

Day 2  
Piyush Sharma  
22bcs15351

## Array & Linked List

Very Easy

### Question 1: Majority Elements

Given an array `nums` of size `n`, return the majority element

The majority element is the element that appears more than  $\lfloor n / 2 \rfloor$  times You may assume that the majority element always exists in the array

Example 1:

Input: `nums = [3,2,3]`

Output: `3`

Example 2:

Input: `nums = [2,2,1,1,1,2,2]`

Output: `2`

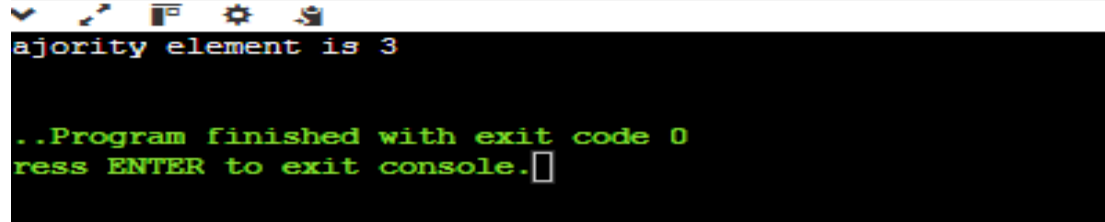
Constraints:

- `n == nums.length`
- $1 \leq n \leq 5 * 10^4$
- $-10^9 \leq \text{nums}[i] \leq 10^9$

Follow-up: Could you solve the problem in linear time and in  $O(1)$  space?

## Output

```
1  #include <stdio.h>
2
3  int main() {
4      int arr[] = {3,2,3};
5      int size = sizeof(arr) / sizeof(arr[0]);
6      int count = 0, candidate = -1;
7
8      for (int i = 0; i < size; i++) {
9          if (count == 0) candidate = arr[i];
10         count += (arr[i] == candidate) ? 1 : -1;
11     }
12
13     count = 0;
14     for (int i = 0; i < size; i++)
15         if (arr[i] == candidate) count++;
16
17     if (count > size / 2)
18         printf("Majority element is %d\n", candidate);
19     else
20         printf("No majority element\n");
21
22     return 0;
23 }
24
```



The screenshot shows the output of the C program. The first line of output is "Majority element is 3". Below this, there is a green text message: "..Program finished with exit code 0". The last line of the screenshot shows "Press ENTER to exit console." with a cursor.

Easy

## Question 2: Pascal's Triangle

Given an integer `numRows`, return the first `numRows` of Pascal's triangle

In Pascal's triangle, each number is the sum of the two numbers directly above it as shown:

Example 1:

Input: `numRows = 5`

Output: `[[1],[1,1],[1,2,1],[1,3,3,1],[1,4,6,4,1]]`

Example 2:

Input: `numRows = 1`

Output: `[[1]]`

Constraints:

- `1 <= numRows <= 30`

Output:

```
1  #include <stdio.h>
2
3  void generatePascalsTriangle(int numRows) {
4      int triangle[numRows][numRows];
5      for (int i = 0; i < numRows; i++) {
6          triangle[i][0] = 1;
7          triangle[i][i] = 1;
8          for (int j = 1; j < i; j++) {
9              triangle[i][j] = triangle[i - 1][j - 1] + triangle[i - 1][j];
10         }
11     }
12     for (int i = 0; i < numRows; i++) {
13         for (int j = 0; j <= i; j++) {
14             printf("%d ", triangle[i][j]);
15         }
16         printf("\n");
17     }
18 }
19 int main() {
20     int numRows;
21     printf("Enter the number of rows: ");
22     scanf("%d", &numRows);
23     generatePascalsTriangle(numRows);
24
25     return 0;
26 }
27
```

Enter the number of rows: 5

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

Medium

### Question 3: Container With Most Water

You are given an integer array `height` of length `n`. There are `n` vertical lines drawn such that the two endpoints of the `i`<sup>th</sup> line are `(i, 0)` and `(i, height[i])`.

Find two lines that together with the x-axis form a container, such that the container contains the most water.

Return the maximum amount of water a container can store.

Notice that you may not slant the container.

Example 1:

Input: `height = [1,8,6,2,5,4,8,3,7]`

Output: `49`

Explanation: The above vertical lines are represented by array `[1,8,6,2,5,4,8,3,7]`. In this case, the max area of water (blue section) the container can contain is `49`.

Example 2:

Input: `height = [1,1]`

Output: `1`

Constraints:

- `n == height.length`
- `2 <= n <= 105`
- `0 <= height[i] <= 104`

Output:

```
main.cpp
1 #include <stdio.h>
2 int maxArea(int height[], int n) {
3     int left = 0, right = n - 1;
4     int max_area = 0;
5
6     while (left < right) {
7         int width = right - left;
8         int min_height = (height[left] < height[right]) ? height[left] : height[right];
9         int area = min_height * width;
10        if (area > max_area) {
11            max_area = area;
12        }
13        if (height[left] < height[right]) {
14            left++;
15        } else {
16            right--;
17        }
18    }
19    return max_area;
20 }
21 int main() {
22     int height[] = {1, 8, 6, 2, 5, 4, 8, 3, 7};
23     int n = sizeof(height) / sizeof(height[0]);
24
25     int result = maxArea(height, n);
26     printf("Maximum area: %d\n", result);
27     return 0;
28 }
```

Maximum area: 49

Hard

#### Question 4: Maximum Number of Groups Getting Fresh Donuts

There is a donuts shop that bakes donuts in batches of `batchSize`. They have a rule where they must serve all of the donuts of a batch before serving any donuts of the next batch. You are given an integer `batchSize` and an integer array `groups`, where `groups[i]` denotes that there is a group of `groups[i]` customers that will visit the shop. Each customer will get exactly one donut.

When a group visits the shop, all customers of the group must be served before serving any of the following groups. A group will be happy if they all get fresh donuts. That is, the first customer of the group does not receive a donut that was left over from the previous group.

You can freely rearrange the ordering of the groups. Return the maximum possible number of happy groups after rearranging the groups.

Example 1:

Input: `batchSize = 3, groups = [1,2,3,4,5,6]`

Output: `4`

Explanation: You can arrange the groups as `[6,2,4,5,1,3]` Then the 1st, 2nd, 4th, and 6th groups will be happy

Example 2:

Input: `batchSize = 4, groups = [1,3,2,5,2,2,1,6]`

Output: `4`

Constraints:

- `1 <= batchSize <= 9`
- `1 <= groupslength <= 30`
- `1 <= groups[i] <= 10^9`

Output:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int compare(const void *a, const void *b) {
4      return *(int*)b - *(int*)a;
5  }
6  int maxHappyGroups(int batchSize, int groups[], int n) {
7      qsort(groups, n, sizeof(int), compare);
8      int happyCount = 0;
9      int remainingDonuts = 0;
10     for (int i = 0; i < n; i++) {
11         int groupSize = groups[i];
12         if (remainingDonuts >= groupSize) {
13             remainingDonuts -= groupSize;
14             happyCount++;
15         } else {
16             remainingDonuts += batchSize - (groupSize % batchSize);
17         }
18     }
19     return happyCount;
20 }
21 int main() {
22     int batchSize = 3;
23     int groups[] = {1, 2, 3, 4, 5, 6};
24     int n = sizeof(groups) / sizeof(groups[0]);
25     int result = maxHappyGroups(batchSize, groups, n);
26     printf("Maximum number of happy groups: %d\n", result);
27     return 0;
28 }
```

Maximum number of happy groups: 3

Very Hard

### Question 5: Find Minimum Time to Finish All Jobs

You are given an integer array `jobs`, where `jobs[i]` is the amount of time it takes to complete the `i`th job. There are `k` workers that you can assign jobs to. Each job should be assigned to exactly one worker. The working time of a worker is the sum of the time it takes to complete all jobs assigned to them. Your goal is to devise an optimal assignment such that the maximum working time of any worker is minimized.

Return the minimum possible maximum working time of any assignment.

Example 1:

Input: `jobs = [3,2,3]`, `k = 3`

Output: `3`

Explanation: By assigning each person one job, the maximum time is `3`.

Example 2:

Input: `jobs = [1,2,4,7,8]`, `k = 2`

Output: `11`

Explanation: Assign the jobs the following way:

Worker 1: `1, 2, 8` (working time =  $1 + 2 + 8 = 11$ )

Worker 2: `4, 7` (working time =  $4 + 7 = 11$ )

The maximum working time is `11`

Constraints:

- `1 <= k <= jobslength <= 12`

- `1 <= jobs[i] <= 10^7`

Output:

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
int canAssignJobs(int jobs[], int n, int k, int maxTime) {
```

```
    int workers = 1, currentTime = 0;
```

```
    for (int i = 0; i < n; i++) {
```

```
        if (currentTime + jobs[i] > maxTime) {
```

```
            workers++;
```

```
            currentTime = jobs[i];
```

```
            if (workers > k) return 0;
```

```
        } else {
```

```
            currentTime += jobs[i];
```

```
    }
```

```
    return 1;}
```

```
int minTimeToFinishJobs(int jobs[], int n, int k) {
```

```
    int left = 0, right = 0;
```

```
    for (int i = 0; i < n; i++) {
```

```
        left = (left > jobs[i]) ? left : jobs[i];
```

```
        right += jobs[i]; }
```

```
    int result = right;
```

```
    while (left <= right) {
```

```
        int mid = left + (right - left) / 2;
```

```
        if (canAssignJobs(jobs, n, k, mid)) {
```

```
            result = mid;
```

```
            right = mid - 1;
```

```
        } else {
```

```
            left = mid + 1;}
```

```
    }
```

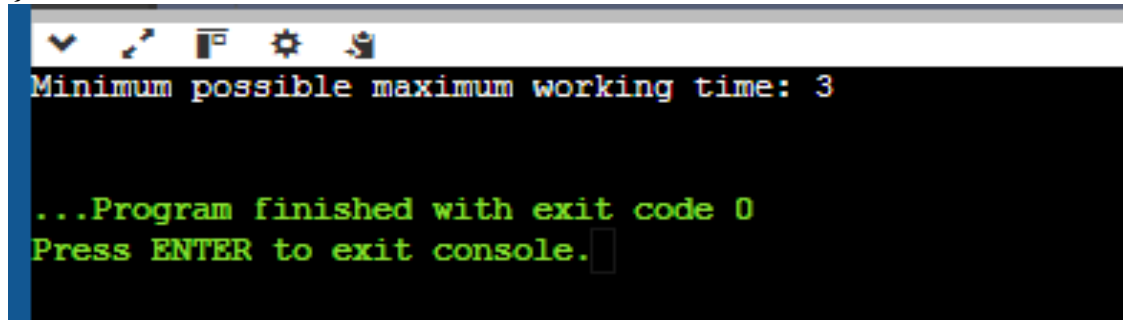
```
    return result;}
```

```
int main() {
```

```
    int jobs[] = {3, 2, 3};
```



```
int k = 3;  
int n = sizeof(jobs) / sizeof(jobs[0]);  
int result = minTimeToFinishJobs(jobs, n, k);  
printf("Minimum possible maximum working time: %d\n", result);  
return 0;  
}
```

A screenshot of a console window with a dark background and a light blue title bar. The title bar contains several icons: a checkmark, a magnifying glass, a square, a gear, and a person. The console output is as follows:  
Minimum possible maximum working time: 3  
  
...Program finished with exit code 0  
Press ENTER to exit console.  
A small white cursor box is visible at the end of the last line.

```
Minimum possible maximum working time: 3  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```