CSCE 411: Design and Analysis of Algorithms, Fall 2025

Test 1, Section 500

Name: _____

Instructions: DO NOT OPEN THIS EXAM UNTIL YOU ARE TOLD TO.

Write your name at the top of this page. Read the explanation of the test format, and the academic integrity statement, and sign at the bottom of the page. WHEN TAKING THE EXAM, WRITE YOUR INITIALS AT THE TOP OF EACH PAGE.

Test Format: The test consists of

- 11 multiple choice problems (5 points each)
- 3 free response problems

The total number of points on the exam is 100.

Academic Integrity: On my honor, I will complete the exam without giving or receiving help from anyone else, and without consulting or using any resources other than *my own* single note sheet that I am allowed as a part of the exam.

Signed: _____

Multiple Choice Questions

Instructions: Circle a single answer for each multiple choice problem.

1. (5 points) Suppose X is a string of length n and Y is a string of length m. The runtime of solving the Longest Common Subsequence (LCS) problem on X and Y using dynamic programming is:

   (a) $\Theta(m+n)$
   (b) $\Theta(mn)$
   (c) $\Theta(m^2n^2)$
   (d) $\Theta(2^m+2^n)$
   (e) $\Theta(2^{m+n})$

2. (5 points) Which of the following sets of codewords for symbols $\{a, b, c, d\}$ is a **prefix code**?

   (a) $\{a:0,\ b:01,\ c:10,\ d:110\}$
   (b) $\{a:00,\ b:01,\ c:10,\ d:11\}$
   (c) $\{a:100,\ b:101,\ c:1,\ d:110\}$
   (d) $\{a:01,\ b:10,\ c:011,\ d:000\}$
   (e) $\{a:11,\ b:111,\ c:10,\ d:110\}$

3. (5 points) What is the key idea behind amortized analysis?

   (a) To find the worst-case runtime of a single operation in a sequence of operations.
   (b) To average the cost of operations over a sequence of operations.
   (c) To find the best-case runtime of a single operation in a sequence of operations.
   (d) To ensure every operation has constant time complexity.
   (e) To minimize space complexity.

4. (5 points) Which of the following problems is most naturally solved using a dynamic programming approach?

   (a) Sorting an array
   (b) Matrix chain multiplication
   (c) Activity selection
   (d) Fractional knapsack
   (e) Binary search

5. (5 points) What is the best time complexity $T(n)$ of a procedure that satisfies the base case $T(2) = T(1) = O(1)$ and the recurrence $T(n) = T(n-2) + O(n)$?

   (a) $O(n^2)$
   (b) $O(n \log n)$
   (c) $O(n)$
   (d) $O(\log^2 n)$
   (e) $O(\log n)$

6. (5 points) In the potential method for amortized analysis, which condition must the potential function Φ satisfy for all states D_i of the data structure?

   (a) $\Phi(D_i) \geq \Phi(D_0)$
   (b) $\Phi(D_i)$ is strictly increasing
   (c) $\Phi(D_i) \leq 0$
   (d) $\Phi(D_i)$ is constant
   (e) $\Phi(D_i) = 0$ for all i

7. (5 points) Suppose you develop an optimized method to multiply 7×7 matrices using exactly 100 multiplications and 200 additions. Using this method, you design a divide-and-conquer algorithm for multiplying two $n \times n$ matrices where $n = 7^p$ for $p \in \mathbb{N}$, by dividing the matrices into square 7×7 blocks. What recurrence relationship represents the runtime $T(n)$?

   (a) $T(n) = 100T(n/7) + 200$
   (b) $T(n) = 200T(n/7) + 100$
   (c) $T(n) = 100T(n/7) + \Theta(n^2)$
   (d) $T(n) = 200T(n/7) + \Theta(n^2)$
   (e) $T(n) = 200T(n/49) + \Theta(n^2)$

8. (5 points) You are designing a dynamic programming algorithm to solve the **weighted interval scheduling problem**: given n intervals, each with a start time, end time, and weight, find the subset of non-overlapping intervals with maximum total weight. Assume that the intervals are sorted in increasing order of end times.

   - Let $\text{OPT}(j)$ be a non-overlapping subset of the first j intervals with maximum total weight.
   - Let w_j be the weight of interval j.
   - Let $p(j)$ be the index of the last interval that does not overlap with interval j.

   HINT: This is just the maximum viewership problem from the homework, reworded.

   Which of the following recurrences correctly represents the optimal substructure for this problem?

   (a) $\text{OPT}(j) = \max\big(\text{OPT}(j-1), w_j + \text{OPT}(p(j))\big)$
   (b) $\text{OPT}(j) = w_j + \text{OPT}(j-1)$
   (c) $\text{OPT}(j) = \max\big(\text{OPT}(j-1), w_j + \text{OPT}(j-1)\big)$
   (d) $\text{OPT}(j) = \min\big(\text{OPT}(j-1), w_j + \text{OPT}(p(j))\big)$
   (e) $\text{OPT}(j) = \sum_{i=1}^{j} w_i$ if intervals i are non-overlapping

9. (5 points) Which of the following is NOT a fundamental step in the divide and conquer paradigm?

   (a) Dividing the problem into subproblems
   (b) Recursively solving the subproblems
   (c) Merging the solutions of the subproblems
   (d) Making a locally optimal choice at each step
   (e) All of the above are fundamental steps

10. (5 points) You are given a set of activities, each with a start time and a finish time. Your goal is to schedule as many non-overlapping activities as possible. Which algorithmic approach is best suited to solve this problem?

    (a) Dynamic programming
    (b) Greedy algorithm
    (c) Divide and conquer
    (d) Backtracking
    (e) Recursion

11. (5 points) Consider a dynamic array that starts with an initial capacity of 1 and doubles in size whenever it becomes full. During a sequence of n insertions, most insertions take constant time, but whenever the array reaches its capacity, a resizing occurs, requiring all existing elements to be copied to a new array. Given this behavior, what is the best upper bound on the total time of an entire sequence of n insertions?

    (a) $O(1)$
    (b) $O(\log n)$
    (c) $O(n)$
    (d) $O(n \log n)$
    (e) $O(n^2)$

Long Answer Questions

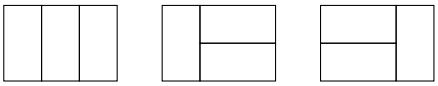
1. (15 points) Consider a $2 \times N$ rectangular strip, where N is a non-negative integer. You are asked to tile the strip using only 1×2 tiles, i.e., all spaces in the strip must be covered and no tiles are overlapping. Tiles can be placed either:

   - Vertically, covering two cells in a single column, or
   - Horizontally, covering two adjacent cells in a single row.

   Determine the number of distinct ways to completely tile a $2 \times N$ strip using these tiles.

   Example: If $N = 3$, the possible tilings are:

   - Three vertical tiles in each column.
   - One vertical tile in the first column, and two horizontal tiles covering columns 2–3 in the top and bottom rows.
   - Two horizontal tiles covering columns 1-2 in the top and bottom rows, and one vertical tile in the last column.

   Tiling 1 Tiling 2 Tiling 3

   Thus, for $N = 3$, there are 3 distinct ways to tile the strip, i.e., $T(3) = 3$. We also have the convention $T(0) = 1$.

   (a) (5 points) Let $T(N)$ be the number of distinct tilings of a $2 \times N$ strip. Write the recurrence relationship for $T(N)$, including the base cases.

   (b) (5 points) What is the runtime of an asymptotically optimal dynamic programming algorithm to compute $T(N)$ for a given $N \geq 0$?

   (c) (5 points) Compute $T(10)$, the number of ways to completely tile a 2×10 strip.

2. (10 points) **Customer waiting time minimization.**
You are given a list of n customers, each with a service time s_i. The waiting time for a customer is defined as the sum of the service times of all customers served before them. Your goal is to determine an optimal order in which to serve the customers so as to minimize the total waiting time.

Example: Suppose $n = 3$, $s_1 = 10$, $s_2 = 4$, $s_3 = 7$ and the order of serving the customers is customer 3, followed by customer 2, followed by customer 1. Then:

- Customer 3 must wait 0 time (service begins immediately).
- Customer 2 must wait 7 time (the service time of customer 3).
- Customer 1 must wait 11 time (the service times of customer 3 and customer 2)

Hence, the total waiting time is $0 + 7 + 11 = 18$.

(a) (5 points) Clearly describe the greedy algorithm that minimizes the total waiting time. **Do not use pseudocode for this part.**

(b) (5 points) What is the asymptotic runtime of an optimal greedy algorithm that minimizes the total waiting time?

3. (20 points) **Counting inversions.**
An **inversion** in an array $A[1:n]$ is a pair of indices (i, j) such that $i < j$ but $A[i] > A[j]$. For example, the array $[3, 1, 2]$ has two inversions: $(3, 1)$ and $(3, 2)$. In the inversion counting problem, the input is the array A and the goal is to count the number of inversions in A.

   (a) (5 points) Suppose you are merging two **sorted** subarrays $A[L:M]$ and $A[M+1:R]$. Fill in the missing parts of the pseudocode below to count all inversions where the first element is in the left subarray and the second is in the right subarray.

Algorithm 1 MergeAndCount(A, L, M, R)

```
1:  i ← L, j ← M + 1, k ← L
2:  Initialize auxiliary array B of length L − R + 1        ▷ \\ NOTE: B is used to sort A
3:  count ← 0
4:  while i ≤ M and j ≤ R do
5:      if ------(i)------ then
6:          B[k] ← A[i]
7:          i ← i + 1
8:      else
9:          B[k] ← A[j]
10:         count ← count + ------(ii)------
11:         j ← j + 1
12:     end if
13:     k ← k + 1
14: end while
15: Copy any remaining elements of A[L : M] and A[M + 1 : R] into B
16: Copy B[L : R] back into A[L : R]        ▷ \\ NOTE: This step will cause A to be sorted
17: return count
```

Fill in the two blanks missing from the pseudocode above in the space provided below:

(i) (2 points) The condition to take the element from the left subarray.

(ii) (3 points) The amount by which to increase the inversion count when taking an element from the right subarray.

   (b) (5 points) What is the running time of this merge step MergeAndCount(A, L, M, R)?

Algorithm 2 CountInversions(A, L, R)

```
1: if _____(i)_____ then                          ▷ \\ HINT: Base case for divide and conquer
2:     return _____(ii)_____
3: end if
4: M ← _____(iii)_____                            ▷ \\ HINT: Define subproblems for divide and conquer
5: count_left ← _____(iv)_____
6: count_right ← _____(v)_____
7: count_split ← MergeAndCount(A, L, M, R)        ▷ \\ NOTE: Combine step
8: return count_left + count_right + count_split
```

(c) (5 points) The above incomplete pseudocode implements an optimal runtime algorithm for counting inversions based on divide-and-conquer, modeled after merge sort. Fill in each numbered blank above in the space provided below.

(i)

(ii)

(iii)

(iv)

(v)

(d) (5 points) State the recurrence relation that characterizes the runtime of the (optimal) algorithm. Then, solve the recurrence to obtain the asymptotic runtime.

Divide-and-conquer. 3 steps: (1) Divide, (2) Conquer, and (3) Combine. Runtimes satisfy a recurrence relationship, which can be turned into an actual runtime using a few different techniques.

Dynamic programming. For problems with optimal substructure and overlapping subproblems. Solves overlapping subproblems only once each using either a top-down or a bottom-up approach.

Greedy algorithms. Make a decision at each step that is "locally" the best choice. Proving a greedy algorithm is optimal for a certain optimization problem typically involves proving that making a greedy choice at the first step is "safe."

Accounting and potential method. Similarity: both store "credit" and "pay" ahead of time. The accounting method stores credit in individual steps. The potential method stores credit as "potential" in a data structure D_i. For accounting method, define $\hat{c}_i$ and prove that $\sum_i c_i \leq \sum_i \hat{c}_i$. For potential method, choose D_i and potential function Φ, and then $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$ is given; proving $\Phi(D_i) \geq \Phi(D_0)$ guarantees $\sum_i c_i \leq \sum_i \hat{c}_i$. For both accounting and potential, must bound $\sum_i \hat{c}_i$ to prove runtime guarantee.

Master theorem. Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the relation $T(n) = aT(n/b) + f(n)$.

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.

Strassen's algorithm. Relies on knowing how to multiply 2×2 matrices using a small number of additions and 7 multiplications. Uses this to multiply two matrices of size $n \times n$ (where $n =$ power of 2) by breaking them into blocks and recursively calling a matrix-matrix multiplication function 7 times. Has recurrence relationships $T(n) = 7T(n/2) + \Theta(n^2)$.

Matrix multiplication problem. For $i = 1, 2, \ldots n$, let A_i be a matrix of size $p_{i-1} \times p_i$. Find the way to parenthesize the matrix chain $A_1 A_2 \cdots A_n$ so that the total computational cost is minimized. It takes $\Theta(pqr)$ operations to multiply AB if A has size $p \times q$ and B has size $q \times r$.

The activity selection problem. Let $(a_1, a_2, a_3, \ldots a_n)$ be activities with distinct start and finish times (s_i, f_i) for $i = 1, 2, \ldots, n$, ordered so that $f_1 < f_2 < \ldots < f_n$. Find the largest set of non-overlapping activities.

Multipop stack. Has operations PUSH(S, x) (pushes x onto S), POP(S) (pops top element off), and MULTIPOP(S, k) (pops $\min\{|S|, k\}$ elements). Running n total operations always has $O(n)$ runtime; key idea is that you can't pop an element until you've pushed it.

The optimal prefix code problem. Given an alphabet C and a frequency $c.freq$ for each $c \in C$ in a given string s, find the prefix code that represents s using a minimum number of bits. A prefix code associated each character with a binary codeword, such that the codeword for one character is never the start of a codeword for another character. A prefix code can be associated with a binary tree in which each character is associated with a leaf of the tree.

Binary counter. Stores an integer in binary using a $\{0,1\}$ array A. Incrementing A updates the binary number stored in A to represent the next integer. Cost is given in terms of number of bits flipped. $A = [0101]$ represents $0 \times 2^0 + 1 \times 2^1 + 0 \times 2^2 + 1 \times 2^3 = 2 + 8 = 10$.

BottomUpMatrixChainMultiplication($\mathbf{p} = [p_0, p_1, \ldots, p_n]$)
  $n = \text{length}(\mathbf{p}) - 1$
  Let $m[1 \ldots n][1 \ldots n]$ be an empty $n \times n$ array
  for $i = 1$ to n **do**
    $m[i, i] = 0$
  end for
  for $\ell = 2$ to $n - 1$ **do**
    for $i = 1$ to $n - \ell - 1$ **do**
      $j = i + \ell - 1$
      $m[i, j] = \infty$
      for $k = i$ to $j - 1$ **do**
        $q = m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j$
        if $q < m[i, j]$ **then**
          $m[i, j] = q$
        end if
      end for
    end for
  end for
  Return $m[1, n]$

GreedyCoinChange(C, $v = [v_1, v_2, \ldots, v_n = 1]$)
  $n = \text{length}(v)$
  Let $m = [0..0]$ be an empty array of n zeros
  for $i = 1$ to n **do**
    while $C \geq v_i$ **do**
      $m[i] = m[i] + 1$
      $C = C - v_i$
    end while
  end for
  Return m

Assorted Reminders

- $\sum_{i=0}^{n} \frac{1}{2^i} \leq 2$
- $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$