Data Structures and Algorithms

Coursework Assignment 1

Candidate Number: 181509

Question 1a: **Algorithm** reversed(L1, L2)

**Input:** Two singly linked lists of integers of same length.

**Output:** Boolean value of reverse check of lists.

reverse := true

List1 := L1

List2 := L2

Size := List1.size()

List2reverse.addFirst(L2.head)

**for** i := 0 **to** size-1

temp := List2.getNext()

List2reversed.addFirst(temp)

**endfor**

temp1 := List1.head

temp2 := List2reserved.head

**for** j := 0 **to** j < size

**if** (temp1 != temp2)

reverse = false

**return** reverse

**else**

temp1 := List1.getNext()

temp2 := List2reversed.getNext()

**endif**

**endfor**

**return** reverse

Question 1b: **Algorithm** reversedDoublyLinked(L1, L2)

**Input:** Two doubly linked lists of integers of the same length.

**Output:** Boolean value of reverse check of lists.

reverse := true

List1 := L1

List2 := L2

temp1 := List1.header

temp2 := List2.trailer

size := List1.size()

**for** i := 0 **to** i <size-1

**if** (temp1 != temp2)

reverse := false

**return** reverse

**else**

temp1 := List1.getNext()

temp2 := List2.getPrev()

**endif**

**endfor**

**return** reverse

Question 1c: The complexity of both algorithms is O(n).

Question 2a: Algorithm grum takes each element in array A and multiples them with each element in array B while adding the results of those operations and storing them in the “val” variable to finally return the sum of all results.

Question 2b:

Line of code Primitive Operations

val = 0 2

for i = 0 to n – 1 do 2n

for j = 0 to n – 1 do 2n(n-1)

val = val + (A[i] \* A[j]) 4(n-1)

return false 1

2 + 2n + 2n2 -2n + 4n – 4 + 1 = 2n2 + 4n – 1 🡪 total number of primitive operations.

Question 2c: O(n2) is the complexity of the algorithm.

Question 2d:

**Algorithm** grum2(A, B, n)

**Input:** Arrays of integers, A and B, both of length n

**Output:** integer value

val := 0

int j := 0

int i := 0

while j =< size do

if i % size then do

j = j + 1

i = 0

val = val + (A[i] \* B[j])

i = i + 1

return val

Question 3a: enqueue (e) : in stacks it is the same as push (e) method, so:

**enqueue** (e){

stack.push (e)}

will do the same thing.

dequeue () : Given that stack 1 is the initial list and stack2 is null list that will hold the queue after removal:

**dequeue** (){

**for** i := 1 **to** n 2n

stack2[i-1] = stack[i] 2(n-1)

**endfor**

**return** stack1[0]} 1

Question 3b: Run-time complexity of enqueue operation is O(1).

Question 3c: 2n + 2n -2 + 1 = 4n – 1.

Run-time complexity of dequeue operation is O(n).

Question 4a: Yes, an inorder traversal of T can be identical to a postorder traversal of T, if there are only right nodes on the binary tree T, including subtrees, then two traversals can be identical.

Question 4b: Yes, an inorder traversal of T can be identical to a preorder traversal of T, if there are only left nodes on the binary tree T, including subtrees, then two traversals can be identical.

Question 4c: Yes, if binary tree T consists nothing but a root, only then the postorder traversal of T and the preorder traversal of T can be identical.

Question 4d: If T is actually a full binary tree, then the answers for Q4 (a), (b) and (c) will be “no”, they will not be identical.