Data Structures Week 3 Exercise

Vammy Johannis Jiang

2702368122

1. A. The following code’s is split into an outer loop and inner loop. The outer loop runs n times, while the inner loop also runs n times for each iteration of the outer loop. The total times the output statement is executed is n \* n which is the square of n.

In which we can conclude that the algorithm is O(n^2).

B. The following code consist of an outer loop and inner loop. The outer loop runs n times, while the inner loop also runs 2 times for each iteration of the outer loop. The total times the output statement is executed is n \* 2. In which we can conclude that the algorithm is n \* 2 which can be simplified to O(n) by ignoring the lower constant factor.

C. The following code consist of an outer loop and inner loop. The outer loop runs n times, while the inner loop also runs i + 1 times for each iteration of the outer loop.The total times the output statement is executed is n \* (n-1) / 2 + n. In which we can conclude that the algorithm is O(n^2) because the time complexity of the algorithm above increases quadratically with the input size (n).

D. The following code consist of an outer loop, inner loop, and a loop nested for print statement. The outer loop runs n-1 times, while the inner loop also runs i - 1 times for each iteration of the outer loop. The loop nesting the print statement will execute for each instance of i. The total times the output statement is executed is \* (n-1) / 2 - n.In which we can conclude that the algorithm is O(n^2) because the time complexity of the algorithm above increases quadratically with the input size (n).

1. - The first loop:  
   First iteration (when ‘i = 3’)

[0, 1, 2, 3, 3, 5, 6, 7]

Second iteration (when ‘i = 4’)  
  
[0, 1, 2, 3, 3, 3, 6, 7]  
  
Third iteration (when ‘ i = 5’)  
  
[0, 1, 2, 3, 3, 3, 3, 7]  
  
Fourth iteration (when ‘i = 6’)  
  
[0, 1, 2, 3, 3, 3, 3, 3]

* The second loop:

First iteration (when ‘i = 7’)  
  
[0, 1, 2, 3, 4, 5, 6, 6]

Second iteration (when ‘i = 6’)  
  
[0, 1, 2, 3, 4, 5, 5, 6]

Third iteration (when ‘i = 5’)

[0, 1, 2, 3, 4, 4, 5, 6]

Fourth iteration (when ‘i = 4’)  
  
[0, 1, 2, 3, 3, 4, 5, 6]

1. A. Sum of an Array

The algorithm calculates the sum of an integer array. It takes in an integer array as input and returns the sum of all integers in the array.

Time Complexity (T(n))

The time complexity of this algorithm is O(n) because the loop iterates through the entire array once, performing a constant number of operations inside the loop. The time complexity of the sumArray algorithm is O(1) + O(n) + O(1), in which simplifies into O(n).

Space Complexity (O(n))

The space complexity of this algorithm is O(1). This is because the space complexity of this algorithm is constant, where the algorithm uses a constant amount of extra space to store the sum variable of the algorithm. Hence, the space complexity is O(1).

B. Matrix Multiplication

The algorithm multiplyMatrices takes two integer matrices firstMatrix and secondMatrix, and their dimensions r1, c1, and c2, as input, and returns the product of the two matrices as a new matrix.

Time Complexity (T(n))

The time complexity of this algorithm is O(r1 \* c2 \* c1). The time complexity of the algorithm is determined by the total number of operations performed during matrix multiplication. With three nested loops iterating over the elements of the matrices, The outer loop runs for r1 times, the middle loop runs for c2 times, and the innermost loop runs for c1 times. Hence, the time complexity is O(r1 \* c2 \* c1), where r1, c1, and c2 represent the dimensions of the matrices.

Space Complexity (O(n))

The space complexity of this algorithm is O(r1 \* c2).This is because the space complexity of this algorithm creates a new matrix to store the result of the multiplication, which has dimensions corresponding to the number of rows of the first matrix and the number of columns of the second matrix (r1 x c2).

C. For Looping

The algorithm consists of three for-loops that iterate over the range [0, n) and perform the same operation in each iteration. The operation is result = i + i, which has a time complexity of O(1).

Time Complexity (T(n))

The time complexity of this algorithm is O(n) because there are three loops each iterating from 0 to n, each loops has time complexity of O(n). The time complexity of the algorithm is O(n + n + n), which simplifies to O(3n), it is simplified into O(n) according to Big O rule.

Space Complexity (O(n))

The space complexity of this algorithm is O(1). This is because the space complexity of this algorithm is constant, where the algorithm uses a constant amount of extra space to store the i, j and k variable of the algorithm. Hence, the space complexity is O(1).

D. While Looping

The algorithm takes an integer n as input and performs a series of arithmetic operations in a while-loop.

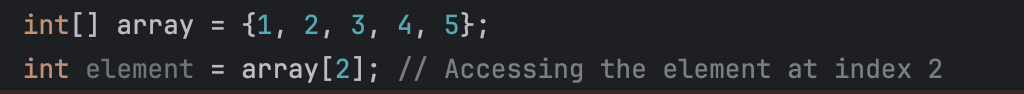
Time Complexity (T(n))

The time complexity of this algorithm is O(log n) because there is a while loop that iterates as long as variable i is greater than 0. In the loop itself, there are time operations mutiplication of i by 2 and division i by 2. Since the value of i is halved in each iteration until it becomes less than or equal to 0, which takes logarithmic time relative input size of n. The time complexity of the algorithm O(log n).

Space Complexity (O(n))

The space complexity of this algorithm is O(1). This is because the space complexity of this algorithm is constant, where the algorithm uses a constant amount of extra space to store the i and k variable of the algorithm where the input size of n does not affect the constant space. Hence, the space complexity is O(1).

1. - Constant Time Complexity (O(1)):  
   Example: While Looping



* Logarithmic Time Complexity(O(log n)):

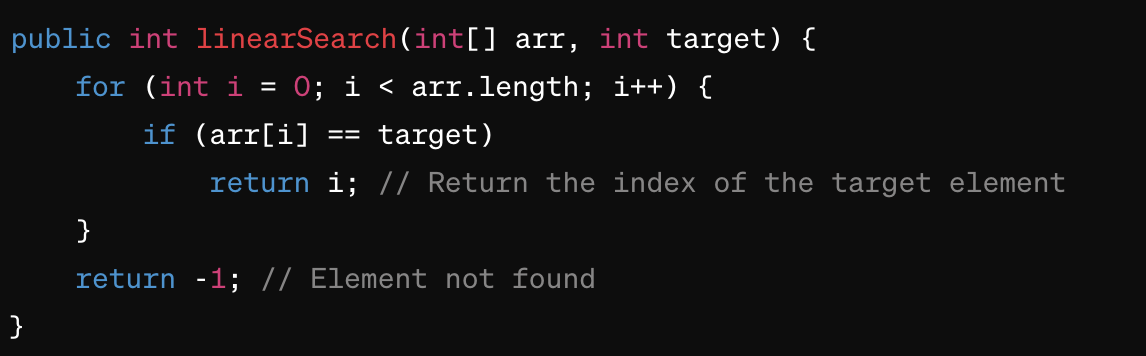
Example : While Looping

A number and symbols on a white background

Description automatically generated

* Linear Time Complexity (O(n)):

Example : Linear Search



* Linearithmic Time Complexity (O(n log n)):

Example : Merge Sort Algorithm

A screen shot of a computer program

Description automatically generated

* Quadratic Time Complexity (O(n^2)):

Example : Bubble Sort Algorithm

A screen shot of a computer code

Description automatically generated

1. An Abstract Data Type (ADT) is a high-level description of a collection of data and operations that can be performed on it, without specifying implementation details. It encapsulates data structure and behavior, promoting modularity and separation of concerns. ADTs provide a conceptual model that defines the permissible actions that can manipulate the data, allowing developers to focus solely on functionality. Examples of ADTs include stacks, queues, lists, and trees.

* Implementation of Queues in Java

A computer screen shot of a program code

Description automatically generated

* Implementation of LinkedList



|  |  |  |
| --- | --- | --- |
| Feature | List | Interface |
| Implementation | Interface | Class |
| Resize Behavior | Dynamic resizing | Dynamic resizing |
| Access Speed | Slower due to abstract methods | Faster due to direct access to array elements |
| Memory Overhead | Lower | Higher due to internal array and additional features |
| Usage | General-purpose | Typically used when fast element access is required |
| Initialization | Can be initialized with various implementations | Needs to be instantiated as an ArrayList |
| Import Statement | **import java.util.List;** | **import java.util.ArrayList;** |

A computer screen shot of a program

Description automatically generated

1. A screenshot of a computer code

   Description automatically generated