

## Experiment 4

**Student Name:** Niraj Kumar

**Branch:** CSE

**Semester:** 6<sup>th</sup>

**Subject:** AP

**UID:** 22BCS16736

**Section:** 638/A

**DOP:** 11-02-2014

**Subject Code:**22CSP-351

### Aim:

**Problem-1: Beautiful Array**

### Algorithm:

- **Start with Base Case**
- Begin with a list containing only [1] as the base case.
- **Build Odd and Even Sequences**
- Use a divide-and-conquer approach:
  - Generate odd numbers:  $2 * \text{num} - 1$  (as long as they are  $\leq n$ ).
  - Generate even numbers:  $2 * \text{num}$  (as long as they are  $\leq n$ ).
- Append these numbers in order to ensure no three numbers satisfy  $2 * \text{nums}[k] == \text{nums}[i] + \text{nums}[j]$ .
- **Convert List to Array and Return**
- Store the result in an integer array and return it

### Code:

```
import java.util.*;

class Solution {
    public int[] beautifulArray(int n) {
        List<Integer> result = new ArrayList<>();
        result.add(1);

        while (result.size() < n) {
            List<Integer> temp = new ArrayList<>();

            for (int num : result) {
                if (num * 2 - 1 <= n) {
                    temp.add(num * 2 - 1);
                }
            }

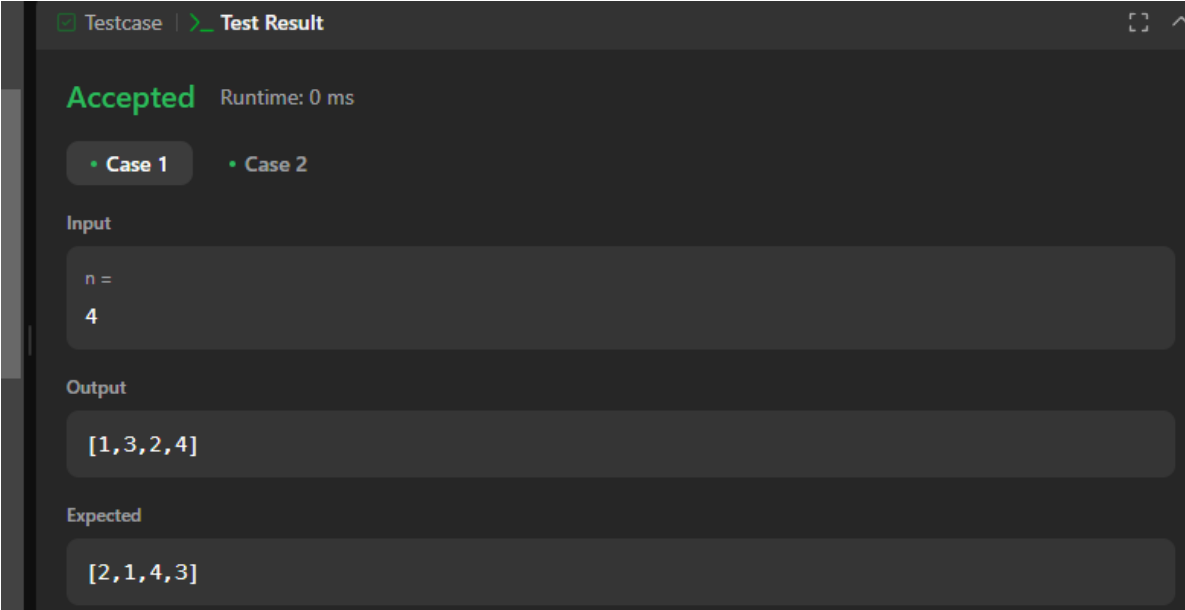
            for (int num : result) {
                if (num * 2 <= n) {
                    temp.add(num * 2);
                }
            }

            result = temp;
        }

        return result.toArray(new int[result.size()]());
    }
}
```

```
temp.add(num * 2);  
}  
}  
  
result = temp;  
}  
  
int[] arr = new int[n];  
for (int i = 0; i < n; i++) {  
    arr[i] = result.get(i);  
}  
  
return arr;  
}  
}
```

## Output:



The screenshot shows a test result interface with a dark theme. At the top, there are tabs for 'Testcase' and 'Test Result', with 'Test Result' being the active tab. Below the tabs, the status 'Accepted' is displayed in green, followed by 'Runtime: 0 ms'. There are two test cases listed: 'Case 1' and 'Case 2'. The 'Input' section shows 'n =' followed by the value '4'. The 'Output' section shows the array '[1, 3, 2, 4]'. The 'Expected' section shows the array '[2, 1, 4, 3]'. The interface includes standard UI elements like a close button and a scroll indicator.

Testcase | Test Result

**Accepted** Runtime: 0 ms

- Case 1
- Case 2

Input

n =  
4

Output

[1, 3, 2, 4]

Expected

[2, 1, 4, 3]



## Aim:

### Problem-2: The Skyline Problem

#### Algorithm :

- Process Building Edges
- Convert each building into two events:
  - Start event (left, -height) → Negative height for priority sorting.
  - End event (right, height) → Positive height to remove it later.
- Sort Events
- Sort by:
  - x-coordinate (ascending).
  - Height (descending for start events, ascending for end events).
- Sweep Line Algorithm with Priority Queue
- Maintain a max-heap (priority queue) to track active building heights.
- Iterate through sorted events:
  - Insert height for a start event.
  - Remove height for an end event.
  - If the max height changes, record the new skyline point.

**Code :**

```
import java.util.*;

class Solution {
    public List<List<Integer>> getSkyline(int[][] buildings) {
        List<int[]> events = new ArrayList<>();

        // Convert buildings to events
        for (int[] b : buildings) {
            events.add(new int[]{b[0], -b[2]}); // Start event (negative height for sorting)
            events.add(new int[]{b[1], b[2]}); // End event
        }

        // Sort events: First by x-coordinate, then height (start before end)
        Collections.sort(events, (a, b) ->
            a[0] != b[0] ? Integer.compare(a[0], b[0]) : Integer.compare(a[1], b[1])
        );

        List<List<Integer>> result = new ArrayList<>();
        PriorityQueue<Integer> maxHeap = new PriorityQueue<>(Collections.reverseOrder());
        maxHeap.add(0); // Ground level

        int prevHeight = 0;

        // Process each event
        for (int[] event : events) {
            int x = event[0], height = event[1];

            if (height < 0) { // Start of a building
                maxHeap.add(-height);
            } else { // End of a building
                maxHeap.remove(height);
            }

            int currentMax = maxHeap.peek();
            if (currentMax != prevHeight) { // If max height changes, record new skyline point
                result.add(Arrays.asList(x, currentMax));
                prevHeight = currentMax;
            }
        }
    }
}
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
        return result;  
    }  
}
```

**Output:**

A screenshot of a test result interface. At the top, there are two tabs: 'Testcase' (with a checkmark icon) and 'Test Result' (with a right-pointing arrow icon). The 'Test Result' tab is active. Below the tabs, the word 'Accepted' is displayed in green, followed by 'Runtime: 1 ms'. There are two buttons: 'Case 1' (selected with a dot) and 'Case 2'. Under the 'Input' section, the text 'buildings =' is followed by a list of arrays: '[ [2,9,10], [3,7,15], [5,12,12], [15,20,10], [19,24,8] ]'. Under the 'Output' section, the text '[ [2,10], [3,15], [7,12], [12,0], [15,10], [20,8], [24,0] ]' is displayed. Under the 'Expected' section, the text '[ [2,10], [3,15], [7,12], [12,0], [15,10], [20,8], [24,0] ]' is displayed. The interface has a dark theme with light gray text and green accents for the 'Accepted' status and the 'Test Result' tab.