \cdot g(n)^2$

by definition, because c is a constant, true by definition of O

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).

you need three loops to add and iterate
$O(n^3)$

(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.)

Because we have three loops no matter what - the outer n loop from 1 to n, the middle loop from i+1 to n, and adding all the entries from n-i as the inner loop - the runtime will always be n^3 at all times because there is no way to break out of each loop.

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

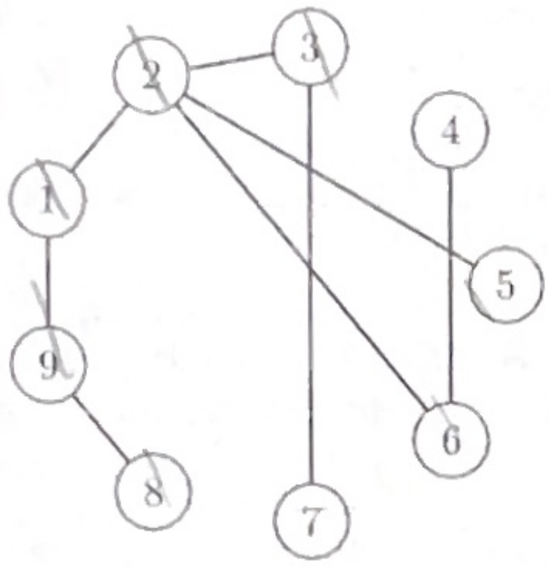
dynamic programming

$$\begin{bmatrix} 0 & x & x & x & x \\ 0 & 0 & x & x & x \\ 0 & 0 & 0 & x & x \\ 0 & 0 & 0 & 0 & x \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

```
for i=1 to n
    for j=i+1 to n
        if (i<j ){
            B(ij) = B(i,j-1) -A[j]
        }
        else { B(i,j) =A(i)}
    }
}
```

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: - done by distances
1, 9, 2, 8, 6, 5, 3, 4, 7

DFS: 1, 2, 3, 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.

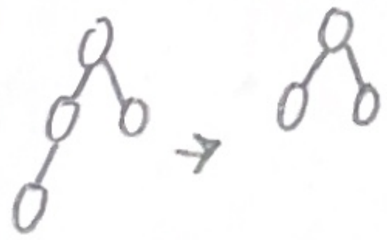
Base case: P(1) = 0 1 leaf no nodes with 2 children, 2 children is 1 less than leaves

Inductive Hypothesis: P(k), k=k-1 holds
Assume remove 2 leaf so that P(k) is there
It holds

Case 1: Removing one leaf from a two child node.

The number of leaves stays at 2, 1 nodes with 2 children, when you remove a leaf the # of nodes with 2 CH decreases, and the number of leaves decrease by 2.

Case 2: Removing one leaf from one child node



The number of leaves stays at 2, 1 node with 2 children, when you remove 2 leaf of child, the # of node with 2 children stays at 2. leaf count is 2, holds.

Therefore, by induction, proof holds.

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.

Proof by Contradiction

Assume that there exists a G that contains two connected components where each node of G is connected by at least n/2 nodes. Then, we will let K and J be two connected components such that they follow the property in this question of $|K|+|J| = n$

Because we want K to follow the property, there must be at least $\frac{n}{2}+1$ nodes, and with this each node will form a connection with the $\frac{n}{2}$ +1 nodes of K.

Because we want J to follow this too, the min. number of nodes in each subgraph is $\frac{n}{2}+1$.

This would mean that the total number of nodes in G is $(\frac{n}{2}+1)+(\frac{n}{2}+1)=$ n+2. Since the total number of node in G is n, there is a contradiction in our logic.

∴ We are proving by contradiction, our claim is true.