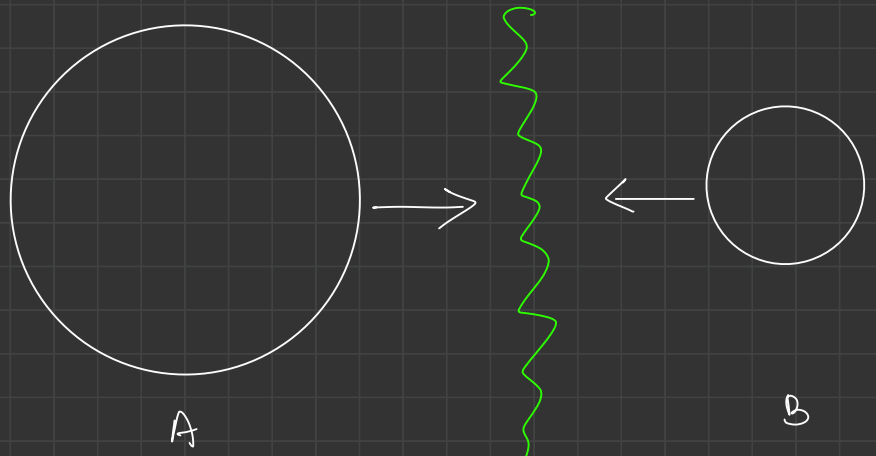
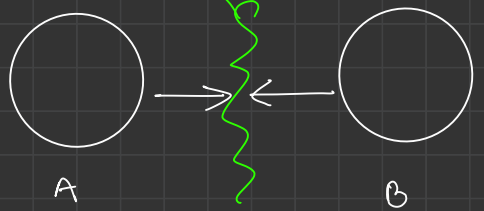
Asteroid Collision



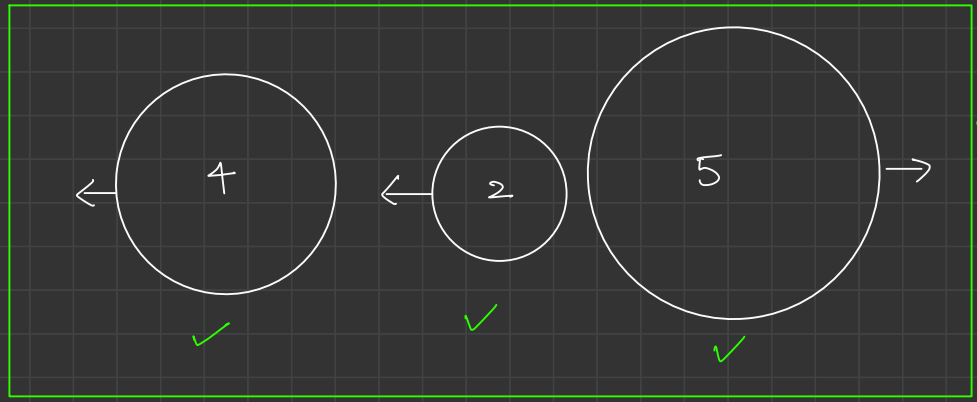
collision

{ smaller will get destroyed }

{ same size when collide both will get destroyed }



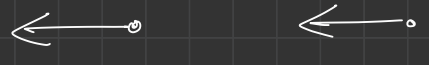
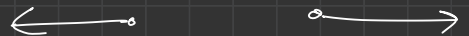
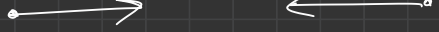
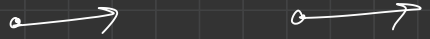
asteroids[] = { 1, 2, 3, -4, -2, 5, -3 }



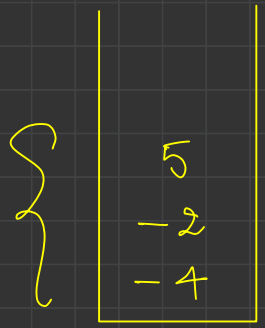
→ asteroids moving in universe freely.

ans: { -4, -2, 5 }

Condition of collision

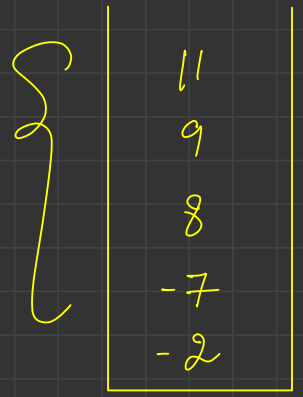
1.  ✗ Never Collide
2.  ✗ Never Collide
3.  ✓ Collision
4.  ✗ Never Collide

asteroids [ ] : {1, 2, 3, -4, -2, 5, -5}

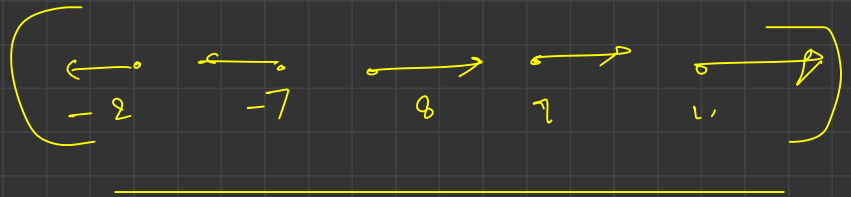


freely moving asteroids ✓

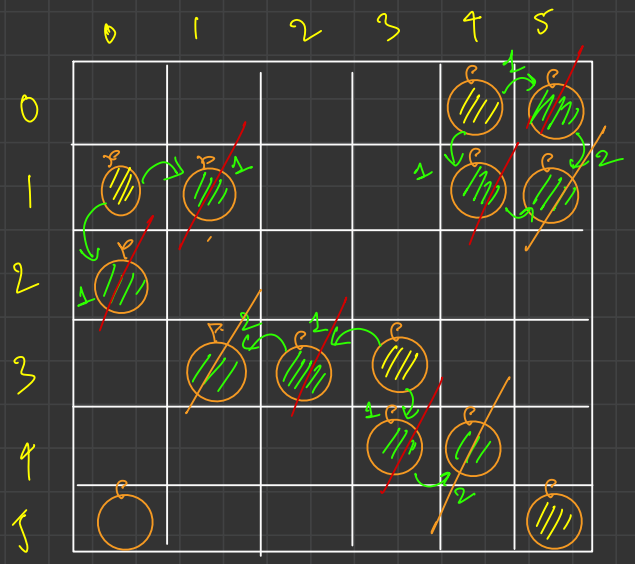
asteroids[] = {-2, 3, 4, -4, 5, -7, 8, 9, -3, -2, 11}



Stack



Rotten Oranges



{ 2 units of time }

95%

Queue

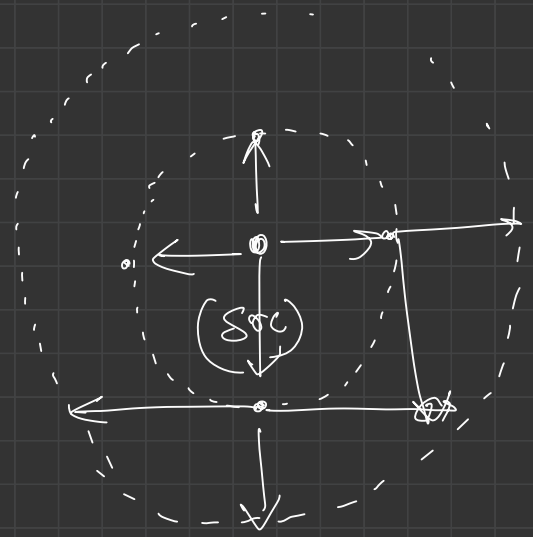
array, trees, graph

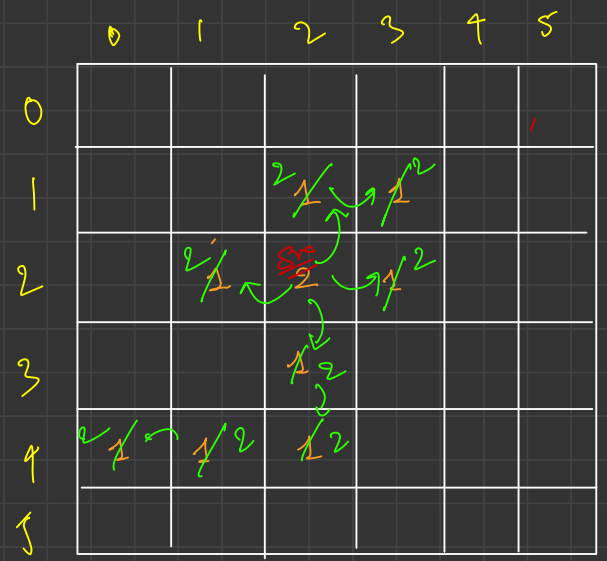
BFS

Breadth first Search

Min time | distance

BFS





(4/6)

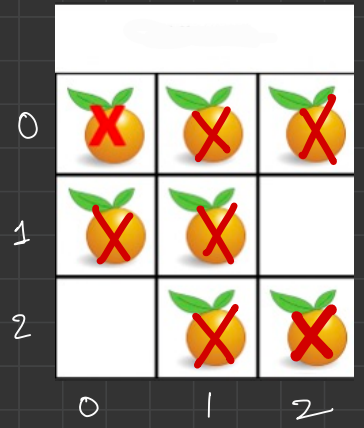
que

time = 0/1/2/3/4/5

(time-1)

```java
int level = 0;

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

    while (size-->0) {
        Pair rpair = que.remove();

        // can I rotten someone on right
        if (rpair.col + 1 < m && grid[rpair.row][rpair.col + 1] == 1) {
            grid[rpair.row][rpair.col + 1] = 2;
            que.add(new Pair(rpair.row, rpair.col + 1));
        }

        // can I rotten someone on bottom
        if (rpair.row + 1 < n && grid[rpair.row + 1][rpair.col] == 1) {
            grid[rpair.row + 1][rpair.col] = 2;
            que.add(new Pair(rpair.row + 1, rpair.col));
        }

        // can I rotten someone on left
        if (rpair.col - 1 >= 0 && grid[rpair.row][rpair.col - 1] == 1) {
            grid[rpair.row][rpair.col - 1] = 2;
            que.add(new Pair(rpair.row, rpair.col - 1));
        }

        // can I rotten someone on top
        if (rpair.row - 1 >= 0 && grid[rpair.row - 1][rpair.col] == 1) {
            grid[rpair.row - 1][rpair.col] = 2;
            que.add(new Pair(rpair.row - 1, rpair.col));
        }
    }

    level++;
}
```

Change Coding Language



(0,2) (1,1)

(queue) (1,1)

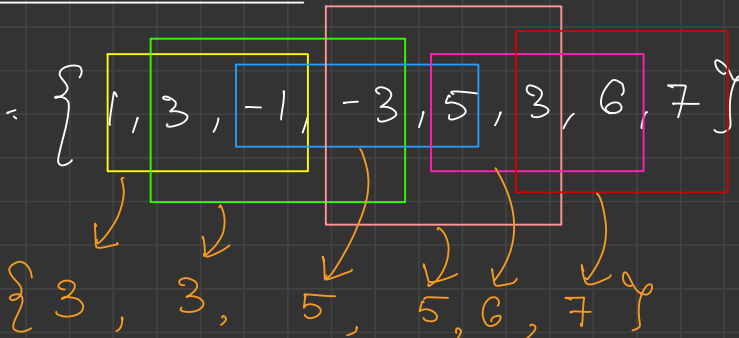
level = ~~0~~ ~~1~~ 2

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

time = level - 1 = 3 - 1 = 2 ~~1~~ ~~0~~ 3

Sliding window Maximum

arr[] = { 1, 3, -1, -3, 5, 3, 6, 7 } K = 3



Brute force

```
for(int i=0; i<n; i++)
{ for(int j=i; j<i+k; j++)
  {
    max
  }
}
```

TC : O(N * K)
SC : O(1)

nger = {1, 4, 4, 4, 6, 6, 7, 8}

j

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

K = 3

i

{3, 3, 5, 5, 6, 7}

```java
static int[] SlidingWindowMaximum(int N, int K, int[] arr){
    // write code here
    int[] ngeri = nextGreaterElementOnRightIndexwise(arr, N);

    int[] ans = new int[N - K + 1];

    int j = 0;
    for (int i = 0; i <= N - K; i++) {
        if (j < i) {
            j = i;
        }

        while (ngeri[j] < i + k) {
            j = ngeri[j];
        }

        ans[i] = arr[j];
    }

    return ans;
}
```

K=3 N=8

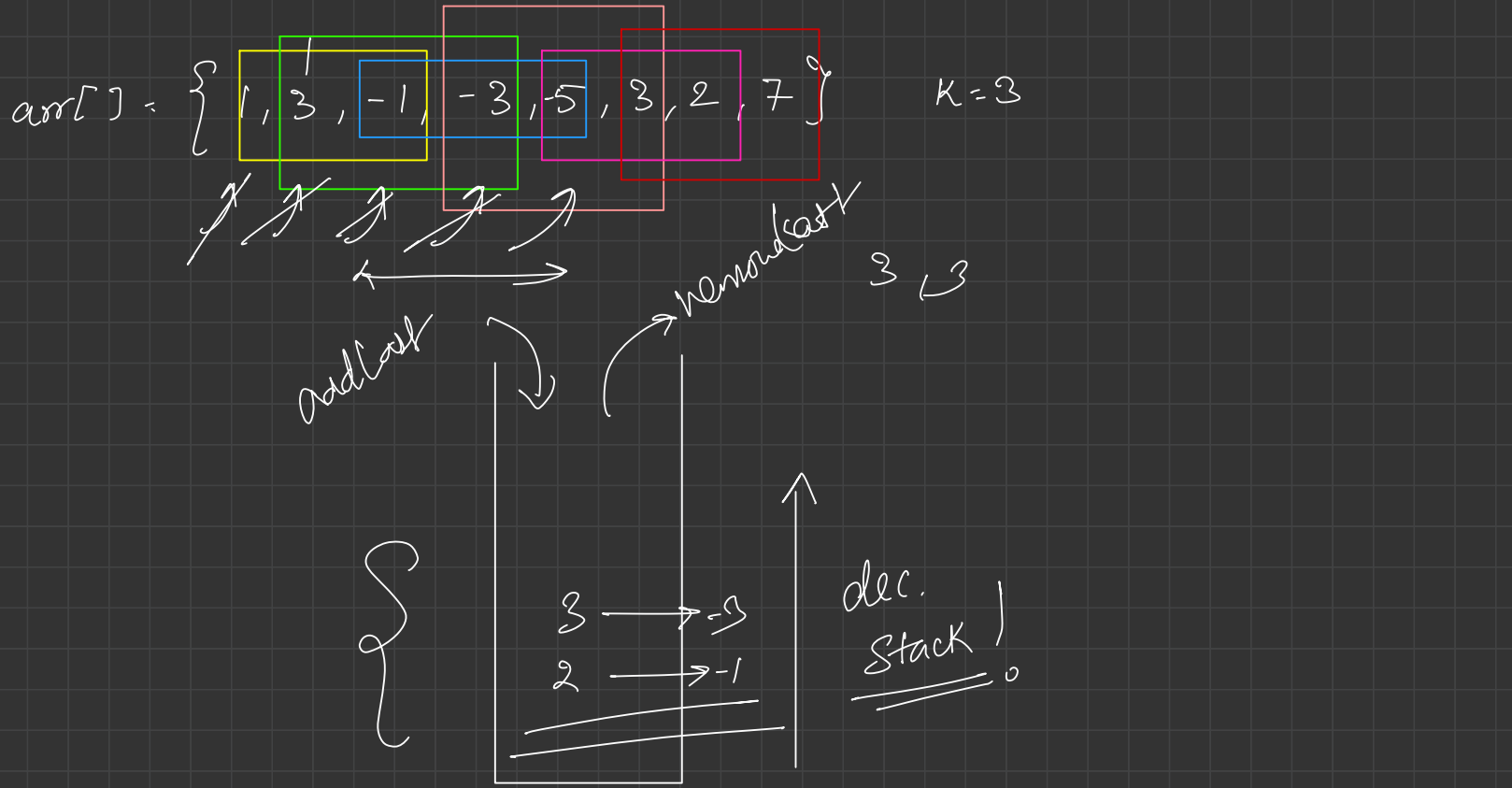
ngeri = {1, 4, 4, 4, 6, 6, 7, 8}

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

ans = {3, 3, 5, 5, 6, 7}

N-K+1 → Windows

TC : O(N)
SC : O(N)


arr[] = { 1, 3, -1, -3, -5, 3, 2, 7 }
K = 3
addLast
removeLast
3
3
3 → -3
2 → -1
dec.
Stack !
0