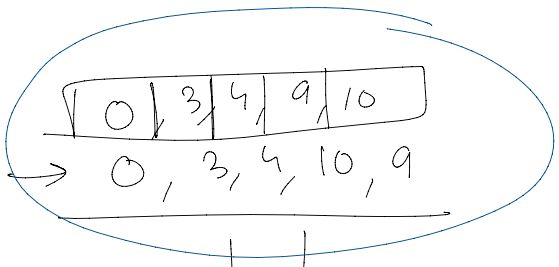
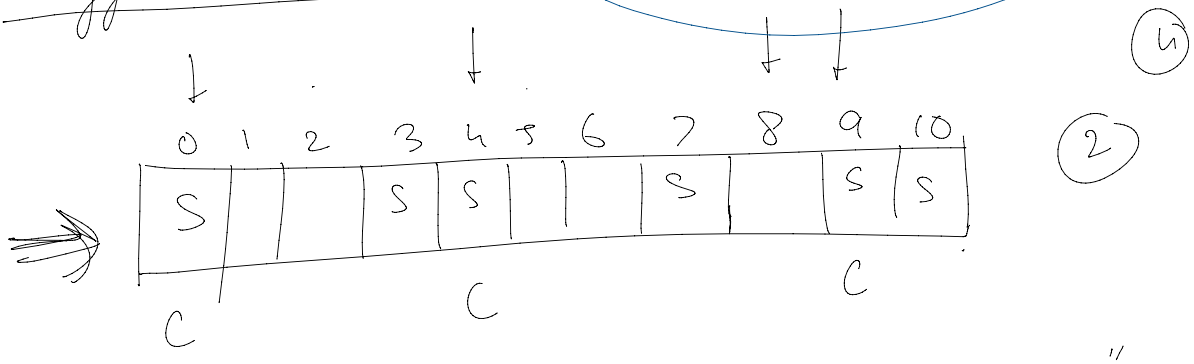
13 May 2024 21:30

> Binary search on a search space

> Recursion / Backtracking



Aggressive Cows



"maximize minimum distance between any 2 cows" → x

```
sort (stalls.begin(), stalls.end());
ans = -1;
e = 10; s = 0, m;

while(s <= e) {
    m = s + (e-s)/2;

    bool pos = IsPossible(stalls, k, m);

    if(pos == false) {
        e = m-1;
    }
    else {
        ans = m;
```

s	e	m
0	10	5
0	4	2
3	4	3
4	4	4

4 3

3

```
        s = m+1;
    }
}
```

③

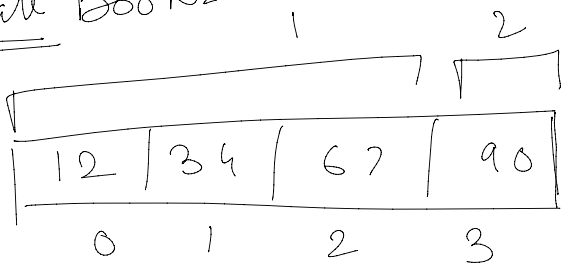
```
bool IsPossible(vector<int> stalls, int k, int m) {
    int dis = 0, last_stall = stalls[0];
    k--;
    for(int i=1; i<stalls.size(); i++) {
        dis = stalls[i] - last_stall;
        if(dis >= m) {
            k--;
            last_stall = stalls[i];
        }
    }
    if(k<=0) {
        return true;
    }
    return false;
}
```

[0, 3, 4, 7, 9, 10]

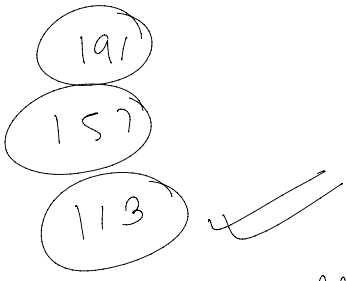
m = 3
k = 4 3 2 1 0
ls = 0 3 7 10

T: O(n log n)
S: O(1)

Allocate Books



(2)



minimize the maximum number of pages allocated to a student → x

```
if( A.size() < B) { return -1;
ans = -1;
s = 0 , e = Sum of pages, m

while(s<=e) {
    m = s + (e-s)/2;

    bool pos = IsPossible( A, B, m);

    if ( pos == true) {
        → ans = m;

        → e = m-1;

    }
    else {
        s = m+1;
    }
```

```
    }
}
return ans; }
bool IsPossible(vector<int> A, int B, int m) {
    int curr_assign=0;
    for(int i=0; i<A.size(); i++) {
        if(curr_assign + A[i] > m) {
            curr_assign = A[i];
            B--;
        }
        else {
            curr_assign += A[i];
        }
    }
    if(B<=0) {
        return true.
    }
    else {
        return false;
    }
}
```

(12) (40)

[12, 35, 67, 90]

(3) Possibility

3

Recursion / Backtracking → (1) Enumeration (2) Optimization (3) Pruning

> A function calling itself is called recursion.

Generate all Parantheses II

1) At any point, no. of closing brackets <= no. of opening brackets.
2) no. of closing brackets = no. of opening brackets.

~~A)~~ ((( )))

~~B)~~ (( )) ( )) (

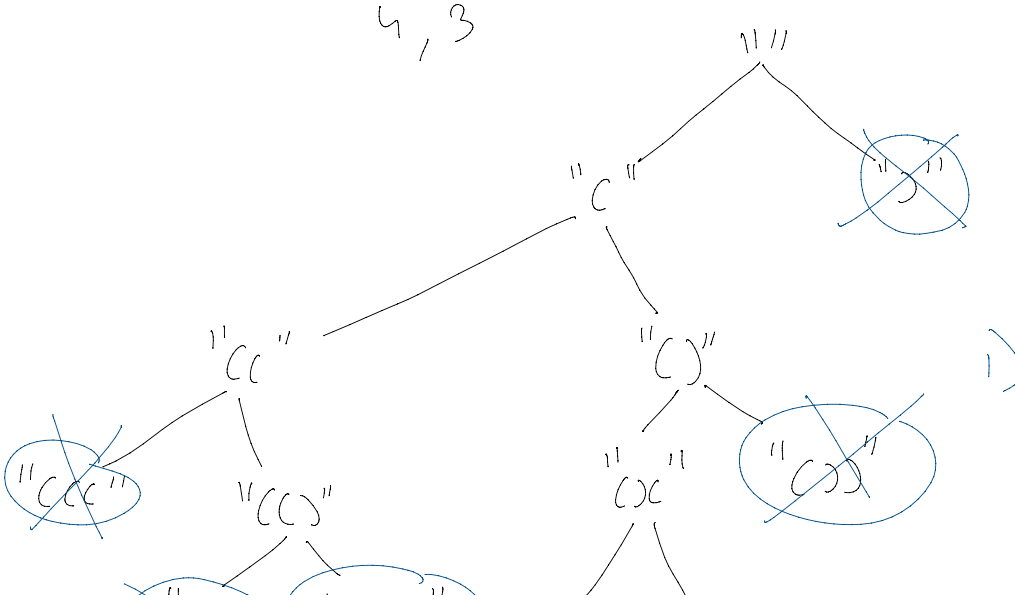
3 , 4

~~C)~~ ( ) ( ) (( )

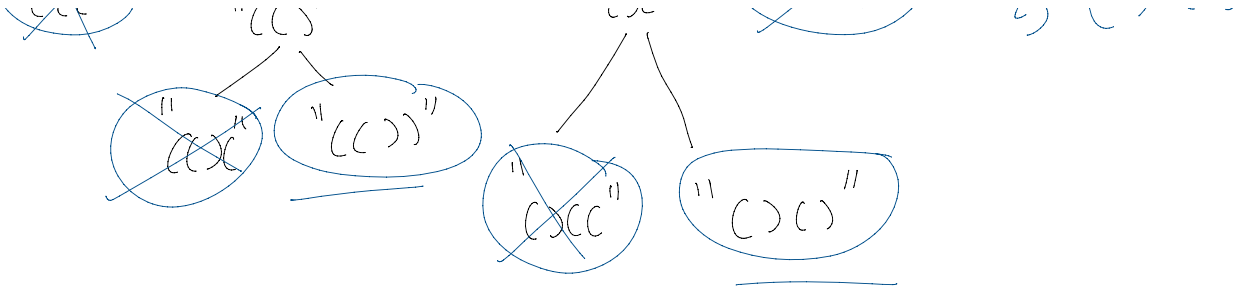
4 , 3

A = 2

Prune



1) "(( ))"
2) "( ) ( )"



vector<string> generateParanthesis(int A) {

Create vector of string ans.

string S = "";

→ recur(ans, A, S, 0, 0);

return ans;

}

recur(ans, A, S, op, cl) {

Stopping condition

if(op == A && cl == A) {

add S to ans;

close;

}

if(op > A || cl > op) {

close;

}

3

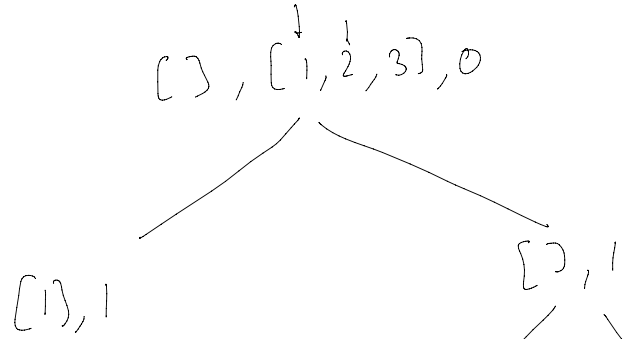
```
recur(arr, A, S+"(", op+1, cl);
recur(arr, A, S+")", op, cl+1);
```

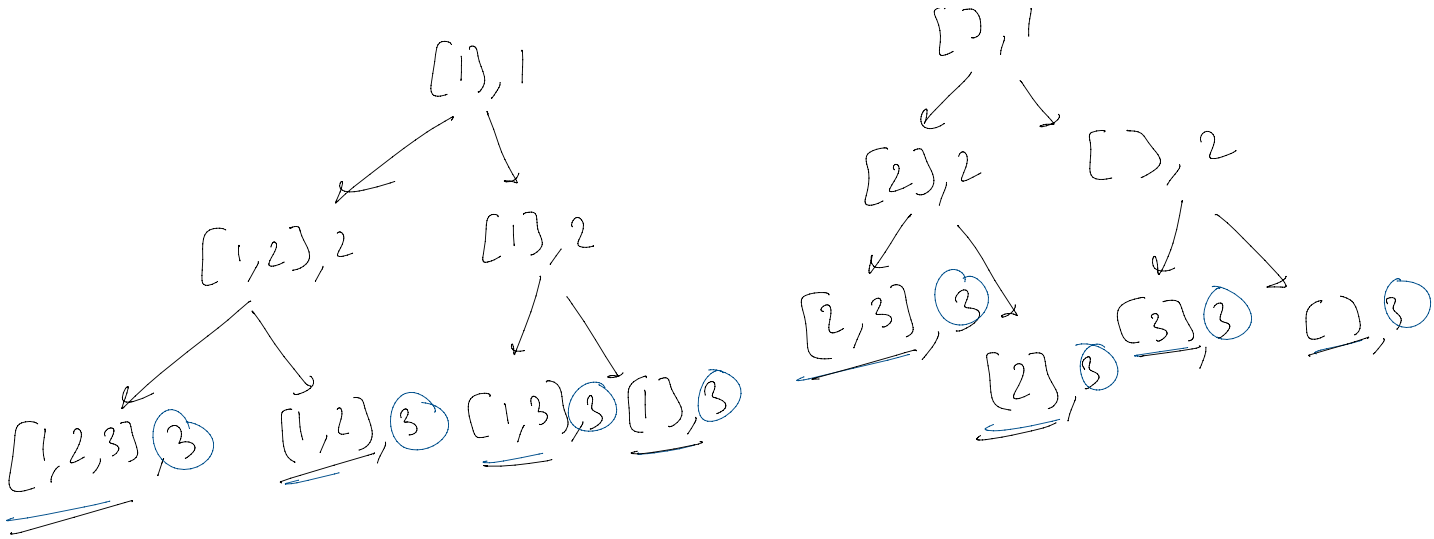
3

Subset

A = [1, 2, 3]

[[ ]
[1]
[1, 2]
[1, 2, 3)
[1, 3)
[2]
[2, 3)
[3]
]






```
vector<vector<int>> subsets(vector<int> A) {
    vector<int> vec;
    vector<vector<int>> ans;
    sort(A);
    recur(ans, vec, A, 0); Sort(ans);
    return ans;
}
recur(&ans, &vec, A, ind) {
    if(ind == A.size()) {
        add vec into the ans;
    }
}
```

Backtracking

```
{
    rec.push_back(A[ind]);
    recur(ans, rec, A, ind+1);   -> with
    rec.pop_back();
    recur(ans, rec, A, ind+1);   -> without
}
```

```
{
    recur(ans, rec, A, ind+1);   -> without
    rec.push_back(A[ind]);
    recur(ans, rec, A, ind+1);   -> with
}
```

[1, 2, 3]