A picture containing text

Description automatically generated

**CS435**

*Prof. Emdad Khan*

*September 2019*

*Lab#2*

***Group 1***

*Group members:*

*Asad Ali Kanwal*

*Aser Ahmad Ibrahim Ahmad*

*Jean Wilbert Volsy*

*Zayed Hassan*

1. Problem 1

|  |  |
| --- | --- |
| Line | *Operations count* |
| int[] arrays(int n) { |  |
| int[] arr = new int[n]; | *2n* |
| for(int i = 0; i < n; ++i){ | *1 + n + 2n* |
| arr[i] = 1; | *2n* |
| } |  |
| for(int i = 0; i < n; ++i) { | *1 + n + 2n* |
| for(int j = i; j < n; ++j){ | *(1 + n + 2n)n* |
| arr[i] += arr[j] + i + j; | *(6n)n* |
| } |  |
| } |  |
| return arr; | *1* |
| } |  |

From (A) and (B):

1. Problem 2
2. The pseudocode for the merge algorithm is as follows (code is in a separate file, Merge.java):

|  |  |
| --- | --- |
| **Algorithm** merge(A1, A2) |  |
| ***Input:*** two sorted arrays A1 and A2 |  |
| ***Output:*** one merged and sorted array R |  |
|  | *Count of operations* |
| totalLength = A1.length + A2.length | *4* |
| R = new array[totalLength] //the merged array to return | *2n* |
| a1Index = 0 //indicates which element is next | *1* |
| a2Index = 0 //in each of the arrays | *1* |
| for (i = 0 ; i < totalLength ; i++) { | *1 + n + 2n* |
| //are there elements left in both arrays?  if (a1Index < A1.length & a2Index < A2.length) { | *2n* |
| /\*compare next element in a1 to next element in a2. Whichever element is selected, we will move to the element next in its array\*/  if (A1[a1Index] < A2[a2Index]) { | *3n* |
| R[i] = A1[a1Index] | *3n* |
| a1Index++ | *2n* |
| } else { |  |
| R[i] = A2[a2Index] | *3n* |
| a2Index++ | *2n* |
| } |  |
| } else { |  |
|  |  |
|  |  |
| /\*if one array is totally used, then take elements from the other one only.\*/  if (a1Index = A1.length) { | *n* |
| R[i] = A2[a2Index] | *3n* |
| a2Index++ | *2n* |
| } else { |  |
| R[i] = A1[a1Index] | *3n* |
| a1Index++ | *2n* |
| } |  |
| } |  |
| } |  |
| return R | *1* |
| } |  |

From the count above:

1. Calculation of the asymptotic running time:

1. Problem 3
2. Problem 4:

Code for the power set algorithm is in a separate file (Powerset.java).

*Lab 2 continued*

1. Problem 1

Prove that

1. Base case:
2. Induction step:

Assume

1. Problem 2
2. is ?
3. is ?

From (1) and (2):

1. is ?

From (1) and (2):

1. Problem 3

|  |  |
| --- | --- |
| **Algorithm** recursiveFactorial (n) | *Count of operations* |
| **Input:** a non-negative integer *n* |  |
| **Output:** *n!* |  |
| if (n = 0 | n = 1) then | *3* |
| return 1 | *1* |
| return n \* recursiveFactorial(n – 1) | *4 (n – 1)* |

1. Guessing method:

T(0) = 4

T(1) = 4

T(2) = 4 + T(1) = 4 + {4}

T(3) = 4 + T(2) = 4 + {4 + 4}

T(4) = 4 + T(3) = 4 + {4 + 4 + 4}

T(5) = 4 + T(4) = 4 + {4 + 4 + 4 + 4}

T(n) = 4 + T(n – 1) = 4 + 4 ( n – 1)

Asymptotic running time:

1. Proof of algorithm correctness:
2. It has a base case, i.e. a line of code that executes without calling the function recursively. This is the line: if (n = 0 | n = 1) then return 1.

Also, since the recursion line “return n \* recursiveFactorial (n – 1)” subtracts “1” with each call, then it will eventually lead to the base case *n = 1*.

1. The base cases mentioned above return “1”, which is correct by definition of the factorial function.
2. Assume the call to recursiveFactorial (n – 1) will return a correct value, then we try to prove that the call to recursiveFactorial (n) is correct, too:

According to the algorithm above, the call to recursiveFactorial (n) is equal to n\*recursiveFactorial (n – 1), which is equal to n!.

1. From the three points above, we have the proof that the proposed recursiveFactorial algorithm is correct.
2. Problem 4

|  |  |
| --- | --- |
| **Algorithm** iterativeFibonacci (n) | *Count of operations* |
| ***Input:*** non negative number *n* |  |
| ***Output:*** array of Fibonacci numbers [0….n] |  |
| if (n = 0 | n = 1) then return n | *4* |
| initialize array fib [n + 1] | *2 (n + 1)* |
| fib [0] = 0 | *2* |
| fib [1] = 1 | *2* |
| for (i = 2 ; i ≤ n ; i++) do | *1 + n + 1 + 2* |
| fib [i] = fib [i – 1] + fib [i – 2] | *n(2 + 2 + 1 + 2)* |
| return fib | *1* |

To know the asymptotic running time:

Proof that the algorithm is correct:

1. It has two base cases: and , and they yield the correct Fibonacci’s values.

Also, the loop in the algorithm starts with . According to the algorithm, it is calculated using the previous two numbers, which are and , the base cases.

1. The loop invariant is . assuming the call to is true, we try to show that also holds. From the pseudocode above: . Since and are the two numbers preceding , then the code is correct according to the definition of Fibonacci’s series.
2. Problem 5

First, we rewrite the given formula on the format of the Master formula:

By comparison to the general form of the master formula, we find that:

1. Problem 6

|  |  |
| --- | --- |
| **Algorithm** countZerosAndOnes (A, start, end) | *Count of operations* |
| ***Input:*** sorted array A of zeros and ones, starting index, ending index |  |
| ***Output:*** count of zeros, count of ones |  |
| if (start ≥ end) then | *1* |
| ones = A.length – start – 1 | *3* |
| zeros = A.length – ones | *2* |
| return zeros, ones | *2* |
| mid = (start + end) / 2 | *3* |
| if (A[mid] = 1) then | *2* |
| return countZerosAndOnes (A, start, mid) | *2 + T(n/2)* |
| else |  |
| return countZerosAndOnes (A, mid +1 , end) | *3 + T(n/2)* |

According to the master formula:

Since