Algorithms and Complexity COMP\_90038

Semester 2 August2020

Name: Sakshi Chandel

Student ID:1124298

Question 1:

**Solution 1(a):**

The basic idea of this algorithm is to initialize an array Z of size such that it can have elements of both array X and Y(size of array X+ size of array Y).Then from array X put all elements to array Z .Then check each element of array Y if it is present in array X using binary search .If present ,don’t add to array Z else add to array Z. Hence will get union of elements from array X and array Y in array Z.

procedure FindSetUnion(X,Y )

n=size of array X

initialize array Z with size 2n;

for i ← 0 to i ←n-1 do :

Z[i]  ← X[i];

for j  ← 0 to j  ← n-1 do :

index  ← BinarySearch(X,0,m,Y[j]); #complexity of binary search -> logn

if(index=-1) #element Y[j] is not in array X

Z[k]  ← Y[j];

k++;

**Since here size of both arrays is n that is why complexity will be n+nlogn,**

**which is in O(nlogn).**

**Question 1:**

**Solution 1 (b):**

procedure FindSetUnion(X,Y)

initialise hashtable as hash\_table ;

n ← size of X ; #size of X=size of Y =n

for i ← 0 to i ← n-1 do :

insert(hash\_table,X[i]);

for j ← 0 to j←n-1 do:

result ← search(hash\_table,Y[j]);

if(result=false)

insert(hash\_table,Y[j]);

Overall complexity = O(n)

Question 2:

Solution 2 (a):

1. Wi,j = A[(i\*(i+1))/2 + j]
2. Left and right children of Wi,j

Left child = A[((i+1)\*(i+2))/2 + j]

Right child= A[((i+1)\*(i+2))/2 + j+1]

1. Left and right parents of Wi,j

Left parent = A[((i-1)\*(i))/2 + j-1]

Right parent= A[((i-1)\*(i))/2 + j]

1. Left and right children of the node corresponding to A[k] with level i=

Left child = k+i+1

Right child = k+i+2

Solution 2 b):

Upper bound and lower bound of level l

Upper bound nodes n= (l\*(l+1))/2

Lower bound nodes n = ((l-1)\*l)/2 +1

Explanation : If the level of web is l , then for upper bound(maximum nodes) number of nodes will be keeping in the mind the structural property that the all but the last level will be completely filled.

Also , any level will have at most l nodes where l>=1 then upper bound will have maximum l nodes in the last level where l>=1.

Hence for level 1 to level l :

Total nodes will be 1+2+3+……l

(Since sum of 1+2+3…to n = n(n+1)/2)

And by formula sum of 1+2+3…..l = l\*(l+1)/2.

If the level of web is l , then for lower bound(minimum nodes) number of nodes will be keeping in the mind the structural property that the all but the last level will be completely filled.

Also, any level will have at most l nodes where l>=1 then for lower bound will have minimum 0 nodes in the last level where l>=1.

Hence for level 1 to level l:

Total nodes will be 1+2+3…..+l-1+0 (where 0 node is at level l)

(Since sum of 1+2+3…to n = n(n+1)/2)

And by formula sum of 1+2+3…..+l-1+0= (l-1)\*(l)/2 .( Here n=l-1)

Solution 2 c):

Question 3:

Solution 3

1. **Recurrence relation:** T(N) = 1 + Sum j = 1 to N-1 (T(j))

Let A[0..n-1] be the input array and L(i) be the length of the LongestIncreasingLengths ending at index i such that A[i] is the last element of the LongestIncreasingLengths.

L(i) = max{v1,1 + L(j)} where 0 < j < i and A[j] < A[i]; or

L(i) = 1, if no such j exists.

1. ***Procedure LongestIncreasingLengths(A[0……n-1])*** where

Input = Array of size n from index 0 to n-1

Output = longest increasing length

procedure LongestIncreasingLengths (A[i….n-1])

m <- size of array A;

if(m == 0) then return 1; // Only one subsequence ends at first index, the number itself

ans ← 1;

for(i ← 1 to i ← m-1 ) do:

if(A[i-0] < A[m-1])

ans = max(ans, 1 + LongestIncreasingLengths(A[i…n-1]));

return ans

**c)Time Complexity:**O(2^N)

The time complexity is exponential. There will be 2^n - 1 nodes will be generated for a n sized array.