• You can do hard things! Grades do matter, but not as much as you may think, but then life is uncertain anyway, so what.

• Don’t cheat. The consequence for cheating is far greater than the reward. Just try your best and you’ll be fine.

• Please read the entire exam before writing anything. There are 6 problems and most have multiple parts.

• This is a closed-book exam. At the end of the exam, you’ll find a multi-page cheat sheet. Do not tear out the cheat sheet! No outside material is allowed on this exam.

• You should write your answers legibly and in the space given for the question. Overly verbose answers will be penalized.

• Scratch paper is available on the back of the exam. Do not tear out the scratch paper! It messes with the auto-scanner.

• You have 75 minutes (1.25 hours) for the exam. Manage your time well. Do not spend too much time on questions you do not understand and focus on answering as much as you can!

• We know that this exam is shorter in length compared to the first one. Make sure you use the time well to think, be precise, and show as much work as possible.

Name: ________________________________

NetID: ________________________________

Date: ________________________________
Problem 1 [10 points]

For each of the following statements, answer if it is True or False. Use the table at the bottom to mark your choices.

i. Recursion is a special case of reduction that includes reducing the problem into smaller instances of itself.

ii. Merging two sorted arrays each of size $n$ into a single sorted array requires a minimum $O(n \log n)$ time.

iii. If topological sort exists for a graph then that graph has a cycle.

iv. If for some two nodes, the pre-post numbering intervals in DFS are disjoint then it means that the graph is disconnected.

v. Checking if a sequence is increasing or not can be achieved in a minimum of $O(n)$ time.

vi. The node with maximum post numbering in DFS is in a sink strongly connected component of the original graph.

vii. Every directed acyclic graph has either a source or a sink but not both.

viii. The asymptotic runtime of Merge Sort depends on the number of splits one makes of the original array.

ix. If a graph has a cycle then there is a back-edge in its DFS.

x. Decreasingly sorted pre numberings in DFS give a topological sort of the given directed acyclic graph.

<table>
<thead>
<tr>
<th>Table 1.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statement</td>
</tr>
<tr>
<td>i.</td>
</tr>
<tr>
<td>ii.</td>
</tr>
<tr>
<td>iii.</td>
</tr>
<tr>
<td>iv.</td>
</tr>
<tr>
<td>v.</td>
</tr>
<tr>
<td>vi.</td>
</tr>
<tr>
<td>vii.</td>
</tr>
<tr>
<td>viii.</td>
</tr>
<tr>
<td>ix.</td>
</tr>
<tr>
<td>x.</td>
</tr>
</tbody>
</table>
Problem 2 [15 points]

Solve the following recurrence relations exactly, i.e., obtain a closed form formula for $f(n)$ without order terms/bounds.

Useful formula: $\sum_{k=0}^{n} ar^k = \frac{a(1-r^{n+1})}{1-r}$.

a. $f(n) = f(n-1) + n2^n$, $n > 0$ and $f(0) = 3$.

b. $f(n) = 2f(n-1) + 1$, $n > 1$ and $f(1) = 1$.

c. $f(n) = 2f(\frac{n}{2}) + n$, $n > 1$ and $f(1) = 1$. 
Problem 3 [20 points]

Longest palindromic subsequence (LPS) of a sequence is defined as a subsequence of maximum length that is also a palindrome. For example, given the sequence BANANA, an LPS is ANANA and has length 5.

Write a dynamic programming algorithm to obtain the length of an LPS of a given sequence by providing the following.

- Recurrence and short English description (in terms of the parameters):

- Memoization data structure and evaluation order:

- Return value:

- Time Complexity:
Problem 4 [20 points]

Consider the graph in Figure 1.

Perform DFS starting from vertex 1 while breaking ties in the numeric order, i.e., the node with a smaller numeric label is visited first in tying situations and answer the following.

a. What is the DFS traversal order, i.e., the order in which you visit different vertices.

b. What are the DFS pre and post numberings of different vertices.

<table>
<thead>
<tr>
<th>Node</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre numbering</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Post numbering</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
c. Draw the DFS spanning tree.

d. Classify the non-tree edges into forward, backward, and cross edges. You may draw these non-tree edges in the DFS spanning tree using different edge-styles, colors, etc.

e. Using DFS or otherwise, obtain a topological sort of the given graph.
Problem 5 [20 points]

a. Consider the following recursive equation.

\[
A(u, v) = \min_{i=1}^{\min(u,v)} u A(u + v - i, i - 1), \ A(0, v) = v, \text{ and } A(u, 0) = u.
\]

Analyze the runtime of the memoized implementation of the above recursion to compute \(A(n, n)\). Show your work.

b. What is the relation between edit distance and longest common subsequence? Explain in detail.
c. Analyze the runtime of the median-of-median algorithm with group size as 3 instead of the usual 5.

d. Provide a logarithmic time algorithm to count the number of instances of a given number in a sorted list.
Problem 6 [15 points]

Consider the problem of multiplying $n, m$-digit numbers, $m \geq n$. One simple strategy to solve this problem is to use Karatsuba’s algorithm $n - 1$ times, i.e., multiply first number with the second number, then their product with the third number, and so on. Answer the following.

a. Analyze the runtime of the above simple strategy.

b. Provide a more time-efficient divide-and-conquer recursive algorithm to solve given problem.

c. Analyze the runtime of your algorithm.
This page is for additional scratch work!
1. Recursion

**Simple recursion**

- **Reduction**: solve one problem using the solution to another.
- **Recursion**: a special case of reduction - reduce problem to a smaller instance of itself (self-reduction).

**Definitions**

- Problem instance of size $n$ is reduced to one or more instances of size $n - 1$ or less.
- For termination, problem instances of small size are solved by some other method as base cases.

Arguably the most famous example of recursion. The goal is to move $n$ disks one at a time from the first peg to the last peg.

**Pseudocode: Tower of Hanoi**

```plaintext
Hanoi (n, src, dest, tmp):
    if (n > 0) then
        Hanoi (n-1, src, tmp, dest)
        Move disk from src to dest
        Hanoi (n-1, tmp, dest, src)
```

**Recurrences**

Suppose you have a recurrence of the form $T(n) = rT(n/c) + f(n)$.

The master theorem gives a good asymptotic estimate of the recurrence. If the work at each level is:

- **Decreasing**: $\frac{r}{c} f(n)$ where $\frac{r}{c} < 1$ $T(n) = O(f(n))$
- **Equal**: $f(n)$ $T(n) = O(n \log_c n)$
- **Increasing**: $\frac{c}{r} f(n)$ where $\frac{c}{r} > 1$ $T(n) = O(n^{\log_c r})$

Some useful identities:

- Sum of integers: $\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$
- Geometric series closed-form formula: $\sum_{k=0}^{n} a r^k = \frac{1 - a^{n+1}}{1-r}$
- Logarithmic identities: $\log(ab) = \log a + \log b$, $\log(a/b) = \log a - \log b$, $\log_a b = \frac{\log c}{\log_c a}$ ($a, b, c > 1$).

**Divide and conquer**

**Algorithm**

*Divide and conquer is an algorithm paradigm involving the decomposition of a problem into the same subproblem, solving them separately and combining their results to get a solution for the original problem.***

**Space**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mergesort</td>
<td>$O(n \log n)$</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>Quicksort</td>
<td>$O(n \log n)$ if using MoM</td>
<td>$O(n)$</td>
</tr>
</tbody>
</table>

We can divide and conquer multiplication like so:

$$bc = 10^n b_L c_L + 10^{n/2} (b_L c_R + b_R c_L) + b_R c_R.$$  

We can rewrite the equation as:

$$bc = b(x) c(x) = (b_L x + b_R) (c_L x + c_R) = (b_L c_L) x^2 + (b_L + b_R)(c_L + c_R) - b_L c_L - b_R c_R x + b_R c_R.$$  

Its running time is $O(n \log n) = O(n^{1.585})$.

**Linear time selection**

The median of medians (MoM) algorithms give a element that is larger than $\frac{1}{3}$’s and smaller than $\frac{2}{3}$’s of the array elements. This is used in the linear time selection algorithm to find element of rank $k$.

**Pseudocode: Quicksort with median of medians**

```plaintext
Median-of-medians (A, d, l)
sublists = [A[j:j+d-1] for j in range(l, len(A), d)]
medians = sorted(sublist[sublist_len//2] for sublist in sublists)

// Base case
if len(A) <= 5 return sorted(A)

// Find median of medians
if len(medians) <= 5
    pivot = sorted(medians[len(medians)//2])
ext else
    pivot = Median-of-medians(medians, len(medians)//2)

// Partitioning step
low = [j for j in A if j < pivot]
high = [j for j in A if j > pivot]
k = len(low)
if k < l
    return Median-of-medians(low, l)
ext else if k < (l+d-1)
    return Median-of-medians(low, l-1)
ext else
    return pivot
```

**Backtracking**

Backtracking is the algorithm paradigm involving guessing the solution to a single step in some multi-step process and recursing backwards if it doesn’t lead to a solution. For instance, consider the longest increasing subsequence (LIS) problem. You can either check all possible subsequences:

**Pseudocode: LIS - Naive enumeration**

```plaintext
algLISNaive(A[1..n]):
    maxmax = 0
    for each subsequence $B$ of $A$ do
        if $B$ is increasing and $|B| > maxmax$
            maxmax = |B|
    return maxmax
```

On the other hand, we don’t need to generate every subsequence; we only need to generate the subsequences that are increasing:

**Pseudocode: LIS - Backtracking**

```plaintext
LIS_smaller(A[1..n], x):
    if $n = 0$ then return 0
    max = LIS_smaller(A[1..n-1], x)
    if $A[n] < x$ then
        max = max (max, 1 + LIS_smaller(A[1..(n-1)], A[n]))
    return max
```

### Problem Set

1. **Problem 1:**

   Suppose you have a recurrence of the form $T(n) = rT(n/c) + f(n)$.

   The master theorem gives a good asymptotic estimate of the recurrence. If the work at each level is:

   - **Decreasing**: $\frac{r}{c} f(n)$ where $\frac{r}{c} < 1$ $T(n) = O(f(n))$
   - **Equal**: $f(n)$ $T(n) = O(n \log_c n)$
   - **Increasing**: $\frac{c}{r} f(n)$ where $\frac{c}{r} > 1$ $T(n) = O(n^{\log_c r})$

   Some useful identities:

   - Sum of integers: $\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$
   - Geometric series closed-form formula: $\sum_{k=0}^{n} a r^k = \frac{1 - a^{n+1}}{1-r}$
   - Logarithmic identities: $\log(ab) = \log a + \log b$, $\log(a/b) = \log a - \log b$, $\log_a b = \frac{\log c}{\log_c a}$ ($a, b, c > 1$).

2. **Problem 2:**

   **Divide and conquer**

   Divide and conquer is an algorithm paradigm involving the decomposition of a problem into the same subproblem, solving them separately and combining their results to get a solution for the original problem.

   **Space**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mergesort</td>
<td>$O(n \log n)$</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>Quicksort</td>
<td>$O(n \log n)$ if using MoM</td>
<td>$O(n)$</td>
</tr>
</tbody>
</table>

   We can divide and conquer multiplication like so:

   $$bc = 10^n b_L c_L + 10^{n/2} (b_L c_R + b_R c_L) + b_R c_R.$$  

   We can rewrite the equation as:

   $$bc = b(x) c(x) = (b_L x + b_R) (c_L x + c_R) = (b_L c_L) x^2 + (b_L + b_R)(c_L + c_R) - b_L c_L - b_R c_R x + b_R c_R.$$  

   Its running time is $O(n \log n) = O(n^{1.585})$.

3. **Problem 3:**

   **Linear time selection**

   The median of medians (MoM) algorithms give an element that is larger than $\frac{1}{3}$’s and smaller than $\frac{2}{3}$’s of the array elements. This is used in the linear time selection algorithm to find element of rank $k$.

   **Pseudocode: Quicksort with median of medians**

   ```plaintext
   Median-of-medians (A, d, l)
   sublists = [A[j:j+d-1] for j in range(l, len(A), d)]
   medians = sorted(sublist[sublist_len//2] for sublist in sublists)

   // Base case
   if len(A) <= 5 return sorted(A)

   // Find median of medians
   if len(medians) <= 5
       pivot = sorted(medians[len(medians)//2])
   else
       pivot = Median-of-medians(medians, len(medians)//2)

   // Partitioning step
   low = [j for j in A if j < pivot]
   high = [j for j in A if j > pivot]
   k = len(low)
   if k < l
       return Median-of-medians(low, l)
   else if k < (l+d-1)
       return Median-of-medians(low, l-1)
   else
       return pivot
   ```

4. **Problem 4:**

   **Backtracking**

   Backtracking is the algorithm paradigm involving guessing the solution to a single step in some multi-step process and recursing backwards if it doesn’t lead to a solution. For instance, consider the longest increasing subsequence (LIS) problem. You can either check all possible subsequences:

   **Pseudocode: LIS - Naive enumeration**

   ```plaintext
   algLISNaive(A[1..n]):
       maxmax = 0
       for each subsequence $B$ of $A$ do
           if $B$ is increasing and $|B| > maxmax$
               maxmax = |B|
       return maxmax
   ```

   On the other hand, we don’t need to generate every subsequence; we only need to generate the subsequences that are increasing:

   **Pseudocode: LIS - Backtracking**

   ```plaintext
   LIS_smaller(A[1..n], x):
       if $n = 0$ then return 0
       max = LIS_smaller(A[1..n-1], x)
       if $A[n] < x$ then
           max = max (max, 1 + LIS_smaller(A[1..(n-1)], A[n]))
       return max
   ```
Dynamic programming

Dynamic programming (DP) is the algorithm paradigm involving the computation of a recursive backtracking algorithm iteratively to avoid the recomputation of any particular subproblem.

Longest increasing subsequence

The longest increasing subsequence problem asks for the length of a longest increasing subsequence in an unordered sequence, where the sequence is assumed to be given as an array. The recurrence can be written as:

\[
LIS(i,j) = \begin{cases} 
0 & \text{if } i = 0 \\
LIS(i-1,j) & \text{if } A[i] < A[j] \\
\max \{LIS(i-1,j), 1 + LIS(i-1,i)\} & \text{else}
\end{cases}
\]

Pseudocode: LIS - DP

\textbf{LIS-Iterative}(A[1..n]):
\textit{A}[n+1] = \infty
\textbf{for} j \leftarrow 0 \textbf{to} n \\
\textbf{if} A[j] \leq A[i] \textbf{then} LIS[0][j] = 1
\textbf{for} i \leftarrow 1 \textbf{to} n - 1 \\
\textbf{for} j \leftarrow i \textbf{to} n - 1 \\
\textbf{if} A[i] \geq A[j] \\
LIS[i,j] = LIS[i-1,j] \\
\textbf{else} \\
LIS[i,j] = \max \{LIS[i-1,j], 1 + LIS[i-1,i]\}
\textbf{return} LIS[n,n+1]

Edit distance

The edit distance problem asks how many edits we need to make to a sequence for it to become another one. The recurrence is given as:

\[
\text{Opt}(i,j) = \begin{cases} 
\alpha_{x,y} + \text{Opt}(i-1,j-1), & \text{if } A[i] = B[j] \\
\delta + \text{Opt}(i-1,j), & \text{if } A[i] \neq B[j] \\
\delta + \text{Opt}(i,j-1)
\end{cases}
\]

Base cases: Opt(0,0) = 0 and Opt(0,j) = \delta \cdot j

Pseudocode: Edit distance - DP

\textbf{EDIST}(A[1..m], B[1..n]):
\textbf{for} i \leftarrow 1 \textbf{to} m \textbf{do} M[i,0] = i\delta
\textbf{for} j \leftarrow 1 \textbf{to} n \textbf{do} M[0,j] = j\delta
\textbf{for} i \leftarrow 1 \textbf{to} m \textbf{do}
\textbf{for} j \leftarrow 1 \textbf{to} n \textbf{do}
\begin{align*}
M[i,j] &= \min \{ \text{COST}[A[i]] \cdot [B[j]] + M[i-1][j-1], \\
& \delta + M[i][j-1], \\
& \delta + M[i-1][j] \}
\end{align*}

2 Graph algorithms

Graph basics

A graph is defined by a tuple \( G = (V,E) \) and we typically define \( n = |V| \) and \( m = |E| \). We define \((u,v)\) as the edge from \( u \) to \( v \). Graphs can be represented as adjacency lists or adjacency matrices though the former is more commonly used.

- **path**: sequence of distinct vertices \( v_1, v_2, \ldots, v_k \) such that \((v_i,v_{i+1}) \in E \) for \( 1 \leq i \leq k-1 \). The length of the path is \( k - 1 \) (the number of edges in the path).
  - Note: a single vertex \( u \) is a path of length 0.
- **cycle**: sequence of distinct vertices \( v_1, v_2, \ldots, v_k \) such that \((v_i,v_{i+1}) \in E \) for \( 1 \leq i \leq k-1 \) and \((v_k,v_1) \in E \). A single vertex is not a cycle according to this definition.
  - **Caveat**: Sometimes people use the term cycle to also allow vertices to be repeated. We will use the term **tour**.
- A vertex \( u \) is connected to \( v \) if there is a path from \( u \) to \( v \).
- The **connected component** of \( u \) \( \text{con}(u) \), is the set of all vertices connected to \( u \).
- A vertex \( u \) can **reach** \( v \) if there is a path from \( u \) to \( v \). Alternatively \( v \) can be reached from \( u \). Let \( \text{rch}(u) \) be the set of all vertices reachable from \( u \).
Directed acyclic graphs

Directed acyclic graphs (dags) have an intrinsic ordering of the vertices that enables dynamic programming algorithms to be used on them. A topological ordering of a dag \( G = (V, E) \) is an ordering \( \prec \) on \( V \) such that if \( (u, v) \in E \) then \( u \prec v \).

**Kahn’s algorithm**

```pseudocode
Kahn(G(V, E)): toposort ← empty list
for v ∈ V do
    in[v] ← \{ u | (u, v) ∈ E \}
while V ≠ empty do
    v ← vertex in V that has in[v] = 0
    Add v to toposort
    Remove v from V
    for all u ∈ v do
        in[u] ← in[u] - 1
return toposort
```

Running time: \( O(n+m) \)

- A dag may have multiple topological sorts.
- A topological sort can be computed by DFS, in particular by listing the vertices in decreasing post-visit order.

**DFS and BFS**

**Pseudocode: Explore (DFS/BFS)**

```pseudocode
Explore(G, u):
    for i ← 1 to n do
        Visited[i] ← False
        Add u to ToExplore and to S
        Visited[u] ← True
        Make tree T with root as u
    while B is non-empty do
        Remove node x from B
        for each edge (x, y) in Adj(x) do
            if Visited[y] = False
                Visited[y] ← True
                Add y to B, S, T (with x as parent)
```

**Pre/post numbering**

- Pre and post numbering aids in analyzing the graph structure. By looking at the numbering we can tell if a edge \( (u, v) \) is a:
  - Forward edge: \( \text{pre}(u) < \text{pre}(v) < \text{post}(v) < \text{post}(u) \)
  - Backward edge: \( \text{pre}(u) < \text{pre}(v) < \text{post}(u) < \text{post}(v) \)
  - Cross edge: \( \text{pre}(u) < \text{post}(u) < \text{pre}(v) < \text{post}(v) \)

**Strongly connected components**

- A maximal group of vertices that are all strongly connected to one another is called a strong component.

**Metagraph**

```pseudocode
Metagraph(G, V, E):
    for i ∈ V do
        for j ∈ V do
            d(i, j, 0) ← ℓ(i, j)
        for k ← 0 to n - 1 do
            for i ∈ V do
                for j ∈ V do
                    d(i, j, k) ← min(d(i, j, k - 1), d(i, k, k - 1) + d(k, j, k - 1))
            for i ∈ V do
                if d(i, i, n - 1) < 0 then
                    return “exists a negative cycle in G
    return d
```

Running time: \( O(n^3) \)

**Shortest paths**

**Dijkstra’s algorithm:**

Find minimum distance from vertex \( s \) to all other vertices in graphs without negative weights.

**Pseudocode: Dijkstra**

```pseudocode
for v ∈ V do
    d(v) ← ∞
    X ← ∅
    d(s, s) ← 0
for i ← 1 to n do
    v ← arg min_{u∈V\setminus X} d(u)
    X ← X ∪ {v}
    for u in Adj(v) do
        d(u) ← min(d(u), d(v) + ℓ(u, v))
return d
```

Running time: \( O(nm) \) (if using a Fibonacci heap as the priority queue)

**Bellman-Ford algorithm:**

Find minimum distance from vertex \( s \) to all other vertices in graphs without negative cycles. It is a DP algorithm with the following recurrence:

\[ d(v, k) = \begin{cases} 0 & \text{if } v = s \text{ and } k = 0 \\ \min_{u \in V} \{ d(u, k - 1) + \ell(u, v) \} & \text{else} \end{cases} \]

**Pseudocode: Bellman-Ford**

```pseudocode
for each v ∈ V do
    d(v) ← ∞
    d(s) ← 0
for k ← 1 to n - 1 do
    for each edge (u, v) in E do
        d(v) ← min{d(v), d(u) + ℓ(u, v)}
return d
```

Base cases: \( d(s, 0) = 0 \) and \( d(v, 0) = \infty \) for all \( v \neq s \).

**Floyd-Warshall algorithm:**

Find minimum distance from every vertex to every vertex in a graph without negative cycles. It is a DP algorithm with the following recurrence:

\[ d(i, j, k) = \begin{cases} 0 & \text{if } i = j \\ \min_{(i, j) \notin E \text{ and } k = 0} \{ d(i, j, k - 1) \} & \text{else} \end{cases} \]

Then \( d(i, j, n - 1) \) will give the shortest-path distance from \( i \) to \( j \).

**Pseudocode: Floyd-Warshall**

```pseudocode
for i ∈ V do
    for j ∈ V do
        ℓ(i, j) ← ∞ if (i, j) ∉ E, 0 if i = j
for k ← 0 to n - 1 do
    for i ∈ V do
        for j ∈ V do
            d(i, j, k) ← min\{d(i, j, k - 1), d(i, k, k - 1) + d(k, j, k - 1)\}
for i ∈ V do
    if d(i, i, n - 1) < 0 then
        return “exists a negative cycle in G
return d