1. Determine the asymptotic running time of the following procedure (an exact computation of number of basic operations is not necessary):

A white screen with black text

Description automatically generated

1. First Loop (O(n)): The first loop initializes the array arr with n elements, all set to 1. This loop runs n times.
2. Second Nested Loops:

Outer loop runs n times (i from 0 to n-1).

Inner loop runs from j = i to n, which means for the first iteration (i = 0), it runs n times, for the second iteration (i = 1), it runs n-1 times, and so on.

Hence first loop is O(n) and the second one is O(n2) and the entire time complexity is O(n2)

1. Consider the following problem: As input you are given two sorted arrays of integers. Your objective is to design an algorithm that would merge the two arrays together to form a new sorted array that contains all the integers contained in the two arrays. For example, on input [1, 4, 5, 8, 17], [2, 4, 8, 11, 13, 21, 23, 25] the algorithm would output the following array: [1,2,4,4,5,8,8, 11, 13, 17, 21, 23, 25] For this problem, do the following:
   1. Design an algorithm Merge to solve this problem and write your algorithm description using the pseudo-code syntax discussed in class.

Algorithm Merge(arr1, arr2)

Input: Two sorted arrays arr1 and arr2

Output: A merged sorted array

Initialize an empty array mergedArray

i = 0, j = 0 // indices for arr1 and arr2

while i < len(arr1) and j < len(arr2) do

if arr1[i] <= arr2[j] then

Append arr1[i] to mergedArray

i = i + 1

else

Append arr2[j] to mergedArray

j = j + 1

end while

// Append remaining elements from arr1

while i < len(arr1) do

Append arr1[i] to mergedArray

i = i + 1

// Append remaining elements from arr2

while j < len(arr2) do

Append arr2[j] to mergedArray

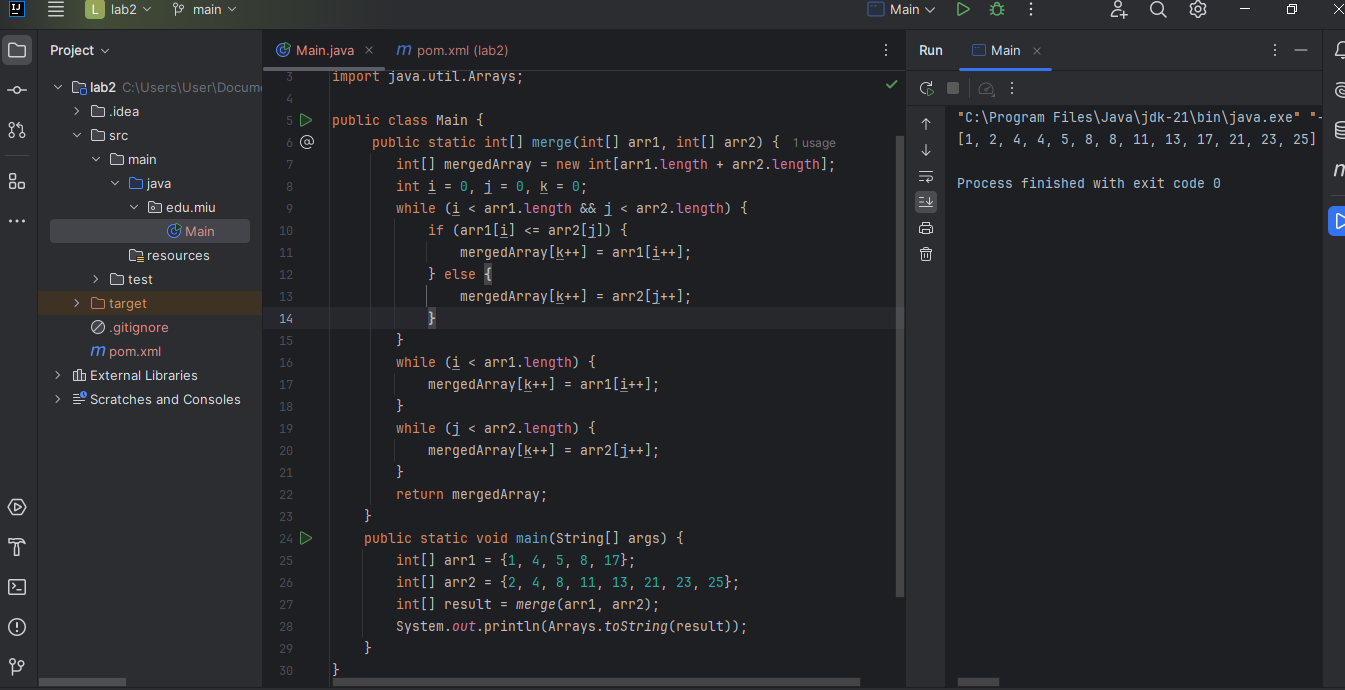
j = j + 1

return mergedArray

* 1. Examining your pseudo-code, determine the asymptotic running time of this merge algorithm

Each element from both arrays is compared once and placed into the mergedArray. Therefore, the time complexity is O(n + m) where n is the size of arr1 and m is the size of arr2.

* 1. Implement your pseudo-code as a Java method merge having the following signature: int[] merge(int[] arr1, int[] arr2) Be sure to test your method in a main method to be sure it really works!



1. Big-oh and Little-oh. Use the definitions of O(f(n)) and limit facts about o(f(n)) given in class to decide whether each of the following is true or false, and in each case, prove your answer.

A close up of a number

Description automatically generated

1. True => because 1 doesn’t grow but 4n2 is already o(n2)
2. True => because n2grows faster than n hence cannot be O(n)
3. True => Log n grows slower than O(n)
4. False => That’s the ideal scenario of O(n)
5. Power Set Algorithm. Given a set X, the power set of X, denoted P(X), is the set of all subsets of X. Below, you are given an algorithm for computing the power set of a given set. This algorithm is used in the brute-force solution to the SubsetSum Problem, discussed in the first lecture. Implement this algorithm in a Java method: List powerSet(List X) Use the following pseudo-code to guide development of your code

A white background with blue text

Description automatically generated

A screenshot of a computer screen

AI-generated content may be incorrect.