1- Assign cookies : s[j] >= g[i]

g = [1,2,3] (3 children) s = [1,1] (2 cookies) } output : 1 child

g = [1,2] (2 child) s = [1,2,3] (3 cookie) } output : 2 children

1 2 3 ↑ ↑ i i   1 1 ↑ ↑ ↑ j j j   cnt = 1

Greedy
→ sort g(children's greed) △ O(nlogn)
→ sort cookies △ O(mlogm)

g: 0 1 2 (n) → 1 2, ↑ ↑ ↑ i i i   s: 0 1 2 3 (m) → 1 2 3, ↑ ↑ ↑ j j j   cnt = 1 2

```
while(i<n && j<m){                       O(n+m)
    if(s[j]>=g[i]) cnt++; i++; j++;
    else j++;
}
```

2- Fractional knapsack

val = [60, 100, 120]
wt = [10, 20, 30]
capacity = 50
} out: 240

50-30 = 20   4 120/30 × 20 = 80

60+100+80 = 240

val 60 100 = 160
wt 10 20
c 50

10 20 30 = 60
5 10 15
100

Base case:
Break & add wt
Pick
Not Pick

val 8 2 10 1 9 7 2 6 4 9
wt 10 1 7 7 5 1 8 6 8 7
21

```
f( val, wt, c, i ){

    if( c < wt[i] ){
        return [val[i]/wt[i] × c];
    }

    int Pick = val[i] + f( val, wt, c-wt[i], i+1);
    int notPick = 0 + f( val, wt, c, i+1);
    return max(Pick, notPick);
}
```

$\frac{9}{7} \times 5$

val	wt
10	7
9	5
9	7
8	10
7	1
6	6
4	8
2	1
2	8
1	7

21 → 21-7 14 → 14-9 5
ans 10, 19, 29 — 8

val	wt	
7	1	-1
2	1	-2
9	5	-7
6	6	-13
10	7	-20
9	7	
1	7	
4	8	
2	8	
8	10	

34

$\frac{9}{7} \times 1 = 1$

21, 1, 34

for 1 weight ⇒ val/wgt
write comparator sort based on

120 100 60
30 20 10
4 5 6

7 1 1 2
9 − 2 −
19 10 7 9
28 9 5 14
34 6 6 20
9 7 27

7/2

9/7 = 1.28

priorityqueue
35

pairs (val, wt) ⇒ val/wt ⇒ desc ⇒ Sort
iterate over pairs
capacity < p.second
ans += (F/S) * C
else
ans += p.f c = c - p.s;

```
bool static customComparator(
    Pair<int,int> P1, pair<int,int> P2){
    return ((double) P1.first / P1.second) >
           ((double) P2.first / P2.second);
}
```

3. coin change - Min coins to make sum   NO. of ways its a DP problem

coins = [25, 10, 5]   sum = 30

minimum 2 coins ⇒ 25, 5

c = 4, 6, 2   S = 5   Not possible -1

sort (coins) ⇒ (desc) 25, 10, 5   sum = 30

```
while i<n:
   if (sum > coins[i])
       sum -= coins[i];
   else
       i++;
```

O(sum) O(1)

coins = 9, 6, 5, 1   sum = 19

minimum coins = 3 (9+9+1)

c = 5, 1   S = 0   output = 0

recursive way:

```
f(i){ if sum==0 return 0;
  if (sum<0 || i>=n) return INT_MAX;
  pick = INTMAX;
  int pick = f(i , sum - coins(i))
  if (pick != INT_MAX) pick++;
  int notpick = 0 + f(i+1, sum);
  return min (pick, notpick);
}
```

4. Lemonade change

bills = 5 5 5 10 20   (100%)

```
if ( bills[i]==5) five_dollars -= 1;
else if (bills[i]==10) {
    if ( five_dollars<=0) return false;
    five_dollars--;
    ten_dollars++;
}
else {
    if ( ten_dollars >=1 && five_dollars >=1) {
        ten_dollars -=1;
        five_dollars -=1;
    }
    else if ( five_dollars >=3){
        five_dollars -=3;
    }
    else return false;
}
return true;
```

O(b) O(1)

5. Valid Parenthesis String

'(' ')' '*'
✓ ✓ ↳) or (or ""

(= 1
) = -1
* ⇒ 0 1 -1

S = () true   S = (*)) true

S = (*) true   +0 -1 } add = 0   S = 1

My thinking process is good

cnt (⇒ +1) ⇒ -1 is good

try to solve simple (*)

() (() ())

solve using rec

```
f(i, ans){
  if (i==n) ←
    return ans==0
  if s[i]='('
      f(i+1, ans+1)
  else if s[i]=')'
      f(i+1, ans-1)
  else if s[i]= *
      return f(i+1, ans) || f(i+1, ans+1)
         || f(i+1, ans-1);
}
```

change to ans==1 for tabulation

100% in Tabulation

DP ↓   TC→O(N²)   SC O(N)

Maintain the range min & max

* → +1, → 0, → -1

invalid ⇒ -1   {min, max}

so carry only valid range 0 →   trim to range 0   → is -1 false

min =
max =

6. N meetings in 1 room

N=6

S	1	3	0	5	8	5
e	2	4	5	7	9	9

→ sort based on start time.

print max no. of meetings & their order

S	0	1	3	5	5	8
e	5	2	4	7	9	9

5 1 1 2 4 1

end time sorting & iterate from 2nd meeting

we can always perform 1st meeting

```
ans=1
vector < pair<int,int>> p;
for(i=0 → n-1){
    p.push(s[i], e[i]);
}
Sort(p.begin(), p.end(), customComparator);
for(j=1 → n-1){
    if(p[j].first > endTime){
        ans++;
        endTime = p[j].second
    }
}
return ans++;
```

```
customComparator(p1 p2){
    return p1.s < p2.s
}
```

7. Jump Game

nums = 2, 3, 1, 1, 4 (0 1 2 3 4) max jumps

↑ Return true if we can reach last index else false

2 3 1 1 4 (0 1 2 3 4)

1 4 3 4 8

3 2 1 0 4 (0 1 2 3 4)

3 3 3 3

```
maxReach=0
for(i=0 → n-1){
    if(maxReach >= n-1)
        return true;
    if(maxReach < i)
        return false;
    maxReach = max(i+nums[i], maxReach);
}
return false;
```

0 1 2 3 4
2 3 1 1 4
i
curr 2 4 3 4 8
maxR 2 4 4 4 4

0 1 2 3 4
2 3 0 1 4

2 4
2 4 → calculate maxR & NO. of jumps (counting times we changed maxR)

8. Jump Game II return min no. of jumps

nums = 2, 3, 1, 1, 4 out=2

try all jumps

last index is always reachable

0 1 2 3 4
2 3 1 1 4
(i) ↗

i+2, 1→ 0+2=2

```
f(i){
    if(i>=n-1) return 0;
    mini = INT_MAX;
    for(j=i+1 → nums[i]+i){
        mini = min(f(j), mini);
    }
    return mini;
}
```

O(N²)
O(N)

Tabulation beats 22.09%.

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
[1 0 9 6 9 6 1 7 9 0 1 2 9 0 3]
i

1 1 11 9 13 11 7 14 17 9 11 13 21 13
↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑

i == maxR

maxR = 0 7 14 currR = 0 7 11 13 14

Jumps = 0 1 2 4

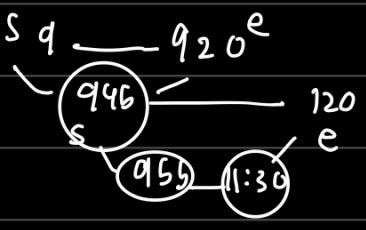
```
maxReach = currReach = jumps = 0;
for(i=0 → n){
    if(maxR >= n-1) return jumps;
    currReach = max(nums+(i), currReach);
    if(i == maxR){
        jumps++;
        maxR = currR;
    }
}
```

O(n)
O(1)

9. Min no. of platforms req for a railway

N = 6 arr = {9:00, 9:45, 9:55, 11:00, 15:00, 18:00}

dep = {9:20, 12:00, 11:30, 11:50, 19:00, 20:00}



so calculate the no. of clashes ⇒ that many tracks

```
for(i=0→n-1){
    for(j=i+1→n-1){
        if(arr[i] ≤ arr[j] ≤ dep[i] || arr[j] ≤ dep[i] ≤ dep[j])
            return true;
    }
}
return false;
```

T ⇒ O(N²)
S ⇒ O(1)

```
Checkclash( ){
    if(checkclash(arr, dep, i, j)){
        cnt++;
    }
}
```

O(n log n) + O(n)
O(1)

Efficient App

0	1	2	3	4	5
900	945	955	1100	1500	1800
920	1130	1150	1200	1900	2000
1	2	3	4	5	

sort both array

just calculate at every step how many platforms needed by using 2 pointers

currplatefors
maxplatforms

i=1 j=0 currp=1 maxp=1

```
while(i<n){
    if(arr[i] < dep[j]){
        currp += 1; i++;
    }
    else{
        currp -= 1; j++;
    }
    maxp = max(currp, maxp)
```

10. Job sequencing problem

N×3 Jobs = [[1,4,20], [2,1,10], [3,1,40], [4,1,30]]

Jobs, Profit — JobID, deadline, profit

Out: 2 60

at t=1

```
vector<int>
customcomparator(v1, v2){
    return v1 < v2;
}
```

cnt_jobs = 0, profit = 0

```
sort()
for(i=1→n-1){
    if(Jobs[i][1] >= i){
        cnt_jobs += 1;
        profit += Jobs[i][2];
    }
}
return {cnt_jobs, profit};
```

currtime not works

at t=1 may be 2 and then 2 2

	t	
3 1 40	→ 1	
4 1 30	→ 2	60
2 1 10	→ 3	
1 4 20	→ 4	

to maximize profit sort by profit first

N = 5

1	1	2	100
2	3	2	27
3	4	1	25
4	2	1	19
5	5	1	15

there is one more approach.

[-1, -1, -1, -1, -1]
1 2 3 4 5

v1[2] > v2[2] profit.

```
timelines(n, -1);
for(i=0→n){
    job_time = Jobs[i][1];
    while( ){ } get right index
    if(job_time >= 0){
        cnt++; profit += Jobs.
    }
}
```

11. candy n children in a line return min no. of candies to distribute
Each child has a rating value ratings

rating = 1, 0, 2 .3 children give candies
- each child has atleast 1 candy
- children with higher ratings get more candies than their neighbors

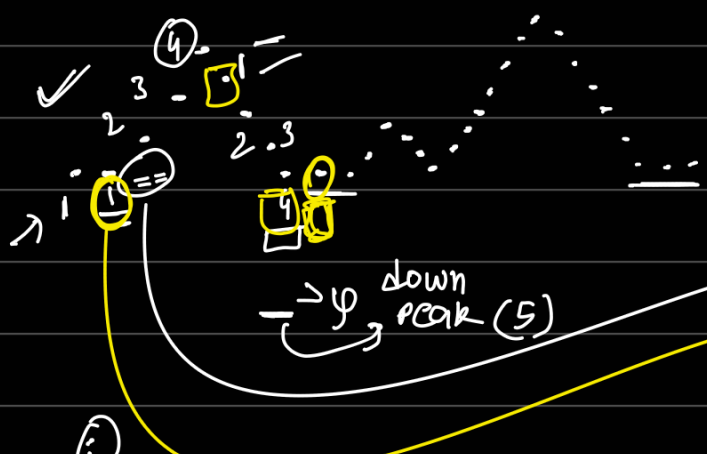
1st 2nd 3rd
2 1 2 candies

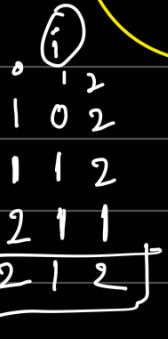
1 2 2
1st 2nd 3rd
1 2 1 candies

left → see left elements & increments
right ← see right elements & increment
Increment if elements are increasing & assign 1 if it decreases & same level also equal to prev one

optimal approach

[uppeak]



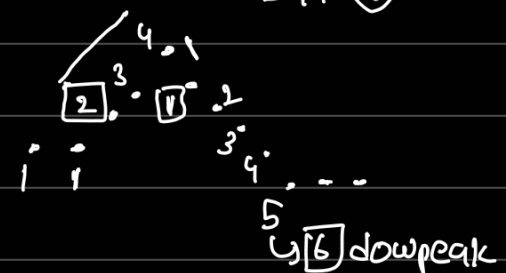


uppeak = 1

1 0 2

1 0 2

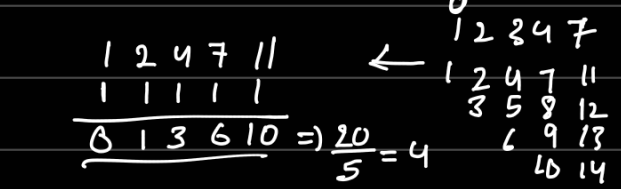
1 + 2 + 1 = 4 (d > 0) +1



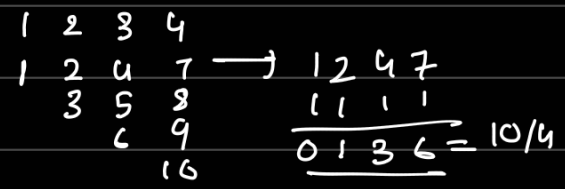

```
while (i<n){      i=1, sum=1,
                  ↓ first ele
//flat
while (arr[i] == arr[i-1]){
    sum += 1; i++;
}
//uppeak ↗4
int uppeak = 1;
while (arr[i-1] < arr[i]){
    uppeak += 1; → (2)
    sum += uppeak; i++;
}
int downpeak = 0;
while ( arr[i-1] > arr[i]){
    downpeak += 1;
    sum += downpeak;
    i++;
}
if (downpeak+1 > uppeak){
sum += (downpeak+1) - uppeak;
}
return sum;
```

12. Shortest Job first

n = 5 bt = burst time = [4, 3, 7, 1, 2]



n = 4 bt = [1 2 8 4] 10/4 = 5/2 = 2



n=5 bt=[3 1 4 2 5]

Calculate wt is prevwt + prevST

like 3+3=6 for 4

1 2 3 4 5
$\frac{20}{5}$ = 4 ← 0 1 3 6 10
2 4 7
5 8
9

maxheap

priority_queue<int> pq(a, a+n);

sort(a, a+n); currwt=0, wt=0; prevwt=0

```
for( i=1 → n-1){                      T O(nlogn) + O(n)
  prevwt = prevwt + a[i-1];           S O(1)
  wt += prevwt;                       S O(n)
}
return wt/n;
```

13. Least Recently Used

7 0 1 2 0 3 0 4 2 3 0 3 2

find page fault

PF

map

→ 0:7 ~~0~~ 3
1:0
2:1
3:2

set
<7 0 1 2>

14- Insert Interval

intervals = [[1,3],[6,9]]
newInterval = [2,5]
output: [[1,5],[6,9]]

4 ——— 8
[[1,2] [3 5] [6,7] [8,10] [12,16]]
[4,8] overlaps with 3 intervals.
[[1,2] [3,10] [12,16]]

[1,2] [3 5] [6,7] [8,10] [12,16] / (4,8)
0 1 2 3 4
↑ ↗ ↗ ↗ ↗
i

new[0]
new[1]

14
✓ 4 8
4 8 ✓

i[0] < 4 <

i[1] new[0]
2<4
5<4 ✗ ⇒
6<4

new[1] i[0]
8 < 6 ⇒
8 < 8
8 < 12 ✓

8<5 ✗ ⇒

new[1] < i[0]

```
if (i[1] < new[0]){
    add it
}
else if (new[1] < i[0]){
    insert newIn() & add new;
}
else {
  In new[0] = min(Innew[0], i[0]);
  In new[1] = max(Innew[1], i[1]);
}
```

(1,2) (3,5) (6,7) (8,10) (12,16)
(4,8)

i: 12 | n: 48 — i[1] < n[0] pass
i: 35 | n: 48 — i[0] < n[0] && n[0] < i[1] overrlapping
67 | 48 — n[0] < i[0] && n[0] < i[1] && i[1] < n[1] overrlapping
8 10 | 4 8 — n[1] <= i[0] && n[0] <= i[0] overlapping
12 16 | 4 8

15- Merge Intervals

[(1,3], (2,6), (8,10), (15,18))

currInt — Iterate

(0 0) | (0 0)

currInt | i

sort it

update currInt else No overlap push to ans vector

```
vector<vector<int>> ans;
ans.push_back(intervals[0]);
for (i=1 -> n-1){
    if (intervals[i][0] < ans.back()[1]){ //overlap
        ans.back()[1] = max( intervals[i][1], currInt[1]);
    }
    else{
        ans.push_back(intervals[i]);
    }
}
```

i = (2,6) (8,10) (15,18)

ans = ((1,6), (8,10), (15,18))

return ans;

16- Non-Overlapping Intervals

min no. of Intervals to remove to make the rest of intervals non-overlapping

Intervals = ((1,2), (2,3), (3,4), (1,3))

out = 1

(1,2) (1,3) (2,3) (3,4)

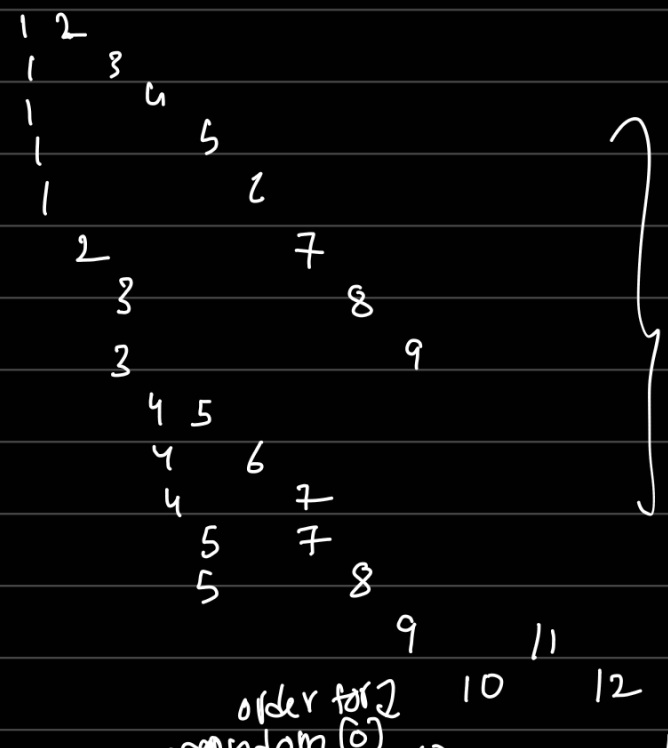
min so keep it — 3>2 so remove it as it overlaps

(1,2) (1,2)(1,2) out = 2

(1,2) (2,3) out = 0

(1,2) (1,3) (1,4) (1,5) (1,6) (2,7) (3,8) (3,9) (4,5) (4,6) (4,7) (5,7) (5,8) (9,11) (10,12) (13,14) (15,16)

(1)



How many meetings we can do in a single room

Sort by end time

Iterate over one by one & if they overlap don't count them else count them

```
ans++;
```

order for 2 random (0)

(1,2) (1,3) (1,4) (1,5) (4,5) (4,6) (1,6)

(5,7) (4,7) (2,7) (5,8) (3,8) (3,9)

(9,11) (10,12)

(13,14)

(15,16)

17 - 16 = 11