COMP 352: Data Structure and Algorithms

Name: Wang Haochen

Assignment 2

Q(1).

i)

Algorithm findTwoNumbersOfValue(A, differ)

Input: A is an unsorted array of integers.

Differ is differ for two numbers.

Output: Display the indices and the values of there elements.

find<-0 //O(1)

Declarei, j //O(1)

// find the array element one by one

For i<-0; i<A.length; i++ //O(n+1)

//find the complement of the number we are looking for.

find = A[i] + differ //check we find the complement in A array.

If find>0 && find in A then

Print(“All 2-Elements differing by a value of”+differ+ “in the array are:”)

Print(“Indices”+ i + “&” + j + “with values” + A[i] + “&” + find )

End\_if

End\_for

ii). This is simply a straight way to implement the algorithm.

iii). f(n) = 1+1+(n+1)= 3+n, for every n>=0

n<=nfor n>=0;

3<=3n for n>=1;

3+n<=4n, for every n>=1, C=4, n0=1; so Big(O)is O(n);

iv). 3+n>=nfor n>=0;

3+n>=n, for every n>=0, C=1, n0=0; So Big-Omega(Ω(n)) isΩ(n);

Q2

i). Algorithm findTwoNumbersOfValueOfQueue(A, differ)

Input: A is an unsorted array of integers.

Differ is differ for two numbers.

Output: Display the indices and the values of there elements.

//create a Queue object

Queue q<- new Queue() //O(1)

//insert all the numbers into the queue

For i<-0; i<A.length; i++ //O(n+1)

q.add(A[i])

end\_for

//first of value of the queue

Index<- q.remove() //O(1)

//index of the first value in the array

countOfIndex<-0 //O(1)

//get next value to be compared

Temp <- q.remove() //O(1)

//index of the values of the other in the queue

countOfRest<-countOfIndex+1 //O(1)

while not q.isEmpty() //O(n+1)

if countOfRest<=q.length

//check if two number’s differ is our differ

If(temp-index)==differ

Print(“All 2-Elements differing by a value of”+differ+ “in the array are:”)

Print(“Indices”+ countOfIndex+ “&” + countOfRest + “with values” + index + “&” + temp )

End\_if

//Put the temp back to the queue

q.add(temp)

countOfRest<-countOfRest+1

` end\_if

//get next the first value

Index<-q.remove()

countOfIndex<-countOfIndex+1

countOfRest<- countOfIndex+1

end\_while

ii). I choose the queue method, because I think it easier to implement. especially to find the indices of the values found.

iii).f(n)=1+(n+1)+1+1+1+1+(n+1)=2n+7

2n<=2n for n>=0;

7<=7n for n>=1;

2n+7<=9n, for every n>=1, C=9, n0=1; so Big(O)is O(n);

iv). 2n+7>=n for n>=0;

2n+7>=n, for every n>=0, C=1, n0=0; So Big-Omega(Ω(n)) is Ω(n);

Q3

a). f(n) = n!+3nlogn+100000/n

n!+3nlogn+100000/n <=(7+3+100000)n!, for every n>=1;

n!+3nlogn+100000/n<=100010n!, for every n>=1, C=100010, n0=1;

O(n!);

n!+3nlogn+100000/n>=n!, for every n>=1, C=1, n0=1;

Ω(n!)

n!+3nlogn+100000/n is Θ(n!);

b). f(n) = 20000n-2n^5

20000-2n^5<=1 n>=10

20000n-2n^5>=1, for every n>=10, C=1, n0=10;

O(1)

C). f(n)=n^2+900000000000n^2

n^2+900000000000n^2<=900000000001n^2, for every n>=0, C=900000000001, n0=0;

O(n^2)

n^2+900000000000n^2>=n^2, for every n>=0, C=1, n0=0;

Ω(n^2)

n^2+900000000000n^2 isΘ(n^2);

d). f(n)=((n+1)logn+log(n+3)+(n^2+7)+20)/n

f(n)=logn+logn/n+log(n+3)/n+n+27/n

logn<=n for n>=0;

logn)/n <=n for n>=1;

Log(n+3)/n<=n for n>=1;

n<=n for n>=1;

27/n<=27n for n>=1;

((n+1)logn+log(n+3)+(n^2+7)+20)/n<=31n, for every n>=1, C=31, n0=1;

O(n);

e). f(n)=1/n^5-30n^7

1/n^5-30n^7<=1 for every n>=1;

1/n^5-30n^7<=1, for every n>=1, C=1, n0=1;

O(1)