## Agenda

1) Pair sum
2) Pair difference
3) container with most water ***

Q.1 **Pair sum**

Given a sorted array, check if there exists a pair (i,j) such that $A[i] + A[j] = K$ $(i != j)$

A = [3 7 8 12 19] K = 19 ans = true

A = [2 5 8 9 10] K = 9 ans = false

i) go on all pairs TC: $O(n^2)$ SC: O(1)

ii) Hashset TC: O(n), SC: O(n)

iii) binary search TC: O(nlog n)

iv) Two pointer

optimal solution

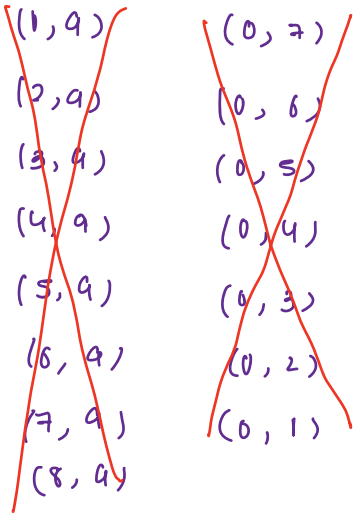
```
                i                 j
A = [-3  0  2  3  6  8  11  14  18  25]      k = 17
     0  1  2  3  4  5  6   7   8   9
```

| A[i] | A[j] | A[i]+A[j] | | |
|---|---|---|---|---|
| -3 | 25 | 22 | 22 > 17 | j-- |
| -3 | 18 | 15 | 15 < 17 | i++ |
| 0 | 18 | 18 | 18 > 17 | j-- |
| 0 | 14 | 14 | 14 < 17 | i++ |
| 1 | 14 | 15 | 15 < 17 | i++ |
| 3 | 14 | 17 | 17 == 17 | |

$A[i] + A[j] > k \Rightarrow$ j--

$A[i] + A[j] < k \Rightarrow$ i++

```
                i                                   j
A =  [-3  0  1  3  6  8  11  14  18  25]        K = 17
      0   1  2  3  4  5  6   7   8   9
```




Sum = -3+25 = 22 → 22 > 17 → j--

Sum = -3+18 = 15 → 15 < 17 → i++

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

TC : O(n)  SC : O(1)

K = 12

```
A =  [-3  0  1  3  6  8  11  14  18  25]
      0   1  2  3  4  5  6   7   8   9
              i              j
```

| i | j | A[i]+A[j] | |
|---|---|---|---|
| 0 | 9 | -3+25 > 12 | j-- |
| 0 | 8 | -3+18 > 12 | j-- |
| 0 | 7 | -3+14 < 12 | i++ |
| 1 | 7 | 0+14 > 12 | j-- |
| 1 | 6 | 0+11 < 12 | i++ |
| 2 | 6 | 1+11 == 12 | |

Q.2 **Pair difference**

Given a sorted array, check if there exists a pair (i,j) such that $A[j] - A[i] = K$ and $K > 0$ $(i \neq j)$

A = [-3 0 1 3 6 8 11 14 18 25] K = 5 true
0 1 2 3 4 5 6 7 8 9

i) go on all pairs TC: $O(n^2)$ SC: $O(1)$

ii) two pointers

A = [-3 0 1 3 6 8 11 14 18 25] K = 17
0 1 2 3 4 5 6 7 8 9
i j

$A[j] - A[i] = 25 - (-3) = 28$

i++ → diff will get decreased

j-- → diff will get decreased

X

i j

A = [-3 0 1 3 6 8 11 14 18 25]

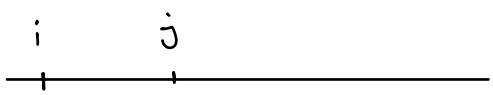
0 1 2 3 4 5 6 7 8 9

k = 5

| A[i] | A[j] | A[j] − A[i] | | |
|---|---|---|---|---|
| -3 | 0 | 3 | 3 < k | j++ |
| -3 | 1 | 4 | 4 < k | j++ |
| -3 | 3 | 6 | 6 > k | i++ |
| 0 | 3 | 3 | 3 < k | j++ |
| 0 | 6 | 6 | 6 > k | i++ |
| 1 | 6 | 5 | 5 == k | |

A[j] − A[i] > k ⇒ i++

↓




A[j] − A[i] < k ⇒ j++

↑

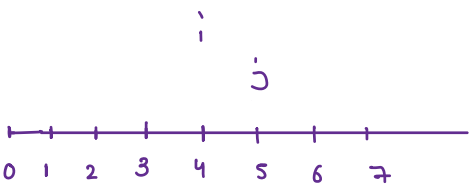
```
                  i     j
A = [-3  0  1  3  6  8  11  14  18  25]        K = 4
     0  1  2  3  4  5  6   7   8   9
```

(2,4) (3,5)
(2,5) (3,6)
(2,6) (3,7)
(2,7) (3,8)
(2,8) (3,9)
(2,9)

diff = 6 - 1 = 5  5 > K  i++
diff = 6 - 3 = 3  3 < K  j++
diff = 8 - 3 = 5  5 > K  i++

```
boolean PairDiff ( int [] A, int K) {
    int i=0, j=1;
    while ( j < A.length ) {
        if ( A[j] - A[i] == K) {
            return true;
        }
        else if (A[j] - A[i] > K) {
            i++;
        }
        else if (A[j] - A[i] < K) {
            j++;
        }
    }
    return false;
}
```

j >= i




K > 0

TC: O(n)

SC: O(1)

```
boolean PairDiff ( int [] A, int K) {
  int i=0, j=1;
  while ( j < A.length ) {
      if( A[j] - A[i] == K) {
          return true;
      }
      else if (A[j] - A[i] > K) {
          i++;
      }
      else if (A[j] - A[i] < K) {
          j++;
      }
  }
  return false;
}
```

K = 7

i j

A = [-3 0 1 3 6 8 11 14 18]
0 1 2 3 4 5 6 7 8

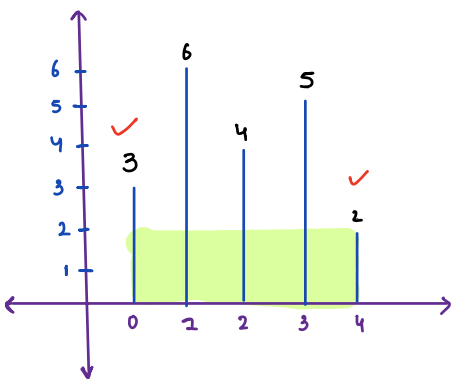
| i | j | A[j] - A[i] |
|---|---|---|
| 0 | 1 | 0 - (-3) < 7 |
| 0 | 2 | 1 - (-3) < 7 |
| 0 | 3 | 3 - (-3) < 7 |
| 0 | 4 | 6 - (-3) > 7 |
| 1 | 4 | 6 - 0 < 7 |
| 1 | 5 | 8 - 0 > 7 |
| 2 | 5 | 8 - 1 == 7 |

Q.3 **Container with most water**

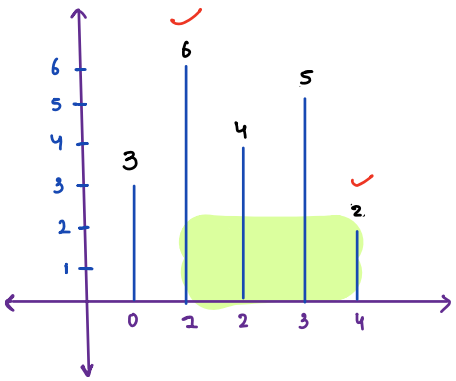
Given an array, where A[i] represents height of each wall.

**Pick any 2 walls** such that max water is accumulated blw them.

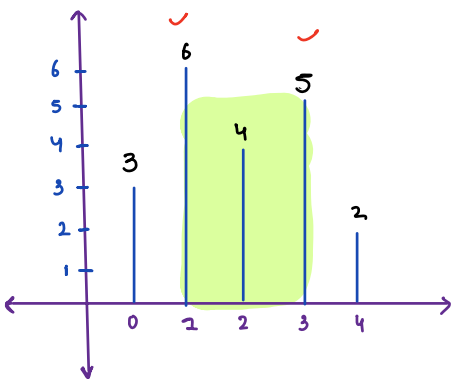
A = [3 6 4 5 2]



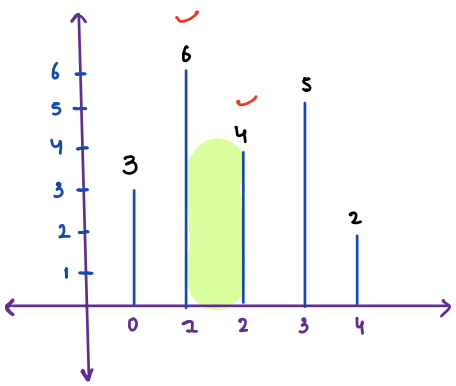

2 * 4 = 8




2 * 3 = 6




5 * 2 = 10




4 * 1 = 4

(i, j)

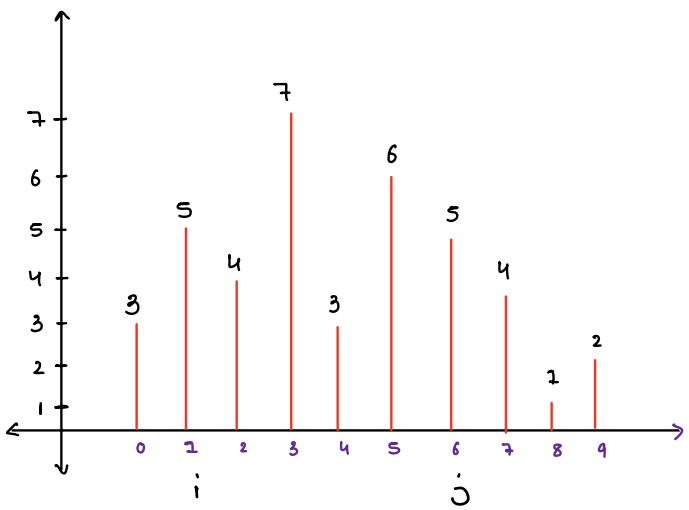
water accumulated blw walls = $\min(A[i], A[j]) * (j - i)$

h w

i) brute force, going on every pair of wall and finding best ans. TC: $O(n^2)$

ii) optimal solution.

A = [3 5 4 7 3 6 5 4 1 2]




ans = ~~0~~ ~~24~~ 25
~~18~~ ~~21~~

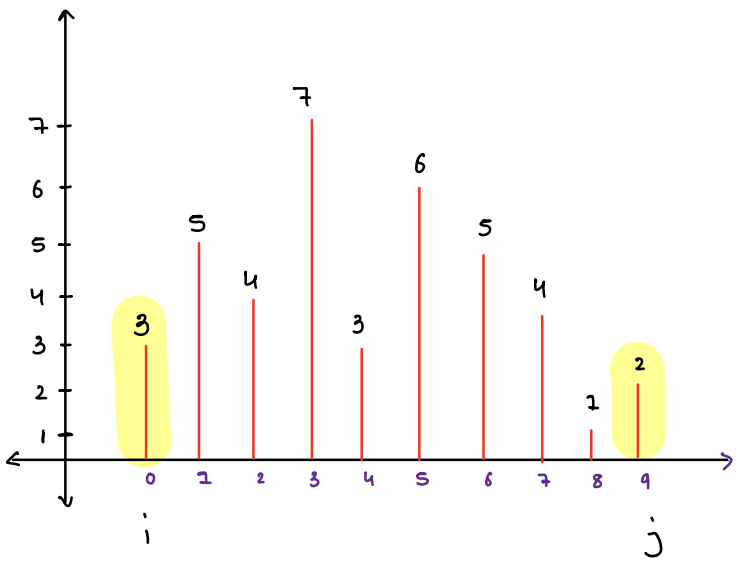
| A[i] | A[j] | h | w | temp | action |
|---|---|---|---|---|---|
| 3 | 2 | 2 | 9 | 18 | j-- |
| 3 | 1 | 1 | 8 | 8 | j-- |
| 3 | 4 | 3 | 7 | 21 | i++ |
| 5 | 4 | 4 | 6 | 24 | j-- |
| 5 | 5 | 5 | 5 | 25 | ..... |

(discarding the smaller out of A[i], A[j])

temp = min(A[i], A[j]) * (j - i)

h

w




0, 9
1, 9
2, 9
3, 9
4, 9
5, 9
6, 9
7, 9
8, 9

temp = min(A[i], A[j]) * (j - i)

Why to discard lower height

⇒ min(A[i], A[j]) * (j - i)

either same or ↓ — width ↓

```
int containerWithMostWater(int[] A) {
    int i=0, j=A.length-1;
    int ans=0;
    while (i<j) {
        int temp= Math.min(A[i], A[j])*(j-i);
        if (temp>ans) {
            ans=temp;
        }
        // discard lower height
        if (A[i]<A[j]) {
            i++;
        }
        else {
            j--;
        }
    }
    return ans;
}
```

TC: O(n)

SC: O(1)

Dry run

```
while (i < j) {
    int temp = Math.min(A[i], A[j]) * (j-i);
    if (temp > ans) {
        ans = temp;
    }
    // discard lower height
    if (A[i] < A[j]) {
        i++;
    }
    else {
        j--;
    }
}
```

ans = ~~0~~ 18 ~~24~~ ~~24~~ 25

```
A = [3 5 4 7 3 6 5 4 1 2]
    0 1 2 3 4 5 6 7 8 9
        i       j
```

| A[i] | A[j] | h | w | temp | action |
|---|---|---|---|---|---|
| 3 | 2 | 2 | 9 | 18 | j-- |
| 3 | 1 | 1 | 8 | 8 | j-- |
| 3 | 4 | 3 | 7 | 21 | i++ |
| 5 | 4 | 4 | 6 | 24 | j-- |
| 5 | 5 | 5 | 5 | 25 | j-- |
| 5 | 6 | 5 | 4 | 20 | i++ |
| 4 | 6 | 4 | 3 | 12 | i++ |
| 7 | 6 | 6 | 2 | 12 | j-- |
| 7 | 3 | 3 | 1 | 3 | j-- |

A = [ 2, 1, 6, 1, 3, 4 ]  B = 8

max value of K such that all subarrays of length K are having sum <= B

| K | sa |
|---|---|
| 1 | 2, 1, 6, 1, 3, 4 |
| 2 | 2 1, 1 6, 6 1, 1 3, 3 4 |
| 3 | 2 1 6, 1 6 1, 6 1 3, 1 3 4 |

ans: lo = 0  hi = A.length

(1K)

A = [ 2, 1, 6, 1, 3, 4 ]  B = 8

ans = ~~0~~ ~~1~~ 2

check all subarrays with len = mid, if all of them are having sum <= B then return true otherwise return false

| | | | |
|---|---|---|---|
| lo = 0 | hi = 6 | mid = 3 | false, hi = mid - 1 |
| lo = 0 | hi = 2 | mid = 1 | true, lo = mid + 1 |
| lo = 2 | hi = 2 | mid = 2 | true, lo = mid + 1 |
| lo = 3 | hi = 2 | | |

Check all subarrays with len = mid, if all of them are having sum <= B then return true otherwise return false

PS array

```
isPossible (int [] A, int len, int B) {
    i = 0, j = len - 1;
    while ( j >= 0 && j < A.length) {
        // sum of sa from i to j
        int sum = PS[j];
        if( i > 0 ) {
            sum -= PS[i-1];
        }
        if (sum > B) {
            return false;
        }
        i++ ; j++ ;
    }
    return true;
}
```

k = 3  B = 8

A = [ 2, 1, 6, 1, 3, 4 ]
    i  j

A, B, C

div. by either B or C

A = 5 B = 3 C = 2

find Ath magical no.

=

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| X | 1 | 2 | 3 | X | 4 | X | 5 |

smaller problem

=

count of magical no. <= particular value (X)

B = 3 C = 5

X = 20

3 5 6 9 10 12 15 18 20

count of multiples of 3 <= X $\Rightarrow \frac{X}{3}$

count of multiples of 5 <= X $\Rightarrow \frac{X}{5}$

count of multiples of both 3 and 5 <= X $\Rightarrow \frac{X}{3*5}$

count of magical no. <= x $\Rightarrow$ $\frac{x}{B} + \frac{x}{C} - \frac{x}{LCM(B, C)}$