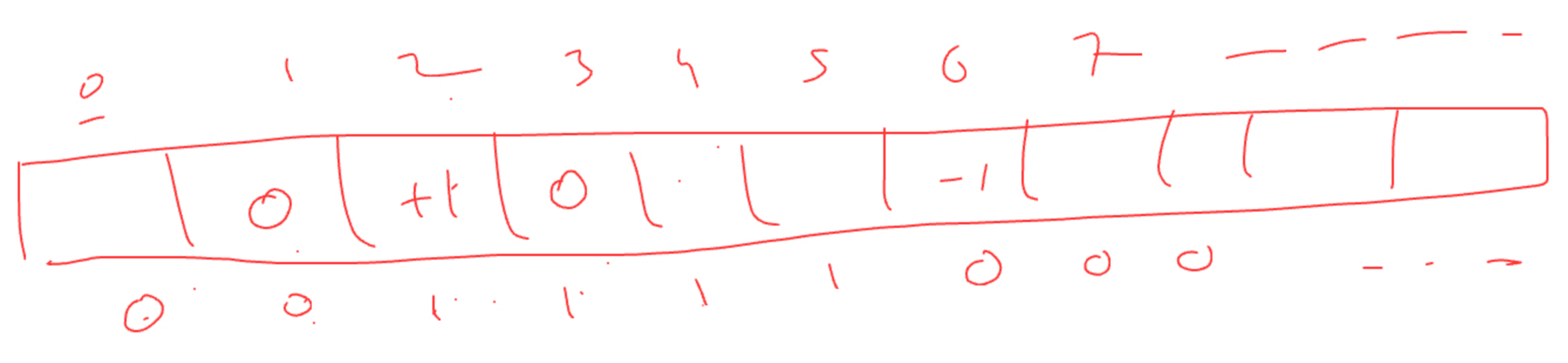
[2, 5].



L[i] → +1

R[i]+1 → -1

maxx.

maxx + 2,

0 1 2 3 4

---

arr: [1, 2, 3, 4]

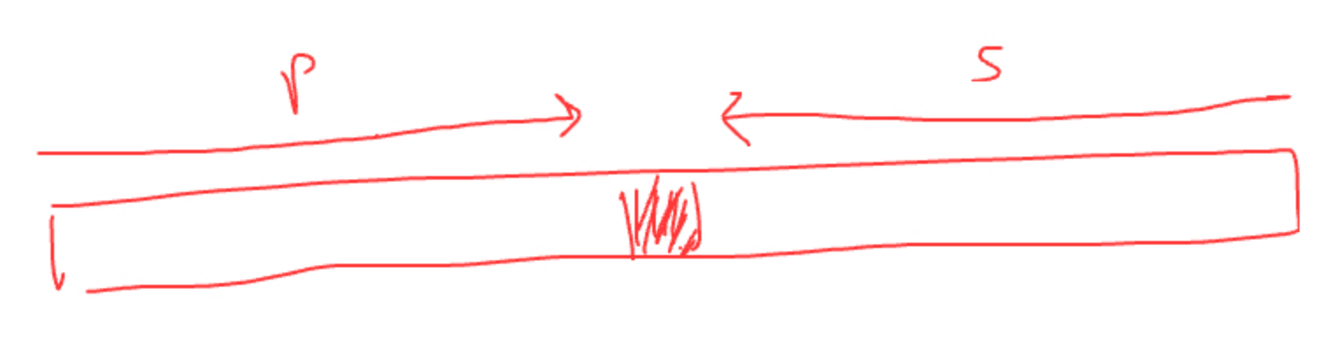
ans: [24, 12, 8, 6]

arr: [-1, 1, 0, -3, 3]

ans: [0, 0, 9, 0, 0].

arr: [-1, 0, 0, -3, 3]

ans: [0, 0, 0, 0, 0].



p × s → 32-bit int

p×s× arr[i] → can be greater than 32-bit.

→ PP[i] = arr[0] × arr[1] - - - arr[i-1].

→ SP[i] = arr[i+1] × arr[i+2] - - - arr[n-1].

ans[i] = PP[i] × SP[i].

~~PP[n-1]~~.

arr: [1, 2, 3, 4]

- PP: [1, 1, 2, 6]

- SP: [24, 12, 4, 1]

ans: [24, 12, 8, 6].

TC: O(n)

AS: O(n).

```
suffProd[n - 1] = 1;
for (int i = n - 2; i >= 0; i--) {
    suffProd[i] = suffProd[i + 1] * arr[i + 1];
}

vector<int> ans(n);
ans[0] = suffProd[0];
int prefProd = 1;
for (int i = 1; i < n; i++) {
    prefProd = prefProd * arr[i - 1];
    ans[i] = prefProd * suffProd[i];
}
```

arr: [1, 2, 3, 4].

suffProd: [24, 12, 4, 1]

ans: [24, 12, 8, 6].

PrefProd = ~~1~~ ~~2~~ 6

---

i: 0 → n-1

j: i+1 → n-1

arr[i] + arr[j] == tar.

Find target - arr[i]

↓

Using Binary Search → O(log(n)) ⇒ O(n log(n)).

&

[1, 2, 5, 6, 10] tar = 8

~~6+10 = 16~~

~~1+2 = 3~~

1 + 10 = 11 ⇒ 11 > 8 → r--

1 + 6 = 7 ⇒ 7 < 8 → l++

2 + 6 = 8 ✓.

[1, 2, 5, 6, 10] tar = 9

1 + 10 = 11 ⇒ 11 > 9 → r--

1 + 6 = 7 l++

2 + 6 = 8 l++

5 + 6 = 11

TC: O(n)

AS: O(1).

[1, 2, 5, 5, 6, 10] tar = 10

O/P → True.

TC: O(n²)

AS: O(1).

---

i: 0 → n-1

j: i+1 → n-1

k: j+1 → n-1

sum = $a_i + a_j + a_k$

TC = O(n³)

[1, 4, 6, 8, 10, 45] tar = 13.

tar - arr[i].

TC: O(n log n + n²) = O(n²)

Quad Sum Problem.

BF: O(n⁴)

2-ptr: O(n³).

⇒ K-sum Problem

BF: O(n^k)

2-ptr: O(n^(k-1))

func (arr, tar, k, idx)

↓

func (arr, tar - arr[i], k-1, i+1)

↓

func (arr, tar - arr[i], k-2, i+1)

↓

(k = 2.) Base - case

[1, 2, 2, 3, 3] → [1, 2, 3].

[1, 1, 2] → [1, 2].

[1, 2, 2, 3, 3] → [1, 2, 3]

[1, 2, 2, 3, 3]

↓

[1, 2, 3 - -] return 3.

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

↓

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