Today's Content :-

Pairs with given sum - 2
Pairs with given diff.
Subarrays with given sum
Container with most water.

Today's Quote :-

The more you sweat in peace,
the less you bleed on war.

Ques :- Given a sorted integer array A & an integer k, find any pair (i,j) s.t.,

A[i] + A[j] = k & i != j,

S.C -> O(1).

```
     0   1   2   3   4   5   6
A = [-5, -2,  1,  8, 10, 12, 15]   k = 11
     i           j
```

1) Brute force :- Check all pairs.

T.C -> $O(n^2)$, S.C -> O(1).

2) Binary Search. A[i] + A[j] = k.

A[j] = k - A[i]

∀ i, Binary Search, k - A[i], s.t. i != j,

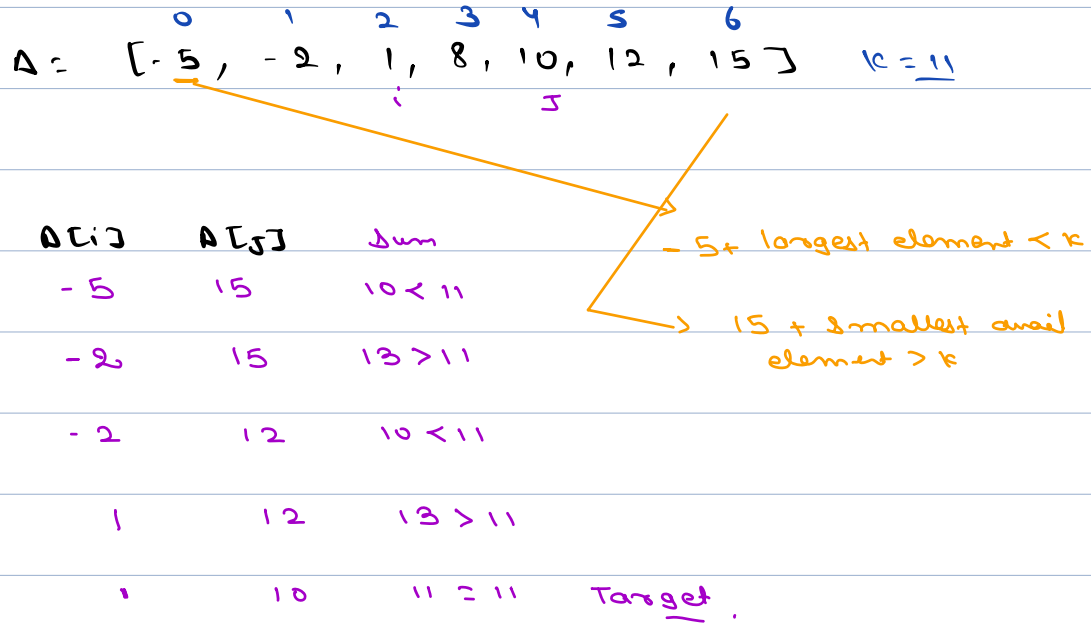
T.C -> O(nlogn)

S.C -> O(1).

```
     0   1   2   3   4   5   6
A = [-5, -2,  1,  8, 10, 12, 15]   k = 11
     i   j               i   j
```

arr[i] + arr[j]
-5 + (-2)
-7 < 11 X
i++, j++

arr[i] + arr[j]
12 + 15
27 > 11 X
i-- j--



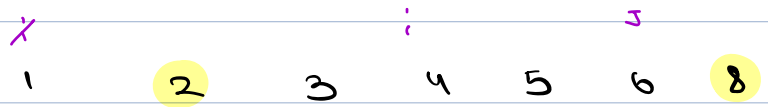
A[i]	A[j]	sum	
-5	15	10 < 11	
-2	15	13 > 11	
-2	12	10 < 11	
1	12	13 > 11	
1	10	11 = 11	Target.

```
i = 0, j = n-1;
while (i<j) {
    if (A[i]+A[j] == k) {
        return true;
    }
    else if (A[i]+A[j]<k) { i++ }
    else { j-- }
}
return false.
```

T.C → O(n)

S.C → O(1).

Ques find all the pairs, in distinct sorted array.

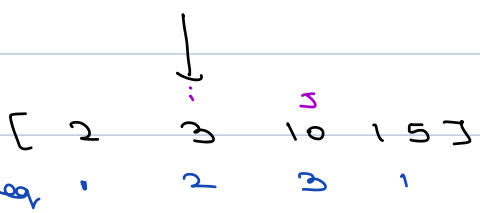


K = 10.

same as prev,
when arr[i]+arr[j] == k,
cnt++;
i++ & j--

duplicates

[2 3 3 10 10 10 15]



sum = 13

Create a distinct array & store freq of every element, whenever find a pair ∈ sum = k, in distinct array multiply the frequencies.

K = 14

[2 4 4 4 5 5 7 10 10 10 15]

i = 0, j = n-1

ans = 0;

T.C → O(n)

S.C → O(1).

while (i < j) {

sum = A[i] + A[j]

if (sum < k) {

i++

}

else if (sum > k) {

j--

}

else {

ii = i, jj = j

if (A[i] == A[j]) {

cnt = j - i + 1;

ans += cnt * (cnt-1) / 2;

break;

}

else {

while (A[ii] == A[i]) {

ii++

}

cnt1 = ii - i

i = ii

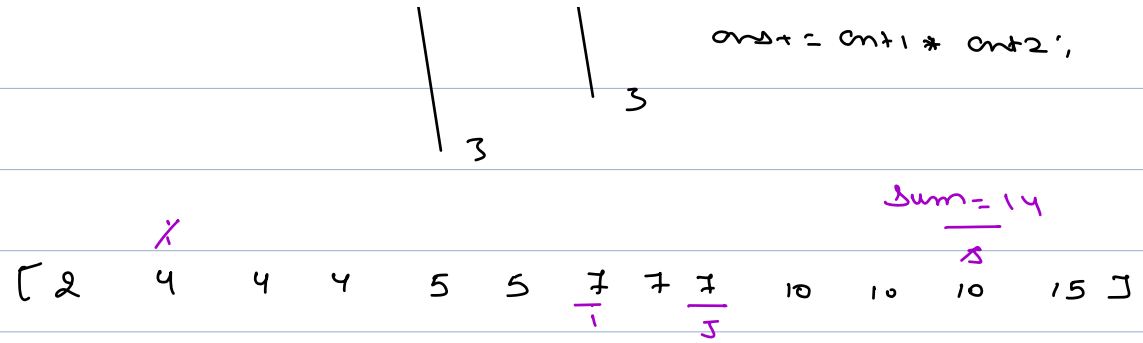
while (A[jj] == A[j]) {

jj--

}

cnt2 = j - jj;

j = jj;



Ques :- Given a **sorted** integer array A & an integer k, find any pair (i,j) s.t.,

$A[j] - A[i] = k$ & $i != j$.

T.C → O(1).

$k > 0$.

A = [-5, -2, 1, 8, 10, 12, 15] k = 11.

(0 1 2 3 4 5 6)

i, j

Brute force :-

(1) check all pairs $O(n^2)$.

(2) **Binary Search**

arr[j] - arr[i] = k

↓

for other i → arr[j] = k + arr[i]

no' do binary search.

idea :- Two pointers.

	0	1	2	3	4	5	6
A =	-5	-2	1	8	10	12	15

k = 11.

i, j

arr[j] - arr[i]

15 - (-5)

=> 20 > 11

This will work.

	0	1	2	3	4	5	6
A =	-5	-2	1	8	10	12	15

k = 11.

i, j

arr[j] - arr[i]

15 - 12

=> 3 < 11

15 - 10

5 < 11

15 - 12

=> 3 < 11

longest element - A[i] < k

any elem - A[i] < k

	0	1	2	3	4	5	6
A =	-5	-2	1	8	10	12	15

k = 11.

i, j

A[j]	A[i]	diff
-2	-5	3 < 11
1	-5	6 < 11
8	-5	13 > 11
8	-2	10 < 11
10	-2	12 > 11

↑ y - x < k

y - x > k

```
i = 0, j = 1
while(j < n) {
    diff = A[j] - A[i];
    if(diff == k) {
        return (i, j);
    }
    else if (diff < k) {
        j++;
    } else {
        i++;
    }
}
```

Ques Given an integer array of +ve elements and an integer k. check if there exists a subarray with sum = k.

```
          0   1   2   3   4   5   6
A : [ 1   3   15  10  20  3   23 ]
```

k = 33. ✓

k = 43, ✗

Brute force Check all subarray sums.
↳ carry forward
T.C → O(n²)
S.C → O(1)

idea 2 :-

k = 33.

```
          0   1   2   3   4   5   6
A : [ 1   3   15  10  20  3   23 ]
pf: [ 1   4   19  29  49  52  75 ]
          ↑               ↑
```

Subarray

sum(i,j) → pf[j] - pf[i-1] i > 0
pf[j] i = 0 ✓

check pf[j]

↓

checks all subarrays starting with i=0.

if pf[j] − pf[i−1] = k

s(i−j) = k

new problem → pair with diff k.

Todo, S.C → O(1), without changing i/p array.

idea 3 :-

sum = ~~9~~ ~~29~~ ~~49~~ ~~48~~ ~~45~~ 30 33 k = 33

	0	1	2	3	4	5	6	7	8
A:	1	3	15	10	20	3	23	33	43
	i	i	i	i	j	j			

k = 13

i=0, j=0, sum = A[0];

while (j < n) {

if (sum == k) { return true }

else if (sum < k) { j++;

if(j==n) { break }

sum += A[j];

}

else {

sum -= A[i]

i++;

}

if (i > j & i <= n-1) {

j++

}

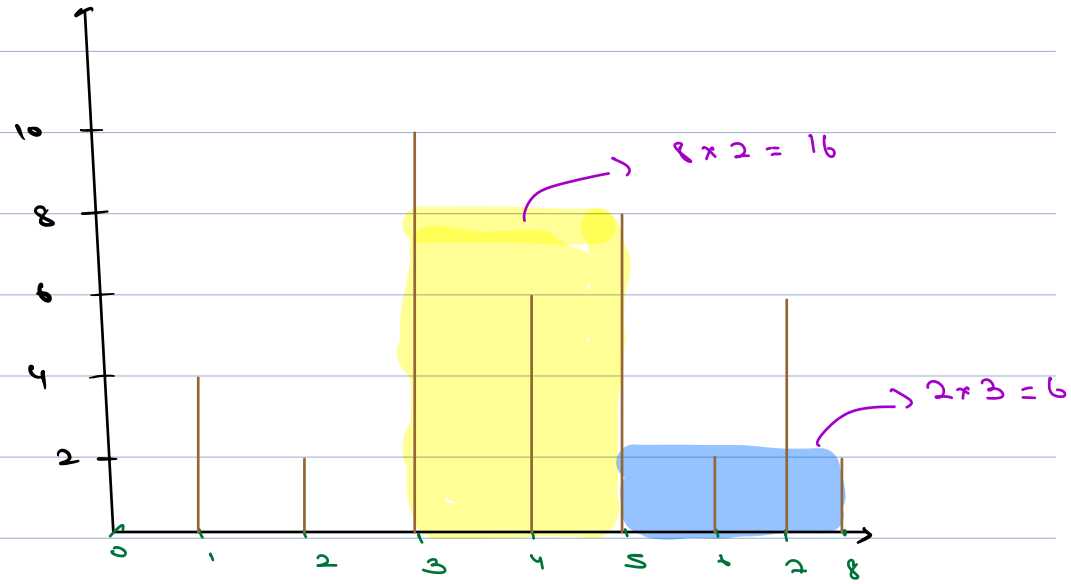
T.C → O(n)

S.C → O(1).

```
    sum += arr[3];
}
```

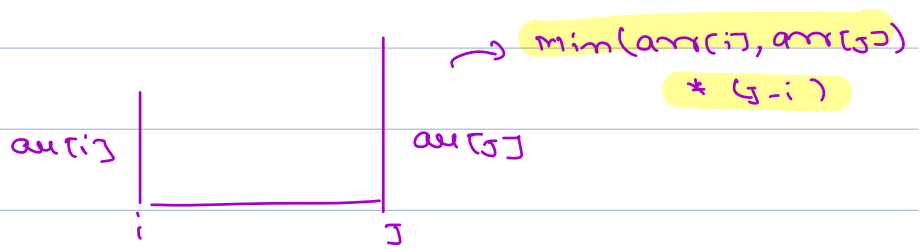
Containers with most water.

Ques.



A [] → [4, 2 10 6 8 2 6 2]

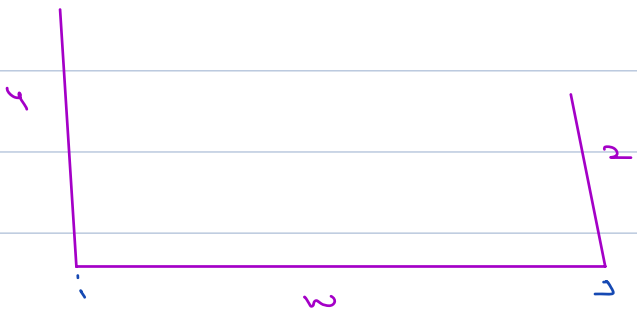
find two walls that can form a container to store maximum water.



A [] → [4, 2 10 6 8 2 6 2]
0 1 2 3 4 5 6 7
i
j

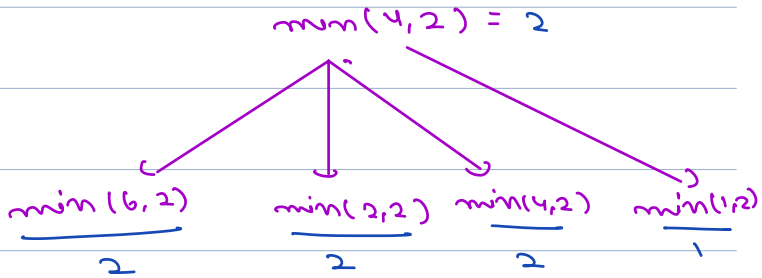
arr[i]	arr[j]	water stored
4	2	14
4	6	24
2	6	10
10	6	24

10	2	6
10	8	16
10	6	6



w x 2.

height & width ↓



T.C -> O(n)

S.C -> O(1).

idea :- Always move the smaller wall.

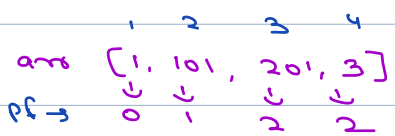
```
i = 0, j = n-1
while (i<j) {
    area = Max (area, Min(a[i], a[j]) * j-i)
    if (A[i] < A[j]) {
        i++;
    else if (A[i] > A[j]) {
        j--
    }else {
```

int en j--;

}

Sophie is a nutritionist who is obsessed with healthy eating. She has a list **A** of size **N** foods with their respective nutritional values in the array A. Sophie gives Bob a query array **B** of size **Q**, where in each query she gives him a range from L to R and asks him to find the count of foods with nutritional value strictly greater than **10** in the subarray A[L], A[L+1], ... A[R]. Bob is busy with building a new gym, so he asks you to find the answer to all Sophie's queries.

NOTE: Queries in array B have 1-based indexing, which means that we will have to subtract 1 from the given range while getting answer



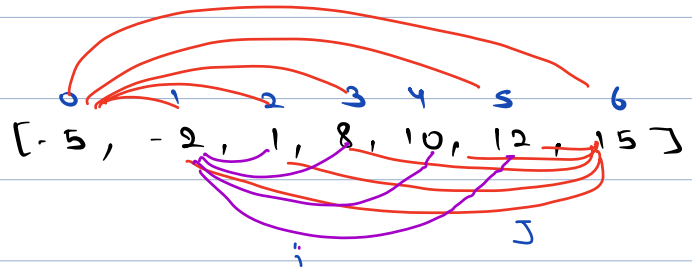
10

1, 3 → 2

1, 4 → 2

1, 2 → 1

4, 4 → 0



k = 11

10 < 11

-2 + 12 = 10

10 < 11