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: Ramapriya Ranganath Wisc id: rranganath

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

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

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

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:

$\exists\, C, N \quad f(n) \le g(n)$ 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 $O(2^{g(n)})$

Solution:

$\exists\, C, N \quad 0 \le f(n) \le C g(n)$ as $n \ge N$

for $n \ge N$ we have $2^{f(n)} \le (2^{g(n)})^C$

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

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

Solution:

$\exists\, C, N \cdot \quad 0 \le f(n) \le C g(n)$ as $n \ge N$

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

then $f^2(n) = O(g^2(n))$

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 $A[i] + A[i+1] + \ldots + A[j]$. (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
```

$\sum_{i=2}^{n} t_i$

$\sum_{i=2} t_i$

(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).

Solution: $O(n^2)$ [nested for loop]

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

(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.)

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

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

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

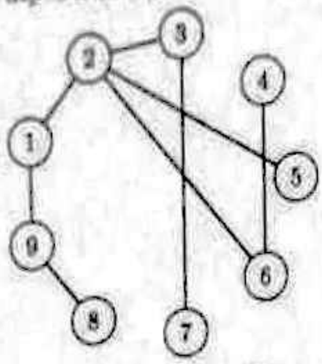
(c) Although the algorithm provided is the most natural way to solve the problem, it contains some highly unnecessary sources of inefficiency. Give a different algorithm to solve this problem, with an asymptotically better running time. In other words, you should design an algorithm with running time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

Solution: Merge sort $O(n \log n)$

```
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        left = arr[:mid]
        right = arr[mid:]
        merge_sort(left)
        merge_sort(right)
        i = j = k = 0
        while i < len(left) and j < len(right):
            if left[i] < right[j]:
                arr[k] = left[i]
                i += 1
            else:
                arr[k] = right[j]
                j += 1
            k += 1
        while i < len(left):
            arr[k] = left[i]
            i += 1
            k += 1
        while j < len(right):
            arr[k] = right[j]
            j += 1
            k += 1
```

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:

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.

Solution:

n = number of nodes with 2 children

m = number of leaves

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

For, $n \geq 1$ $m \geq 2$

(1) at least one parent has 2 children (special case full binary tree), we choose any one of such parent consider its node c then we trim c, we get T', with

$n' = n-1$, $m' = m-2+1 = m-1$

we have $n' = m' - 1$

$n = n'+1 = m'$ => $m = m'+1 = n+1$

2) All parents of leaves has only one children (which is leaf) then we train all those parents, got T' with n', m'

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

We do this continuously 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?

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: let the claim be true

We prove it by contradiction

Consider G with n nodes is not connected. G has separated connected graph G1, G2.

since every node in G has at least degree(n/2)

for $\forall$ C $\in$ G1, C is connected to (n/2) nodes, we know G1 must contain at least $(\frac{n}{2}) + 1$ nodes.

similarity we have same for G2.

the sum of G1 + G2 nodes = $\frac{n}{2} + 1 + \frac{n}{2} + 1 = n + 2$

Contradiction

we must have G connected.