

Binary Search



Search in Rotated Sorted Array

33. Search in Rotated Sorted Array Solved 10

Medium Topics Companies

There is an integer array `nums` sorted in ascending order (with distinct values).

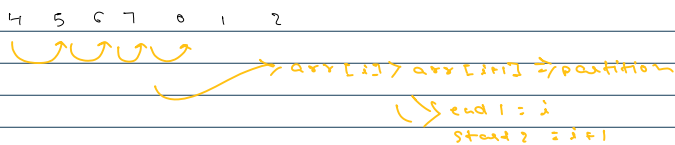
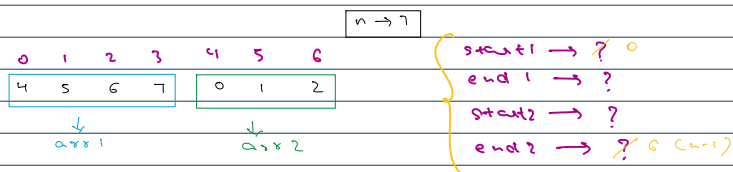
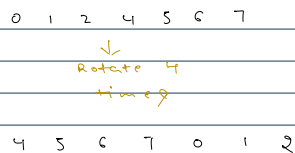
Prior to being passed to your function, `nums` is **possibly rotated** at an unknown pivot index `k` ($1 \leq k < \text{nums.length}$) such that the resulting array is `[nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]]` (**0-indexed**). For example, `[0,1,2,4,5,6,7]` might be rotated at pivot index 3 and become `[4,5,6,7,0,1,2]`.

Given the array `nums` after the possible rotation and an integer `target`, return the index of `target` if it is in `nums`, or `-1` if it is not in `nums`.

You must write an algorithm with $O(\log n)$ runtime complexity.

Example 1:
Input: `nums = [4,5,6,7,0,1,2]`, `target = 0`
Output: `4`

Example 2:
Input: `nums = [4,5,6,7,0,1,2]`, `target = 3`
Output: `-1`



```
public class Main {
    public static void main(String[] args) {

        int[] arr = {2, 3, 4, 5, 7, 8, 11, 13, 13, 16, 17, 18, 22, 24};
        int item = 13;

        int startIndex = 5;
        int endIndex = 8;

        System.out.println(binarySearch(arr, item, startIndex, endIndex));
    }

    public static int binarySearch(int[] arr, int item, int low, int high){

        int n = arr.length;

        while(low<=high){
            int mid = (low+high)/2;
            if(arr[mid]==item){
                return mid;
            }
            else if(arr[mid]>item){
                high = mid - 1;
            }
            else {
                low = mid + 1;
            }
        }
        return -1;
    }
}
```

```

class Solution {
    public int search(int[] nums, int target) {

        int n = nums.length;

        int start1 = 0;
        int end1 = 0;

        int start2 = 0;
        int end2 = n-1;

        for(int i=0; i<n-1; i++){
            if(nums[i] > nums[i+1]){
                end1 = i;
                start2 = i+1;
            }
        }

        System.out.println(start1 + " " + end1 + " " + start2 + " " + end2);

        int ans = binarySearch(nums, target, start1, end1);

        if(ans == -1){
            ans = binarySearch(nums, target, start2, end2);
        }

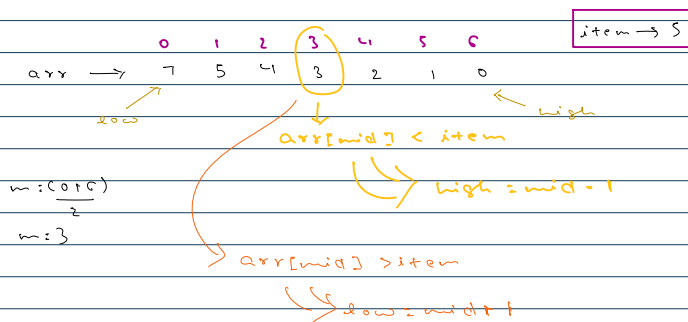
        return ans;
    }

    public static int binarySearch(int[] arr, int item, int low, int high){

        int n = arr.length;

        while(low<=high){
            int mid = (low+high)/2;
            if(arr[mid] == item){
                return mid;
            }
            else if(arr[mid]>item){
                high = mid - 1;
            }
            else {
                low = mid + 1;
            }
        }
        return -1;
    }
}

```



```

public class Main {
    public static void main(String[] args) {

        int[] arr1 = {2, 3, 4, 5, 7, 8, 11, 13, 13, 16, 17, 18, 22, 24};
        int item1 = 13;

        int startIndex1 = 5;
        int endIndex1 = 8;
    }
}

```

```
System.out.println(binarySearchAscending(arr1, item1, startIndex1, endIndex1));
```

```
int[] arr2 = {24, 22, 18, 17, 16, 13, 13, 11, 8, 7, 5, 4, 3, 2};  
int item2 = 13;
```

```
int startIndex2 = 0;  
int endIndex2 = 13;
```

```
System.out.println(binarySearchDescending(arr2, item2, startIndex2, endIndex2));
```

```
}
```

```
public static int binarySearchAscending(int[] arr, int item, int low, int high){
```

```
    int n = arr.length;
```

```
    while(low<=high){  
        int mid = (low+high)/2;  
        if(arr[mid] == item){  
            return mid;  
        }  
        else if(arr[mid]>item){  
            high = mid - 1;  
        }  
        else {  
            low = mid + 1;  
        }  
    }  
    return -1;  
}
```

```
public static int binarySearchDescending(int[] arr, int item, int low, int high){
```

```
    int n = arr.length;
```

```
    while(low<=high){  
        int mid = (low+high)/2;  
        if(arr[mid] == item){  
            return mid;  
        }  
        else if(arr[mid]>item){  
            low = mid + 1;  
        }  
        else {  
            high = mid - 1;  
        }  
    }  
    return -1;  
}
```

```
}
```

Maximum Subarray

53. Maximum Subarray

Given an integer array `nums`, find the `subarray` with the largest sum, and return its sum.

Example 1:
Input: `nums = [-2,1,-3,4,-1,2,1,-5,4]`
Output: `6`
Explanation: The subarray `[4,-1,2,1]` has the largest sum 6.

Example 2:
Input: `nums = [1]`
Output: `1`
Explanation: The subarray `[1]` has the largest sum 1.

Example 3:
Input: `nums = [5,4,-1,7,8]`
Output: `23`
Explanation: The subarray `[5,4,-1,7,8]` has the largest sum 23.

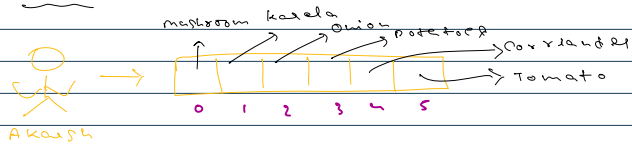
Subarray

It is a contiguous part of an array.

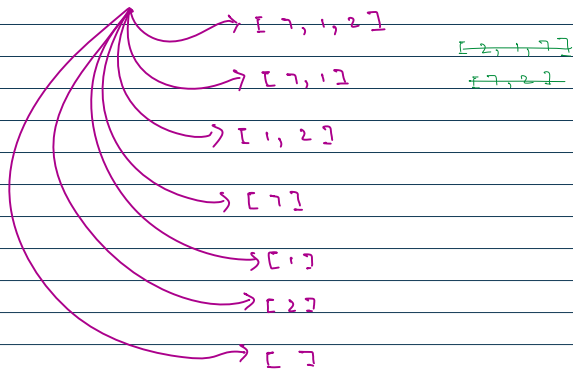
Story

It is a contiguous part of an array.

Story



arr \rightarrow [7, 1, 2]



arr \rightarrow [2, 3, -7, 4, -1, 9]

i = 0

2 \Rightarrow 2
 2 3 \Rightarrow 5
 2 3 -7 \Rightarrow -2
 2 3 -7 4 \Rightarrow 2
 2 3 -7 4 -1 \Rightarrow 1
 2 3 -7 4 -1 9 \Rightarrow 10

i = 1

3 \Rightarrow 3
 3 -7 \Rightarrow -4
 3 -7 4 \Rightarrow 0
 3 -7 4 -1 \Rightarrow -1
 3 -7 4 -1 9 \Rightarrow 8

i = 2

-7 \Rightarrow -7
 -7 4 \Rightarrow -3
 -7 4 -1 \Rightarrow -4
 -7 4 -1 9 \Rightarrow 5

i = 3

4 \Rightarrow 4
 4 -1 \Rightarrow 3
 4 -1 9 \Rightarrow 12

i = 4

-1 \Rightarrow -1
 -1 9 \Rightarrow 8

i = 5

9 \Rightarrow 9

i = 0

2 \Rightarrow 2
 2 3 \Rightarrow 5
 2 3 -7 \Rightarrow -2
 2 3 -7 4 \Rightarrow 2
 2 3 -7 4 -1 \Rightarrow 1
 2 3 -7 4 -1 9 \Rightarrow 10

```
public class Main {
    public static void main(String[] args) {
```

```
        int[] arr = {2, 3, -7, 4, -1, 9};
        int n = arr.length;
```

```
        int sum = 0;
```

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

```
            sum = sum + arr[i];
```

```
            System.out.print(sum + " ");
```

X Loop \Rightarrow 0 to n-1

```
for(int i=0; i<n; i++){
    sum = sum + arr[i];
    System.out.print(sum + " ");
}
```

```
System.out.println();
```

```
sum = 0;
for(int i=1; i<n; i++){
    sum = sum + arr[i];
    System.out.print(sum + " ");
}
```

```
System.out.println();
```

```
sum = 0;
for(int i=2; i<n; i++){
    sum = sum + arr[i];
    System.out.print(sum + " ");
}
```

```
System.out.println();
```

```
sum = 0;
for(int i=3; i<n; i++){
    sum = sum + arr[i];
    System.out.print(sum + " ");
}
```

```
System.out.println();
```

```
sum = 0;
for(int i=4; i<n; i++){
    sum = sum + arr[i];
    System.out.print(sum + " ");
}
```

```
System.out.println();
```

```
sum = 0;
for(int i=5; i<n; i++){
    sum = sum + arr[i];
    System.out.print(sum + " ");
}
```

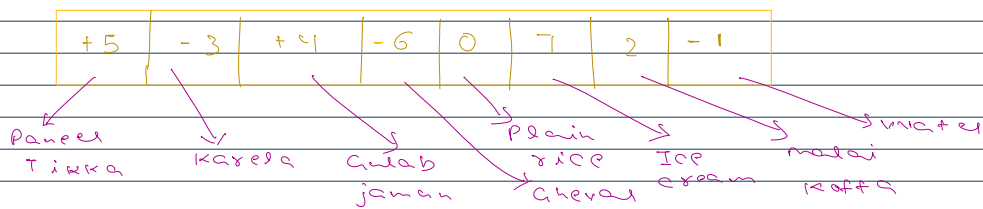
```
}
```

```
}
```

Kadane's Algo



Akash



Maximum P
 cost P → ?
 score P

→ [1, 2]

Loop \Rightarrow 0 to n-1

for(int j=0; j<n; j++) {

}

```

class Solution {
public int maxSubArray(int[] nums) {
    int n = nums.length;
    int ans = Integer.MIN_VALUE;
    int sum = 0;
    for(int j=0; j<n; j++){
        sum = sum + nums[j];
        ans = Math.max(ans, sum);

        if(sum < 0){
            sum = 0;
        }
    }
    return ans;
}
}

```

Product of Array Except Self

238. Product of Array Except Self Solved

Medium Topics Companies Hint

Given an integer array `nums`, return an array `answer` such that `answer[i]` is equal to the product of all the elements of `nums` except `nums[i]`.

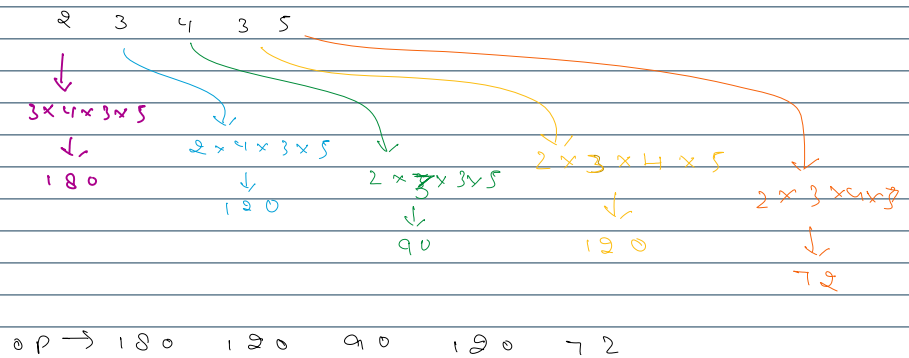
The product of any prefix or suffix of `nums` is guaranteed to fit in a 32-bit integer.

You must write an algorithm that runs in $O(n)$ time and without using the division operation.

Handwritten note: $O(n)$ loop \rightarrow nested loop

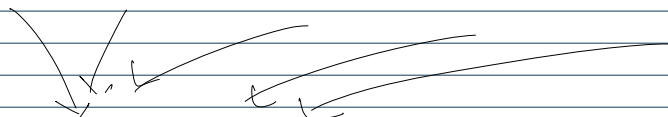
Example 1:
 Input: `nums = [1,2,3,4]`
 Output: `[24,12,8,6]`

Example 2:
 Input: `nums = [-1,1,0,-3,3]`
 Output: `[0,0,9,0,0]`



$$2 \quad 3 \quad 4 \quad 3 \quad 5 \quad \Rightarrow \text{Product} = 360$$

$$\frac{360}{2} \quad \frac{360}{3} \quad \frac{360}{4} \quad \frac{360}{3} \quad \frac{360}{5} = \boxed{180 \quad 120 \quad 90 \quad 120 \quad 72}$$



Product arr[i]

```
public class Main {
```

```
    public static void main(String[] args) {
        int[] arr = {2, 3, 4, 3, 5};
        // System.out.println(product(arr));
```

```
        int[] nums = productOfArrayExceptSelf(arr);
```

```
        for(int i=0; i<nums.length; i++){
            System.out.print(nums[i]+" ");
        }
```

```
    }
```

```
    public static int product(int[] arr){
```

```
        int prod = 1;
        int n = arr.length;
        for(int i=0; i<n; i++){
            prod = prod*arr[i];
        }
        return prod;
```

```
    }
```

```
    public static int[] productOfArrayExceptSelf(int[] arr){
```

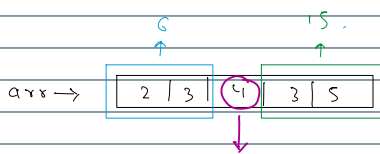
```
        int prod = product(arr);
        int n = arr.length;
        int[] op = new int[n];
```

```
        for(int i=0; i<n; i++){
            op[i] = prod/arr[i];
        }
```

```
        return op;
```

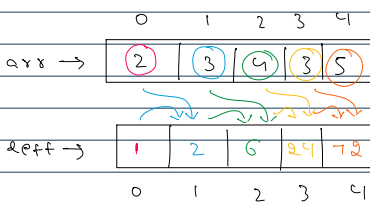
```
    }
```

```
}
```

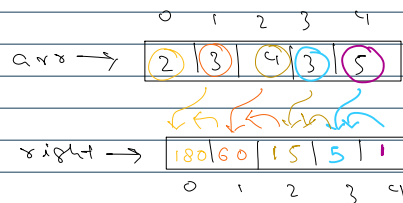


op[2] = product of left array × product of right array

6 × 15 = 90



left[0] = 1



right[n-1] = 1

```

left[1] = left[0] * arr[0]
left[2] = left[1] * arr[1]
left[3] = left[2] * arr[2]
left[4] = left[3] * arr[3]

for(int i=1; i<n; i++) {
    left[i] = left[i-1] * arr[i-1]
}

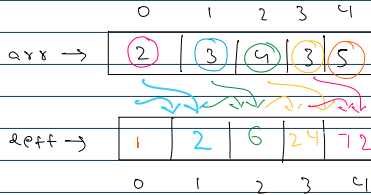
```

```

right[3] = right[4] * arr[4]
right[2] = right[3] * arr[3]
right[1] = right[2] * arr[2]
right[0] = right[1] * arr[1]

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

```



left[0] = 1

$left[1] = left[0] * arr[0]$
 $left[2] = left[1] * arr[1]$
 $left[3] = left[2] * arr[2]$
 $left[4] = left[3] * arr[3]$

```

for(int i=1; i<n; i++) {
    left[i] = left[i-1] * arr[i-1]
}

```

```

class Solution {
public int[] productExceptSelf(int[] nums) {
    int n = nums.length;
    int[] left = new int[n];
    int[] right = new int[n];
    left[0] = 1;
    for(int i=1; i<n; i++){
        left[i] = left[i-1] * nums[i-1];
    }
    right[n-1] = 1;
    for(int i=n-2; i>=0; i--){
        right[i] = right[i+1] * nums[i+1];
    }
    for(int i=0; i<n; i++){
        nums[i] = left[i] * right[i];
    }
    return nums;
}
}

```

Trapping Rain Water

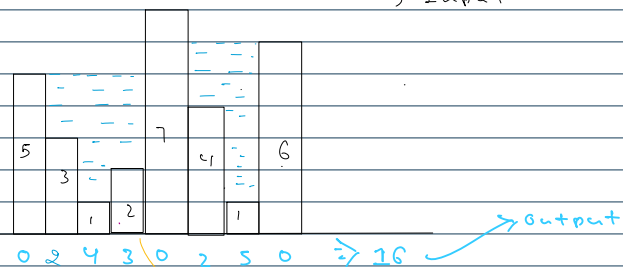
42. Trapping Rain Water Solved

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it can trap after raining.

Example 1:

Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]
Output: 6
Explanation: The above elevation map (black section) is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped.

5 3 1 2 7 4 1 6

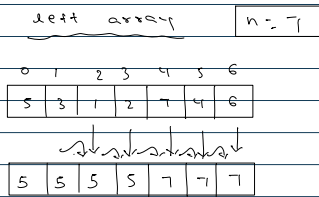


0

1

$left \downarrow$
 5
 $right \downarrow$
 7
 $5 - 2 = 3$
 $arr[i]$
 $min(left, right) \rightarrow 5$

$\min(\text{left}, \text{right}) \Rightarrow 5$
 $\min(5, 7)$



$\text{right}[i+1] = \text{arr}[i+1];$

```

left[0] = arr[0];
left[1] = max(left[0], arr[1]);
left[2] = max(left[1], arr[2]);
left[3] = max(left[2], arr[3]);
left[4] = max(left[3], arr[4]);
left[5] = max(left[4], arr[5]);
left[6] = max(left[5], arr[6]);

```

```

right[6] = arr[6];
right[5] = max(right[6], arr[5]);
right[4] = max(right[5], arr[4]);
right[3] = max(right[4], arr[3]);
right[2] = max(right[3], arr[2]);
right[1] = max(right[2], arr[1]);
right[0] = max(right[1], arr[0]);

```

```

for(int i = n-2; i >= 0; i--) {
    right[i] = max(right[i+1],
                    arr[i]);
}

```

```

class Solution {
public:
    int trap(vector<int> height) {
        int n = height.size();
        vector<int> left(n);
        vector<int> right(n);
        left[0] = height[0];
        for(int i = 1; i < n; i++) {
            left[i] = max(left[i-1], height[i]);
        }
        right[n-1] = height[n-1];
        for(int i = n-2; i >= 0; i--) {
            right[i] = max(right[i+1], height[i]);
        }
        int sum = 0;
        for(int i = 0; i < n; i++) {
            sum = sum + min(left[i], right[i]) - height[i];
        }
        return sum;
    }
}

```