Two pointers

nums = [2, 7, 11, 15] target = 9

n = 4

{0, 1} {1, 2} {2, 3}

{0, 2} {1, 3}

{0, 3}

new int[]{i, j}

```
public int[] twoSum(int[] nums, int target) {
    int n=nums.length;

    for(int i=0; i<n; i++){
        for(int j=i+1; j<n; j++){
            if(nums[i]+nums[j]==target){
                return new int[]{i,j};
            }
        }
    }

    return new int[]{}; ⇒ empty j away
}
```

i = 0

j = 1

return

nums[0] + nums[1]

2 + 7 == 9

{0, 1}

-1 - 0 ⇒

(-1)

0 - (1)

0	1	2	3	4	5	6	7
-1,	0,	4,	5,	9,	12,	15,	18

(6)

target = 16

csum ⇒ 17 14 15 19 16

Csum = nums(i) + nums(j);

i++;

nums(i) - nums(j)

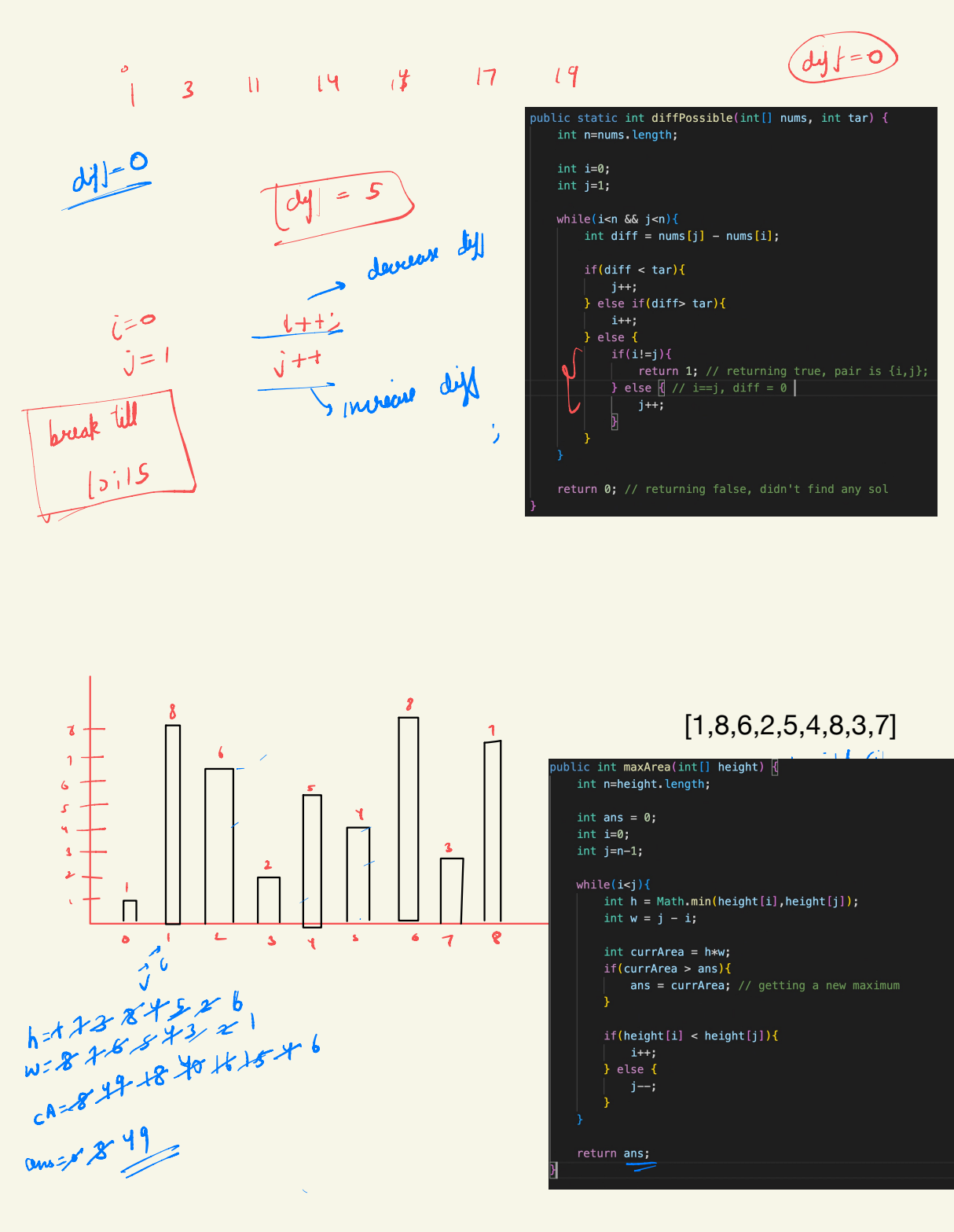
csum < target

csum > target

csum == tar {i, j}

j-1

```
public int[] twoSum(int[] nums, int target) {
    int n = nums.length;

    int i=0;
    int j=n-1;

    while(i<j){
        int csum = nums[i] + nums[j];

        if(csum < target){
            i++;
        } else if(csum > target){
            j--;
        } else {
            return new int[]{i+1,j+1};
        }
    }

    return new int[]{};
}
```


1 3 11 14 17 19
diff=0
diff=0
diff = 5
decrease diff
i=0
j=1
i++
j++
increase diff
break till
j=15
public static int diffPossible(int[] nums, int tar) {
int n=nums.length;
int i=0;
int j=1;
while(i<n && j<n){
int diff = nums[j] - nums[i];
if(diff < tar){
j++;
} else if(diff> tar){
i++;
} else {
if(i!=j){
return 1; // returning true, pair is {i,j};
} else { // i==j, diff = 0
j++;
}
}
}
return 0; // returning false, didn't find any sol
}
[1,8,6,2,5,4,8,3,7]
public int maxArea(int[] height) {
int n=height.length;
int ans = 0;
int i=0;
int j=n-1;
while(i<j){
int h = Math.min(height[i],height[j]);
int w = j - i;
int currArea = h*w;
if(currArea > ans){
ans = currArea; // getting a new maximum
}
if(height[i] < height[j]){
i++;
} else {
j--;
}
}
return ans;
}

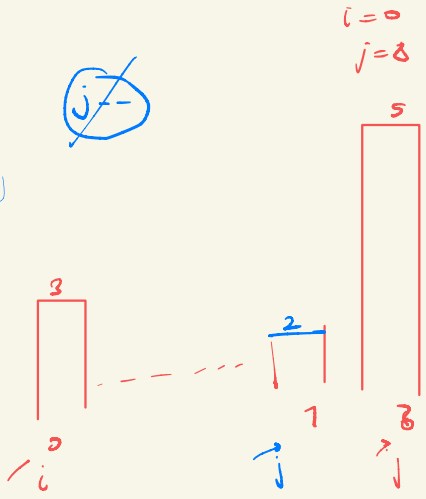
height (i) < height (j)

area = h(i) x w
→ h(i) x 8

area = h(i) x 1 { nhj > h(i)

{ nhj < hi

area = (nhj) x 1
2



Sort 0-1 array

→ O(n) O(1)

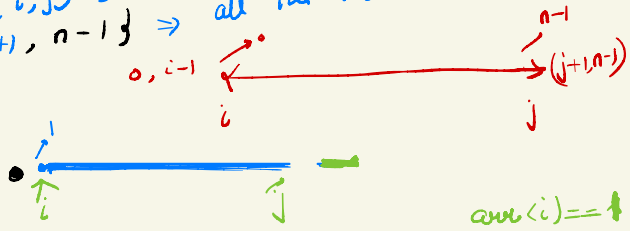
arr ⇒ { 1, 1, 0, 1, 0, 0, 1, 3 }
0 1 2 3 4 5 6
i
j

i = 0

j = 5

areas

{ 0, i-1 } ⇒ all the zeros
{ i, j } → undiscovered
{ j+1, n-1 } ⇒ all the ones



arr(i) == 1

j--;

```
void swap(int[] arr, int i, int j){
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
}

void segregate0and1(int[] arr, int n) {
    int i=0;
    int j=n-1;

    while(i<=j){
        if(arr[i]==1){
            swap(arr,i,j);
            j--;
        } else {
            i++;
        }
    }
}
```

Sort 0, 1 and 2

{0, p1} → zeros

{p1+1, p2-1} → ones

{p2, p3} → undiscovered

{p3+1, n-1} → twos

p1 = -1
p2 = 0
p3 = n-1

0 1 2 3 4 5 6 7
{0, 1, 1, 1, 0, 2, 2, 2}
n
p1
p2
p3

```java
public void sortColors(int[] nums) {
    int n=nums.length;

    int p1 = -1;
    int p2 = 0;
    int p3 = n-1;

    while(p2<=p3){ // until undiscovered area is not finish
        if(nums[p2]==2){
            swap(nums,p2,p3);
            p3--;
        } else if(nums[p2]==0){
            p1++;
            swap(nums,p1,p2);
            p2++;
        } else {
            p2++;
        }
    }
}
```

triplet
O(n²)

0 1 2 3 4 5 6
1, 2, 3, 4, 5, 6, 7
i

{1, 5}

2, 6 < 6

target = 8

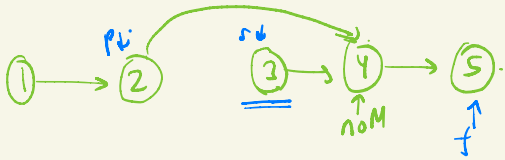
pairs
O(n)

s=0
{0, 5} < 8
{0, 4}
{0, 3}
{0, 2}
{0, 1}

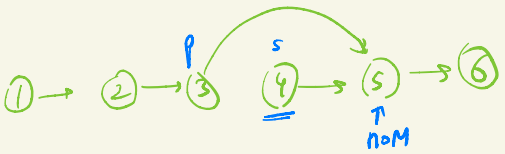
4-1
> 2
{1, 4}
(2, 3)
{1, 3}
{1, 2}

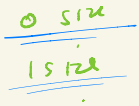
ans = 5 + 3 + 1 = 9

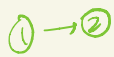
5 - 0 =

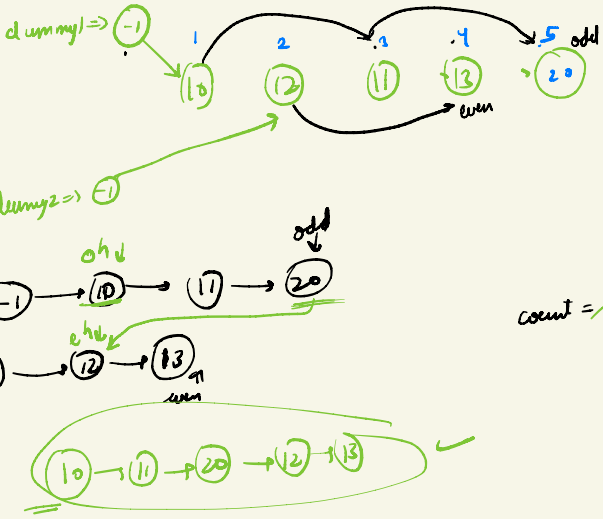


```
Node deleteMid(Node head){
    if(head==null){
        return null;
    }

    if(head.next==null){
        return new Node(-1);
    }

    Node prev = null;
    Node slow = head;
    Node fast = head;

    while(fast!=null && fast.next!=null){
        prev=slow;

        slow=slow.next;
        fast=fast.next.next;
    }
    Node nextOfMid = slow.next;
    prev.next = nextOfMid;
    return head;
}
```

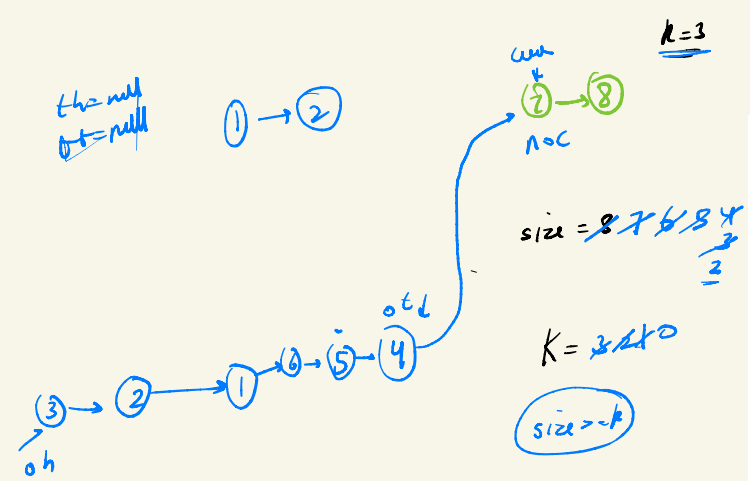








```
Node dummy1 = new Node(-1);
Node dummy2 = new Node(-1);

Node odd = dummy1;
Node even = dummy2;
Node curr = head;
int count = 1;

while(curr!=null){
    Node nextOfCurr = curr.next;
    curr.next=null;

    if(count%2==1){
        // connect odd
        odd.next = curr;
        curr = nextOfCurr;
        odd = odd.next;
    } else {
        even.next = curr;
        curr = nextOfCurr;
        even = even.next;
    }

    count++;
}

Node oddHead = dummy1.next;
Node evenHead = dummy2.next;

// connect odd to even
odd.next = evenHead;

return oddHead;
```




```
int size = getSize(head);
ListNode curr = head;
// base case ()

while(){
    // reverse in pair of k
    int K=k;
    while(K-- > 0){
        ListNode nextOfCurr = curr.next;
        curr.next = null;

        addFirst(curr);
        curr = nextOfCurr;

        size--;
    }

    // add th,tt to my original
    if(oh==null){
        oh = th;
        ot = tt;
    } else {
        ot.next = th;
        ot = tt;
    }
}

ot.next = curr;

return oh;
```

th=null;
tt=null;

```
void addFirst(ListNode nn){
    if(th == null){
        th = nn;
        tt = nn;
    } else {
        nn.next = th;
        th = nn;
    }
}
```