**Midterm Exam 1: Data Structures & Algorithms CIS350 2025**

**Instructions:**

* Total Points: 100  
  Write your answers clearly for each section. For pseudocode, follow standard algorithm writing conventions as used in your textbook.
* Submit your exam on the specified due date and time.
* All answers must be provided in one MS Word or in one PDF file uploaded to the Midterm 1 drop box

**Section 1: Multiple Choice Questions (50 points, 1 point per question)**

**Question 1:**

Which of the following time complexities grow the fastest as n increases?

a) O(log n)  
b) O(n)  
c) O(n log n)  
**d) O(2^n)**

**Question 2:**

Which of the following is **true** about recursion?

**a) Every recursive function can be converted to an iterative function.**  
b) A recursive function always executes faster than an iterative function.  
c) Recursive functions are always easier to understand than iterative functions.  
d) Every recursive function leads to infinite recursion.

**Question 3:**

What is the time complexity for accessing an element in an array by index?

**a) O(1)**  
b) O(n)  
c) O(log n)  
d) O(n^2)

**Question 4:**

Which sorting algorithm has the best average-case time complexity?

a) Bubble Sort  
**b) Quick Sort**  
c) Insertion Sort  
d) Selection Sort

**Question 5:**

Which algorithm has a time complexity of O(n log n) in the worst case?

**a) Merge Sort**  
b) Bubble Sort  
c) Insertion Sort  
d) Linear Search

**Question 6:**

Given two functions f(n) = 100n and g(n) = n^2, which of the following is **true**?

a) f(n) grows faster than g(n).  
**b) g(n) grows faster than f(n).**  
c) Both grow at the same rate.  
d) There is not enough information to determine.

**Question 7:**

What is the Big-O time complexity for finding the minimum element in an unsorted array of size n?

a) O(1)  
b) O(log n)  
**c) O(n)**  
d) O(n^2)

**Question 8:**

In a recursive algorithm that sums the elements of an array of size n, what is the time complexity?

a) O(1)  
b) O(log n)  
**c) O(n)**  
d) O(n^2)

**Question 9:**

What is the time complexity of an algorithm that runs a loop inside a loop, where both loops iterate n times?

a) O(n)  
b) O(log n)  
c) O(n log n)  
**d) O(n^2)**

**Question 10:**

Which of the following conditions is necessary for a recursive function to terminate?

a) Infinite recursion  
**b) Base case**  
c) Recursion step  
d) Stack overflow

**Question 11:**

What is the time complexity for inserting an element at the end of a dynamic array?

**a) O(1) amortized**  
b) O(n)  
c) O(log n)  
d) O(n^2)

**Question 12:**

Consider a recursive function that calls itself twice on inputs n-1 and n-2. What is the time complexity of this function?

a) O(n)  
b) O(n^2)  
**c) O(2^n)**  
d) O(log n)

**Question 13:**

If an algorithm’s time complexity is O(n log n), which of the following statements is true as n increases?  
a) The algorithm will perform better than O(n^2).  
b) The algorithm will perform worse than O(n).  
**c) Both a and b are correct.**  
d) None of the above.

**Question 14:**

A recursive algorithm has a time complexity of O(n!). For small inputs (n < 5), this algorithm performs:  
a) Faster than O(n^2)  
b) Faster than O(n log n)  
**c) Slower than O(n)**  
d) All of the above

**Question 15:**

What is the time complexity to access the middle element of an unsorted array of size n?  
a) O(1)  
**b) O(n)**  
c) O(log n)  
d) O(n^2)

**Question 16:**

What is the time complexity of an algorithm that performs a binary search on a sorted array of size n?  
a) O(1)  
b) O(n)  
**c) O(log n)**  
d) O(n^2)

**Question 17:**

Which of the following recursive functions correctly computes the sum of the first n natural numbers?  
a) sum(n) = n \* sum(n - 1)  
b) sum(n) = sum(n - 1)  
**c) sum(n) = n + sum(n - 1)**  
d) sum(n) = n + sum(n)

**Question 18:**

What is the time complexity of an algorithm that checks if an element exists in an **unsorted** array of size n?  
**a) O(n)**  
b) O(log n)  
c) O(1)  
d) O(n log n)

Consider the following pseudocode for a recursive function:

**Algorithm RecursiveFunction(n)**

**if n <= 1 then**

**return 1**

**else**

**return RecursiveFunction(n - 1) + RecursiveFunction(n - 2)**

**Question 19:**

What is the time complexity of this function?  
a) O(n)  
b) O(n^2)  
**c) O(2^n)**  
d) O(log n)

**Question 20:**

What is the primary purpose of exception handling ?

**a) To prevent all runtime errors from occurring.**

b) To provide a structured way to manage and respond to runtime errors or unexpected events.

c) To improve the performance of code by optimizing error-handling routines.

d) To replace the need for traditional error codes and return values.

**Question 21:**

Chief among the principles of the object-oriented approach

are the following:

1. Abstraction, Open/Close Principle, and Modularity.
2. Inheritance, Encapsulation, and Interface Segregation Principle.
3. **Abstraction, Encapsulation, and Modularity.**
4. Inversion Control, Encapsulation, and Inheritance.

**Question 22:**

Some of the algorithm design patterns discussed in your textbook are:

1. Recursion, Divide-and-conquer, Brute force, The greedy method, and Inheritance
2. Divide-and-conquer, Prune-and-search, Dynamic programming and Facade
3. Amortization, Brute force, Dynamic programming, and Abstraction
4. **Recursion, Amortization, Brute force, and, Dynamic programming**

**Question 23**

Some of the software engineering design patterns discussed in your textbook are:

1. Position, Adapter, Iterator, and Binary Search method
2. Template method, Composition, Comparator, and Transaction method
3. **Adapter, Template method, Comparator, and Decorator**
4. Position, , Composition, Decorator, and Singleton

**Question 24**

Why are primitive data types such as int and float generally not classified as data structures?

a) Because they are stored in contiguous memory blocks.

**b) Because they cannot represent collections or relationships among multiple elements.**

c) Because they are implemented differently in each programming language.

d) Because they lack compatibility with algorithmic operations.

**Question 25**

What is the time complexity of accessing an element in a linked list?

1. **O(n)**
2. O(nlogn)
3. O(log n)
4. O(1)

**Question 26**

A recursive algorithm divides an input array in half in each recursive call until it reaches its base case and then backtrack, what best describes its time complexity:

1. O(n)
2. O(nlogn)
3. **O(log n)**
4. O(n²)

**Question 27**

What is true about an ADT of a data structure:

1. It defines the complexity of operations of a data structure
2. **It is a description of operations performed on a data structure**
3. It is an implementation of data structure operations
4. It defines the public interface of a data structure

**Question 28**

In a dynamically allocated array, when the allocated size is **exceeded**, what commonly happens?  
a) The program crashes  
b) Memory is automatically expanded  
c) A segmentation fault occurs  
**d) A new, larger array is allocated and the elements are copied**

**Question 29**

In a **singly linked list**, what is the **time complexity** of inserting an element at the end?  
a) O(1)  
**b) O(n)**  
c) O(log n)  
d) O(n²)

**Question 30**

What happens when attempting to delete the last node of a singly linked list without a tail pointer?  
**a) The previous node still holds a reference to the deleted node**  
b) The program crashes  
c) The previous node’s next pointer is set to NULL  
d) The last node is automatically garbage collected

**Question 31**

Which of the following statements about the memory usage of a doubly linked list compared to a singly linked list is true?

a) A doubly linked list requires twice the memory per node compared to a singly linked list  
**b) A doubly linked list requires more memory per node compared to a singly linked list**  
c) A doubly linked list requires the same amount of memory per node as a singly linked list  
d) A doubly linked list requires less memory because it reduces traversal time

**Question 32**

Which of the following operations has a significantly better performance in a doubly linked list compared to a singly linked list?

a) Searching for an element  
b) Reversing the list  
c) Traversing from head to tail  
**d) Sorting the list**

**Question 33**

What happens when a recursive function is missing its base case?

a) The function executes normally and returns a result  
b) The function executes indefinitely until stopped manually  
c) The function will throw a compilation error  
**d) The function executes for a fixed number of iterations based on system constraints**

**Question 34**

What is the worst-case time complexity of a recursive function that makes k recursive calls per function call and runs until depth n?

a) O(n)  
**b) O(k^n)**  
c) O(n^k)  
d) O(log n)

**Question 35**

What is the time complexity difference between recursive and iterative Fibonacci calculations?

a) Recursive: O(n), Iterative: O(1)  
**b) Recursive: O(2^n), Iterative: O(n)**  
c) Recursive: O(n log n), Iterative: O(n)  
d) Recursive: O(n!), Iterative: O(log n)

**Question 36**

What is the primary reason iterative solutions are often preferred over recursion for problems like factorial calculation?

**a) Iterative solutions use less memory**  
b) Recursive solutions are slower in all cases  
c) Recursive solutions require global variables  
d) Iterative solutions always have better time complexity

**Question 37**

According to the text, what factors influence the running time of an algorithm?

a) Hardware environment  
b) Software environment  
c) Input size and variation in inputs of the same size  
**d) All of the above**

**Question 38**

According to your textbook, why do we need precise methods for analyzing data structures and algorithms?

**a) To determine if an algorithm is implemented correctly**  
b) To classify some algorithms and data structures as “good”  
c) To ensure that an algorithm is written in a particular programming language  
d) To determine the security of an algorithm

**Question 39**

When deleting the last node of a singly linked list, why is the operation considered inefficient?

a) Because arrays are required to handle the deletion.

**b) Because there is no back reference to identify the previous node.**

c) Because Java requires explicit garbage collection.

d) Because the tail pointer automatically resets to null.

**Question 40**

Which primitive operations are assumed to take constant time in algorithm analysis? (Select all that apply)

**a) Assigning a value to a variable.**

**b) Evaluating an expression.**

**c) Accessing an array element.**

d) Looping through an entire array automatically.

**Question 41**

Which operation requires shifting elements to fill the gap in an array?

a) Traversal

b) Update

**c) Deletion**

d) Access

**Question 42**

Which statement about the space complexity of arrays is correct? (Select all that apply)

a) Worst case is O(1) since arrays are fixed.

**b) Worst case is O(N) since all N elements are stored.**

c) Worst case is O(N2) due to shifting elements.

d) Space complexity cannot be determined.

**Question 43**

Why might a doubly linked list be chosen over a singly linked list? (Select all that apply)

**a) Supports traversal in both directions**

**b) Allows easier deletion with access to a node**

c) Uses less memory than a singly linked list

**d) Provides O(1) insertion at both head and tail**

**Question 44**

In binary search using recursion, what happens if the target value is less than the middle element?

a) The function stops and returns false.

**b) The search continues in the left half of the array.**

c) The search continues in the right half of the array.

d) The array is reversed before continuing.

**Question 45**

Which of the following are examples of linear recursion? (Select all that apply)

a) Binary sum of an array

**b) Fibonacci sequence (first attempt)**

**c) Summing the first n elements of an array**

d) PuzzleSolve algorithm

**Question 46**

Which best describes why tail recursion can be transformed into iteration?

a) It always halves the input size.

**b) No computation remains after the recursive call, so a loop can substitute the recursion.**

c) Each recursive call creates multiple subproblems.

d) It always runs in O(logn) time.

**Question 47**

When computing powers using recursion, what are the advantages of using repeated squaring over the simple recursive method? (Select all that apply)

**a) It reduces the total number of multiplications.**

**b) It improves runtime complexity from O(n) to O(logn).**

c) It eliminates the need for a base case.

d) It avoids recursion entirely.

**Question 48**

Why is the basic recursive Fibonacci implementation inefficient?

a) It lacks a base case.

**b) It recomputes the same subproblems multiple times, leading to exponential growth.**

c) It requires storing pairs of numbers.

d) It uses tail recursion.

**Question 49**

Given the following algorithm, what is its time complexity:

**Algorithm** doSomething(*A*):

***Input:*** An array *A* of *n* comparable elements

**A screenshot of a computer

AI-generated content may be incorrect.**

a) O(1)  
b) O(n)  
c) O(log n)  
**d) O(n²)**

**Question 50:**

Which of the following algorithms can be applied to arrays for sorting? (Select all that apply)

**a) Bubble Sort**

**b) Insertion Sort**

**c) Merge Sort**

d) Binary Search

**Section 2: Analysis and Problem-Solving (50 points)**

**Question 51: (5 points)**

**Write a recursive algorithm** in pseudocode to find the maximum element in an array A of size n. The algorithm should compare elements recursively to find the maximum.

1. Provide the pseudocode (3 points), and

**Algorithm** RecursiveMax(array, i=0, current\_max = None):

**if** i == len(array) **then:**

**return** current\_max

{Base case, we have checked every element in the list so return}

**if** current\_max == None **or** array[i] > current\_max **then:**

current\_max = array[i]

{check if current item is greater than the largest item we’ve seen so far}

**return** RecursiveMax(array, i = (i +1), current\_max)

{recurse and check the next item in the list}

1. explain the time complexity of the algorithm in Big-O notation (2 points).

**O(n).** The algorithm always has to check every item in the list to determine the max, so the function will be called n times.

**Question 52: (10 points)**

****

**A white sheet with black text

AI-generated content may be incorrect.**

**A white background with black text

AI-generated content may be incorrect.**

1.

|  |  |
| --- | --- |
| C1 | n |
| C2 | n |
| C3 | n |
| C4 | n^2 |
| C5 | n^2 |
| C6 | n^2 |
| C7 | n |

2. f(n) = 2n^3 + 4n

3. for every c >= 5 and n0 >=2 f(n) <= c \* g(n)

**Question 53: (5 points)**

Given the following pseudocode, write the exact output for n = 4:

**Algorithm OutputExample(n)**

**if n <= 0 then**

**return**

**OutputExample(n - 1)**

**print(n)**

1. What is the output when OutputExample(4) is called? (3 points)

1

2

3

4

1. Explain how the recursion works step-by-step. (2 points)

4 is not <= 0, so OutputExample(3) is called. Then OutputExample(2), OutputExample(1), and finally OutputExample(0). This is the base case, so the function returns, and prints 1, then 2, 3, and finally 4.

**Question 54: (5 points)**

You are given two algorithms, A and B, to solve the same problem. The pseudocode for both algorithms is provided below.

**Algorithm A**:

**Algorithm A(n)**

**for i = 1 to n do**

**print(i)**

**Algorithm B**:

**Algorithm B(n)**

**if n == 0 then**

**return**

**print(n)**

**B(n - 1)**

a) What is the time complexity of **Algorithm A**? (2 points)

**O(n)**

b) What is the time complexity of **Algorithm B**? (2 points)

**O(n)**

c) Which algorithm is more efficient as n becomes large? Explain why. (1 points)

Algorithm A. While they both have the time complexity, Algorithm A is more memory efficient. It is only called once, while Algorithm B is called n times, adding a new frame on the call stack each time.

**Question 55: (4 points)**

Estimate the time complexity of the following algorithm using Big-O notation:

**Algorithm EstimateExample(n)**

**for i = 1 to n do**

**for j = 1 to n do**

**print(i \* j)**

a) Provide the time complexity. (2 points)

**O(n^2)**

b) Explain the reasoning behind your estimate. (2 points)

The algorithm contains a nested loop. The inner loop is going to run n times, once for each time the outer loop runs, performing n operations each time it runs. In total the inner loop is going to run n \* n times, or n^2.

**Question 56: (4 points)**

You are given the following pseudocode that finds the sum of the first n integers:

**Algorithm SumExample(n)**

**sum = 0**

**for i = 1 to n do**

**sum = sum + i**

**return sum**

a) What is the time complexity of this algorithm? (2 points)

**O(n)**

b) Explain whether this algorithm can be improved in terms of time complexity. (2 points)

No, to find the sum every number up until n has to be found and added.

**Question 57: (4 points)**

Consider the following two algorithms:

* Algorithm 1 has a time complexity of O(n^2).
* Algorithm 2 has a time complexity of O(n log n).

As n increases:

a) Which algorithm will perform better for small values of n (e.g., n = 10)? (1 points)

**Algorithm 2**

b) Which algorithm will perform better for large values of n (e.g., n = 10000)? (1 points)

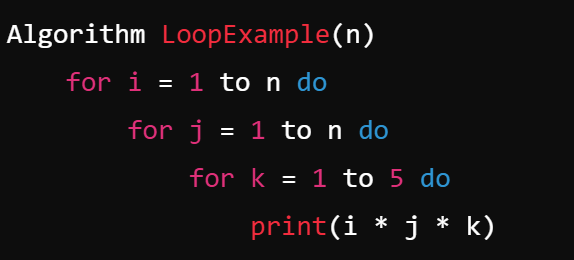
**Algorithm 2**

c) Justify your answer using growth functions and time complexity analysis. (2 points)

n^2 will always be larger than n log n. Since log n is always smaller than n, n \* log n will always be smaller than n \* n.

**Question 58: (3 points)**

Given the following pseudocode, analyze the time complexity:

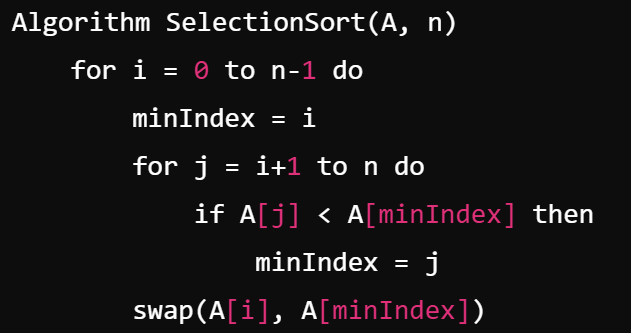


The time complexity will be O(n^2). The innermost loop only runs 5 times for each time the middle loop runs, the middle loop runs n times for each time the outer loop runs, and the outer loop runs n times. So the print statement will be called 5 \* n \* n times, or 5n^2, so the asymptotic growth rate is O(n^2).

**Question 59: (4 points)**

You are given two sorting algorithms, **Algorithm A** and **Algorithm B**. The pseudocode for both algorithms is provided below:

**Algorithm A**:



**Algorithm B**:

A black background with white text

Description automatically generated

a) What is the time complexity of **Algorithm A**? (1 points)

**O(n^2)**

b) What is the time complexity of **Algorithm B**? (1 points)

**O(n^2)**

c) Which algorithm is more efficient in practice for small arrays? (2 points)

Algorithm A will be more efficient. It is never going to swap with anything to the left of the minindex, so at most it is going to do n swaps, where the second algorithm could possibly do more.

**Question 60: (6 points)**



such that the function *f* (*n*) is *O*(*g*(*n*)),



since *f* (*n*) ≤ *c* ·*g*(*n*) when *n* ≥ *n*0.

(a) Find the function *g*(*n*) (2 points)

**g(n) = 2^(n+2)**

(b) Find Big-O of *f(n)* by finding constants c and *n* ≥ *n*0 (4 points)

c = 2 , n0 = 30.

f(30) =~ 4.3 \*10^9,

2 \* g(30) =~ 8.6\*10^9

**End of Document**