# Ad hoc problems

Saturday, 14 August 2021 8:04 PM

→ BST } → Nodes are unique


Preorder = (Root left Right)

ID 6 3 -1 9 25 20 14 31 28 36

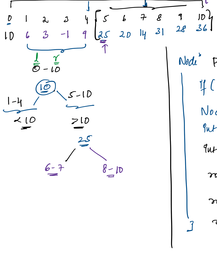
Given Preorder for BST, Construct Tree

Solutions

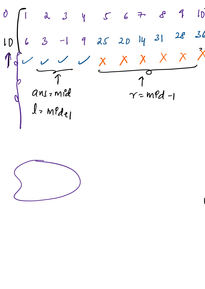
Sor preorder we get inorder → NlogN

Using preorder & inorder construct tree → O(N) using map (extra space)

O(N) using map (extra space)




```
Node* preBST(int pre[], int l, int r){
    if(l > r){return false}
    Node *root = new root(pre[l])
    int val = pre[l]
    int p = ptr(pre[], int l+1, int r, val) // In pre[] from l+1 → r, get index which is less than val
    root->left = (pre, l+1, p)(k)   logN
    root->right = (pre, p+1, r)(N-k-1)
    return root;
}
```




partition can be done using binary search

BC: TC → logN

Recurrence Relation

T(N) = T(k) + T(N-k-1) + logN ?

k=0 in worst case / skewed Tree

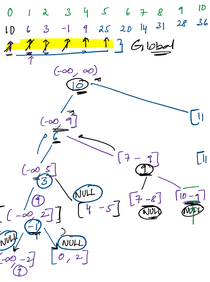
T(N) = T(0) + T(N-1) + logN

T(N) = T(N-1) + logN → using substitution

T(N) = logN + logN-1 + T(N-2)

T(N) = logN + logN-1 + logN-2 + T(N-3)

T(N) = N logN




```
int find = 0;
Node* preBST(int pre[], int l, int r)
    if(l > r){return NULL}
    if(pre[find] < l || pre[find] > r){ // out of range
        return NULL
    }
    Node *root = new Node(pre[find])
    {find++; int val = pre[find]; } Be very careful global variables
    root->left = preBST(pre, l, val-1)   (k)
    root->right = preBST(pre, val+1, r)
    return root;
}
```

→O(1)

T(N) = T(k) + T(N-k-1) + O(1)

Best Case: k = N/2

T(N) = 2T(N/2) + O(1)

T(N) = O(N)

k = 0

T(N) = T(N-1) + O(1)

T(N) = O(N)

?

Merge in Merge ( N ≤ 50, A[i] ≤ 10^9 )

Given N Elements check if there exists a subset with given sum.

→ Bit Manipulations

TC: $2^N * N$

→ Back Tracking

TC: $2^N$

→ Dynamic Progr

TC: N * (Sum)

N/2 N/2

Generate all subset sum & store

TC: $2^{N/2}$ SC: $2^{N/2}$

// Search for sum-x

Generate all subsets one by one by N

1) // Insert all subset sum of part A in a set

TC: $2^{N/2}$ SC: $2^{N/2}$

2) Now for each subset sum of part B

TC: $2^{N/2}$ [O(1)]

a) Search for sum-x in part A

Overall TC: $2^{N/2} + 2^{N/2}$ SC: $2^{N/2}$

Given an array check how many pairs a[i]+a[j] = K

& a single a[i] can be swapped to multiple variables?

a+b

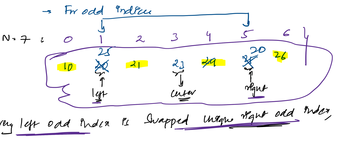
All pairs sum → O(N^2)

1:15 PM

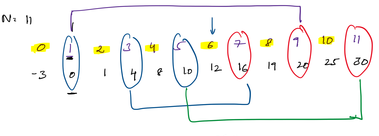
365) We are given an array → from a sorted array

→ All even indices of a sorted array are left as it is

→ For odd indices




Every left odd index is swapped unless right odd index, only case




→ Linear Search : TC: O(N)

Input[] k → Search if k is present or not


1) Can we modify our BS such that it will be apply only on even indices : TC = O(log N)

What if element is present in odd index? → Is 20 present at even index? ✗

→ 20 present at odd index

Search we are trying to search is 20 → 20 not present at ptr1 current odd index

→ Search for mean element less than 20?

→ It should be present in even index : find = 8

→ Ideally what should be index for 20 without swapping = 9 (index)

→ Value present at find+1, ar[find]

If (ar[find+1] > k) {

// k is right side

[find+2, N-1) search for ar[find] ideal position

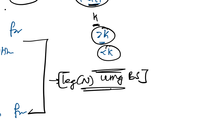
}

else

k is at left side

[0, find] search for ar[find] ideal position

}




{log(N) using BS}