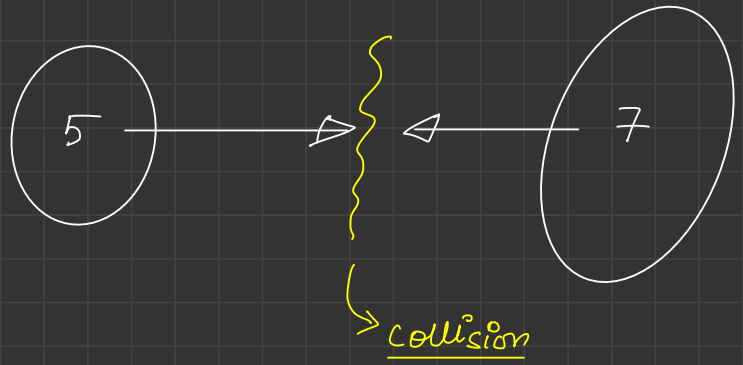
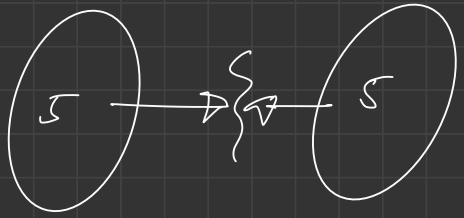
Asteroid Collision



Rule 1: When 2 asteroids collide, then, the smaller asteroid gets destroyed & the bigger moves unaffected.

Rule 2: Both same size collide, both will get destroyed.

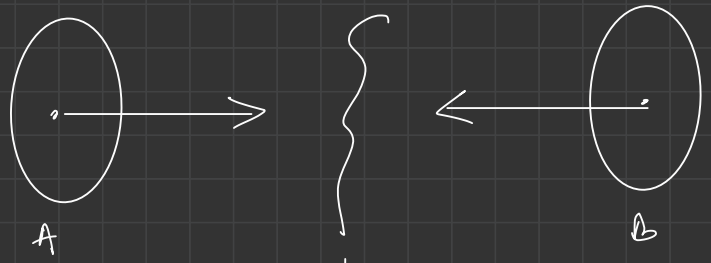


Case1

Case2

Case 3.

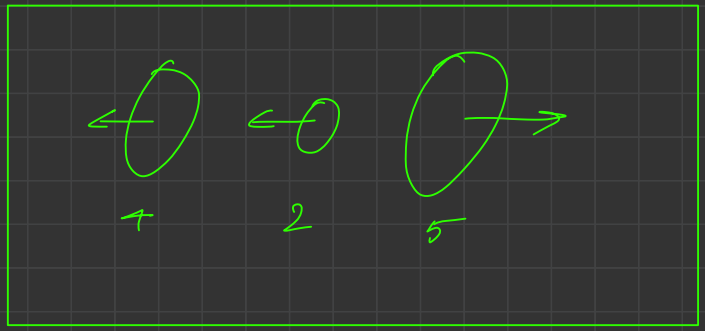
Case 4



Collision

asteroids[] = { 1, 2, 3, −4, −2, 5, −3 }

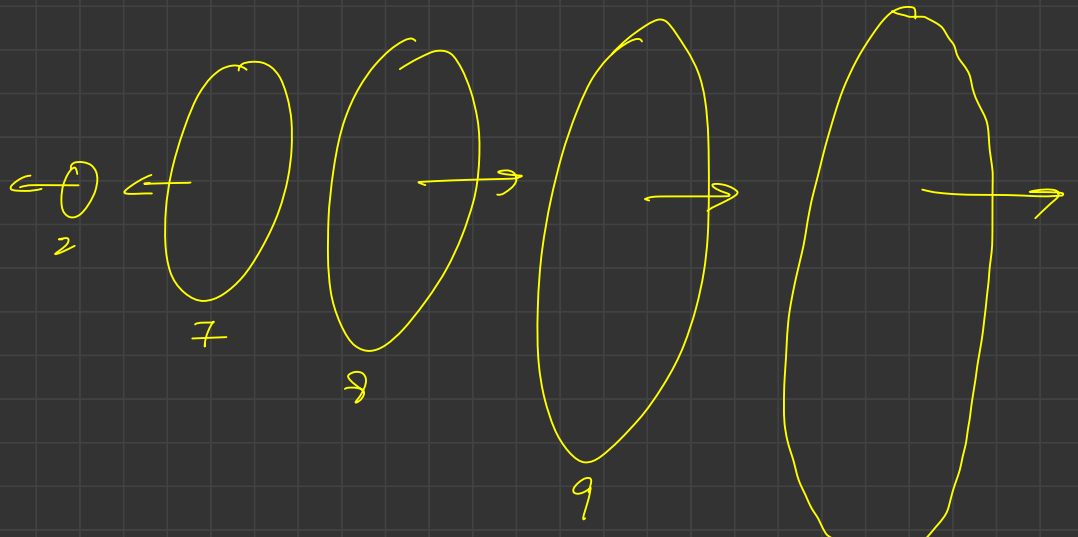
(+)ve value means moving towards right }
(−)ve value means moving towards left }



{ −4, −2, 5 }

↳ ans!

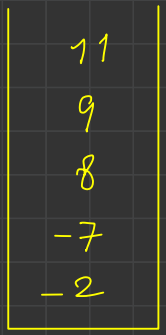
asteroid[] = { -2, 3, 4, -4, 5, -7, 8, 9, -3, -2, 11 }



{ -2, -7, 8, 9, 11 }

Ans

asteroid[] = {-2, 3, 4, -4, 5, -7, 8, 9, -3, -2, 11}

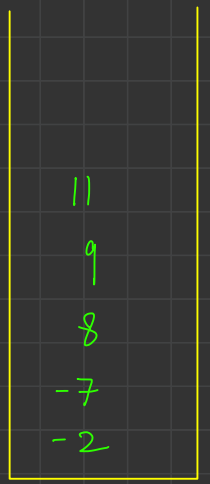


Stack

↳ people of stable universe

asteroid[] = {-2, 3, 4, -4, 5, -7, 8, 9, -3, -2, 11}

```java
// stack : people moving freely in my stable universe
Stack<Integer> st = new Stack<>();

for (int asteroid : asteroids) {
    if (asteroid > 0) {
        // no threat to stable universe
        st.push(asteroid);
    } else {
        // try to destroy as much asteroid of stable universe moving towards right
        while (st.size() > 0 && st.peek() > 0 && st.peek() < -1 * asteroid) {
            // destroy peek person
            st.pop();
        }

        if (st.size() > 0 && st.peek() > 0 && st.peek() > -1 * asteroid) {
            // don't do anything as incoming asteroid got destroyed
        } else if (st.size() > 0 && st.peek() > 0 && st.peek() == -1 * asteroid) {
            // both peek value and incoming gets destroyed
            st.pop();
        } else {
            st.push(asteroid);
        }
    }
}
```



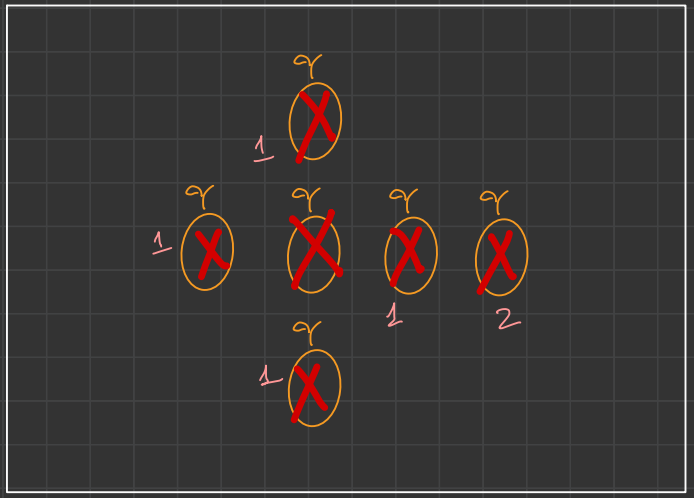
{-2, -7, 8, 9, 11}

Rotten Oranges

↳ BFS

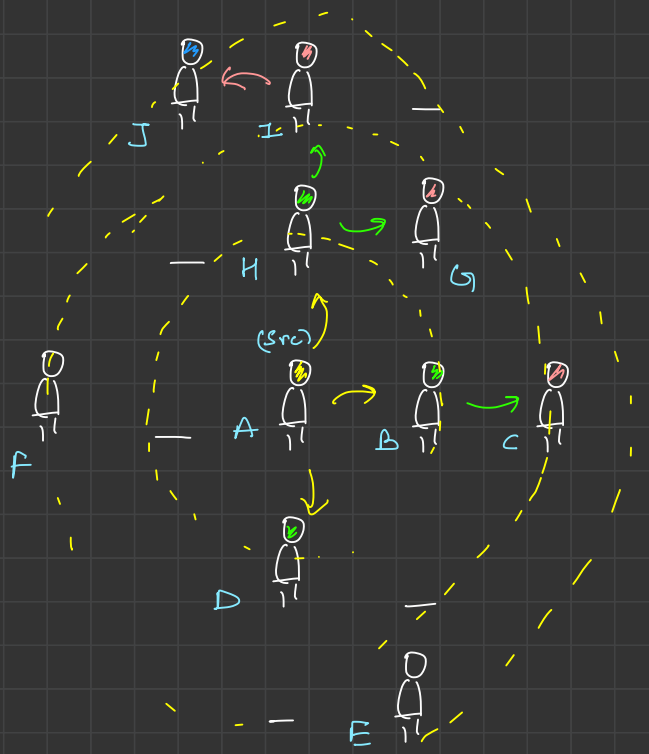
(Breadth First Search)

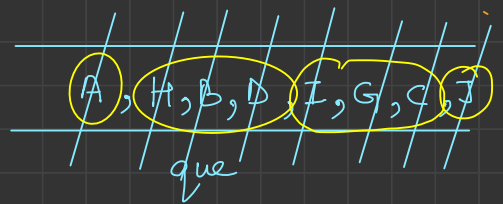
min time = 2



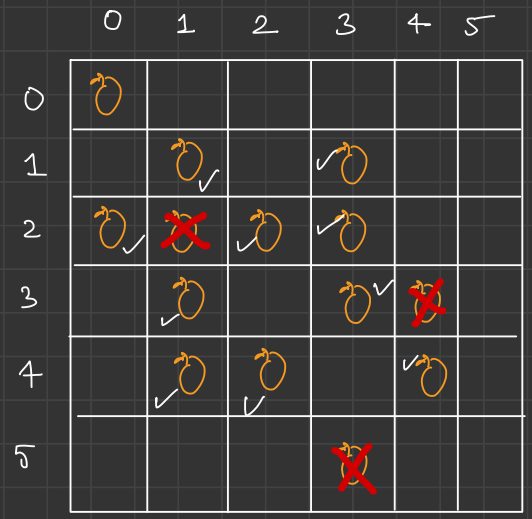
3 units fine

Breadth first Search





dist = 0 X X 3



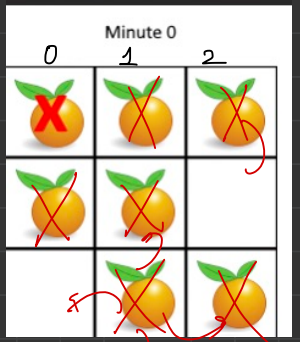
time = level - 1

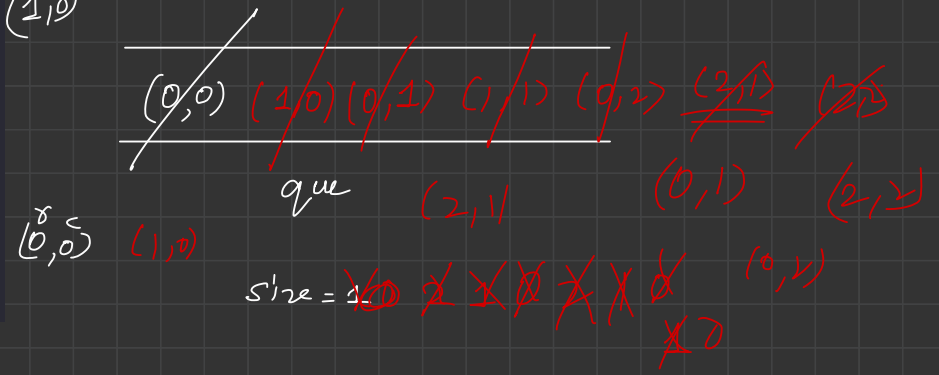
que

level = ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ 4

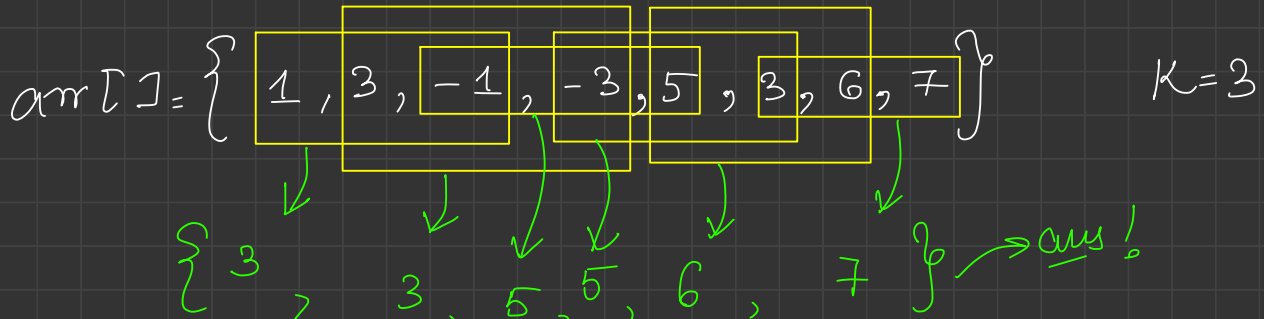
size = ~~2~~ ~~1~~ 0

```java
int level = 0;

while (que.size() > 0) {
    int size = que.size();

    while (size-->0) {
        Pair rpair = que.remove();
        int r = rpair.row;
        int c = rpair.col;

        if (r + 1 < n && grid[r + 1][c] == 1) {
            grid[r + 1][c] = 2;
            que.add(new Pair(r + 1, c));
        }
        if (r - 1 >= 0 && grid[r - 1][c] == 1) {
            grid[r - 1][c] = 2;
            que.add(new Pair(r - 1, c));
        }
        if (c + 1 < m && grid[r][c + 1] == 1) {
            grid[r][c + 1] = 2;
            que.add(new Pair(r, c + 1));
        }
        if (c - 1 >= 0 && grid[r][c - 1] == 1) {
            grid[r][c - 1] = 2;
            que.add(new Pair(r, c - 1));
        }
    }

    level++;
}
```





Sliding Window Maximum

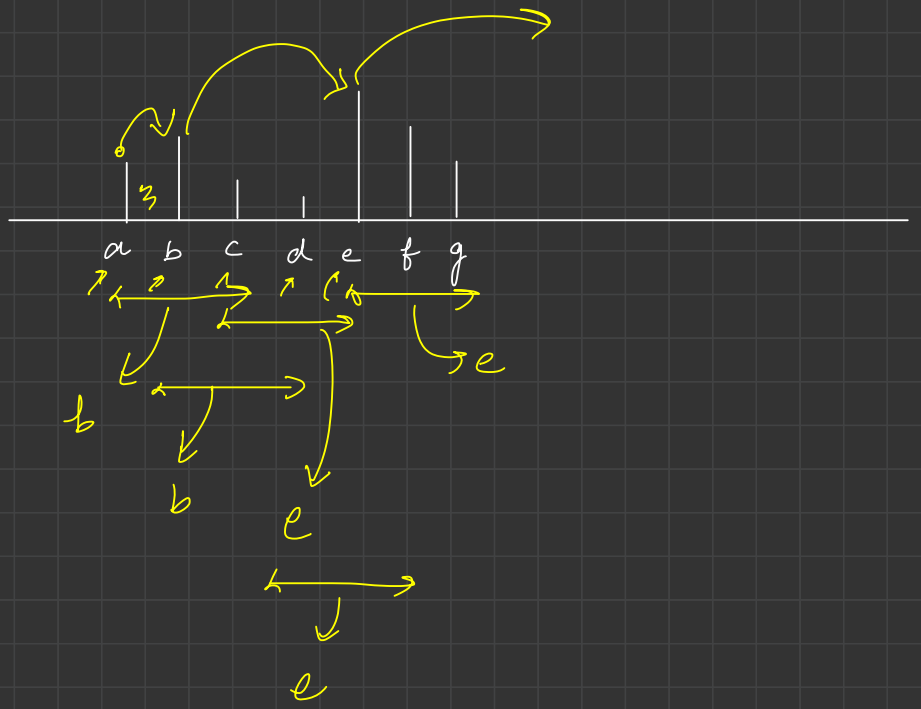


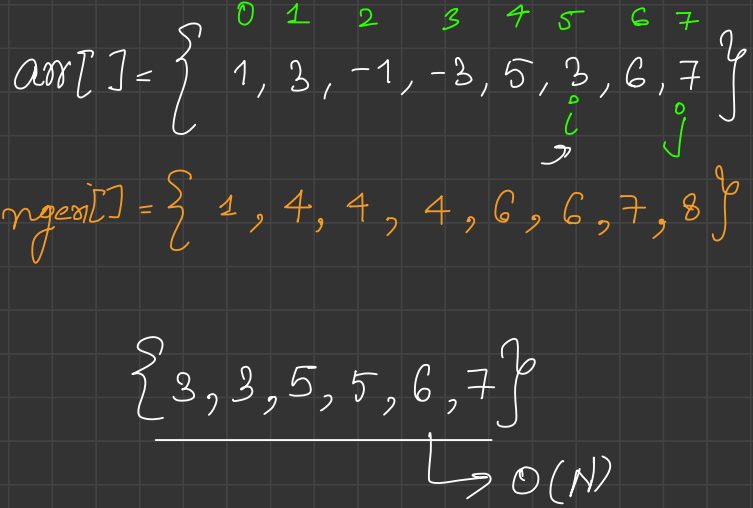
	0	1	2	3	4	5	6	7
arr[] = {	1	3	-1	-3	5	3	6	7 }

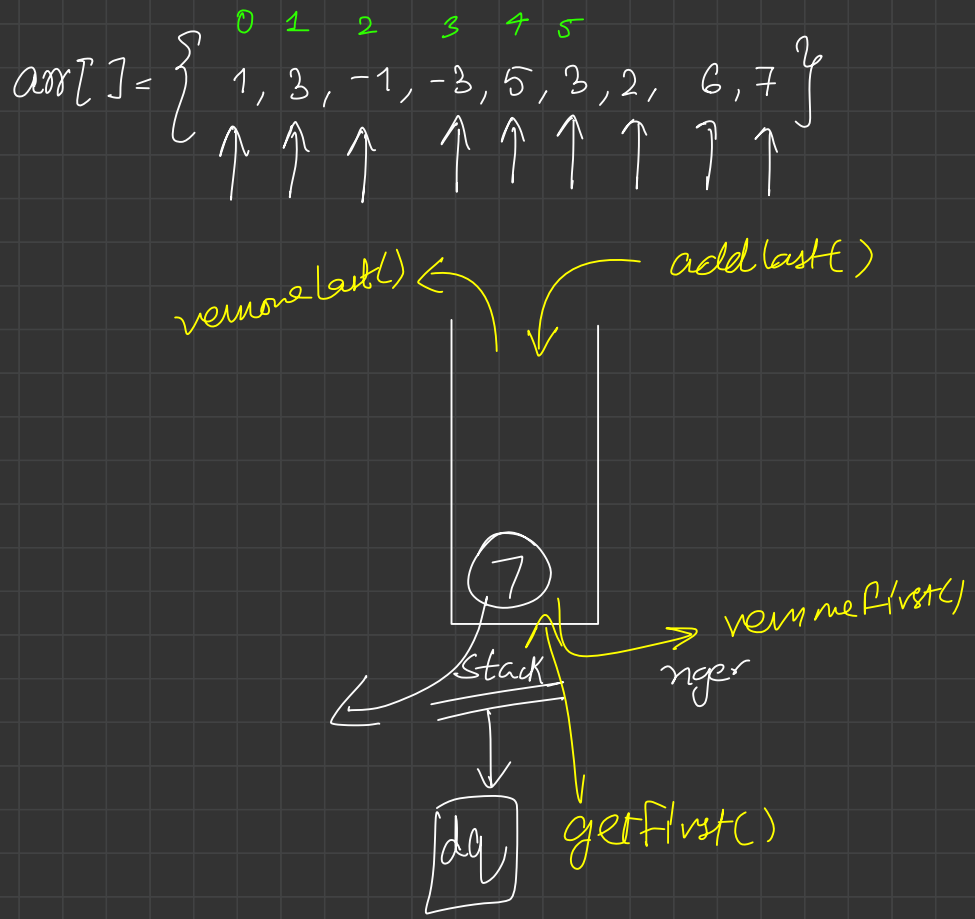
```
{ for(i = 0 ⟶ N-K+1)
    for(int j=i; j<K; j++)
```

Brute force

↳ TC: O(N×K)
SC: O(1)

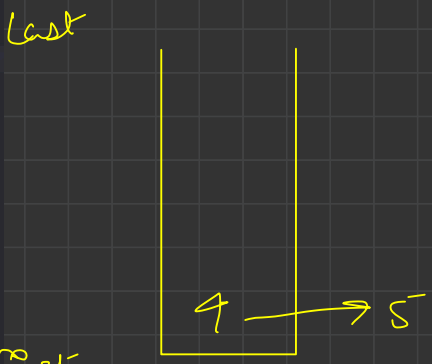

a
b
c
d
e
f
g
b
b
e
e
e


0 1 2 3 4 5 6 7
arr[] = { 1, 3, -1, -3, 5, 3, 6, 7 }
i
j
nge[] = { 1, 4, 4, 4, 6, 6, 7, 8 }
{3, 3, 5, 5, 6, 7}
O(N)


arr[] = { 1, 3, -1, -3, 5, 3, 2, 6, 7 }
0 1 2 3 4 5
removeLast()
addLast()
7
Stack
removeFirst()
Queue
dq
getFirst()

```java
// TC: O(N), SC: O(N)
static int[] SlidingWindowMaximum(int n, int k, int[] arr){
    // write code here
    Deque<Integer> st = new ArrayDeque<>();

    int[] ans = new int[n - k + 1];
    int win_num = 0;
    for (int i = 0; i < n; i++) {
        if (st.size() > 0 && st.getFirst() == i - k) {
            st.removeFirst();
        }

        int ele = arr[i];
        while (st.size() > 0 && arr[st.getLast()] < ele) {
            st.removeLast();
        }

        st.addLast(i);

        if (i >= k - 1) {
            ans[win_num] = arr[st.getFirst()];
            win_num++;
        }
    }

    return ans;
}
```

arr[] = { 1, 3, -1, -3, 5, 3, 2, 6, 7 }
0 1 2 3 4 5 6 7 8



3, 3, 5