Practical-4 DAA

Yash Wanjari

A2_B3_40

Aim: Implement maximum sum of subarray for the given scenario of resource allocation using the divide and conquer approach.

Problem Statement: A project requires allocating resources to various tasks over a period of time. Each task requires a certain amount of resources, and you want to maximize the overall efficiency of resource usage. You're given an array resources where resources[i] represents the amount of resources required for the i th task. Your goal is to find the contiguous subarray of tasks that maximizes the total resources utilized without exceeding a given resource constraint. Handle cases where the total resources exceed the constraint by adjusting the subarray window accordingly. Your implementation should handle various cases, including scenarios where there's no feasible subarray given the constraint and scenarios where multiple subarrays yield the same maximum resource utilization.

Code:-

```
class Result {
  int sum;
  int start;
  int end;

  Result(int sum, int start, int end) {
    this.sum = sum;
    this.start = start;
    this.end = end;
  }
}
```

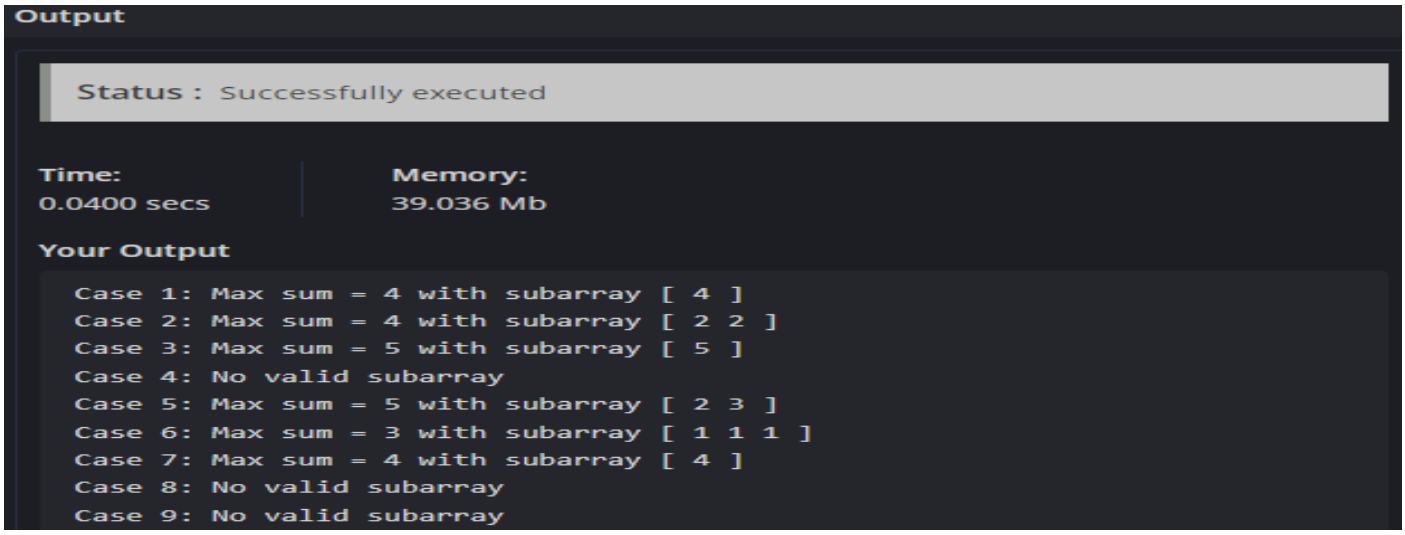
```
public class Main {

    public static Result maxSubarray(int[] arr, int left, int right, int limit) {
        if (left > right) return new Result(Integer.MIN_VALUE, -1, -1);

        if (left == right) {
            return (arr[left] <= limit) ? new Result(arr[left], left, left)
                                        : new Result(Integer.MIN_VALUE, -1, -1);
        }

        int mid = (left + right) / 2;

        Result leftSum = maxSubarray(arr, left, mid, limit);
        Result rightSum = maxSubarray(arr, mid + 1, right, limit);
        Result crossSum = crossSubarray(arr, left, mid, right, limit);

        if (leftSum.sum >= rightSum.sum && leftSum.sum >= crossSum.sum) return leftSum;
        if (rightSum.sum >= leftSum.sum && rightSum.sum >= crossSum.sum) return rightSum;
        return crossSum;
    }

    public static Result crossSubarray(int[] arr, int left, int mid, int right, int limit) {
        int run = 0, bestLeft = Integer.MIN_VALUE, leftPos = -1;
        for (int i = mid; i >= left; i--) {
            run += arr[i];
```

```
    if (run <= limit && run > bestLeft) {
        bestLeft = run;
        leftPos = i;
    }
}

run = 0;
int bestRight = Integer.MIN_VALUE, rightPos = -1;
for (int i = mid + 1; i <= right; i++) {
    run += arr[i];
    if (run <= limit && run > bestRight) {
        bestRight = run;
        rightPos = i;
    }
}

if (bestLeft == Integer.MIN_VALUE && bestRight == Integer.MIN_VALUE)
    return new Result(Integer.MIN_VALUE, -1, -1);
if (bestLeft == Integer.MIN_VALUE) return new Result(bestRight, mid + 1, rightPos);
if (bestRight == Integer.MIN_VALUE) return new Result(bestLeft, leftPos, mid);

int total = bestLeft + bestRight;
if (total <= limit) return new Result(total, leftPos, rightPos);
return (bestLeft >= bestRight) ? new Result(bestLeft, leftPos, mid)
                               : new Result(bestRight, mid + 1, rightPos);
```

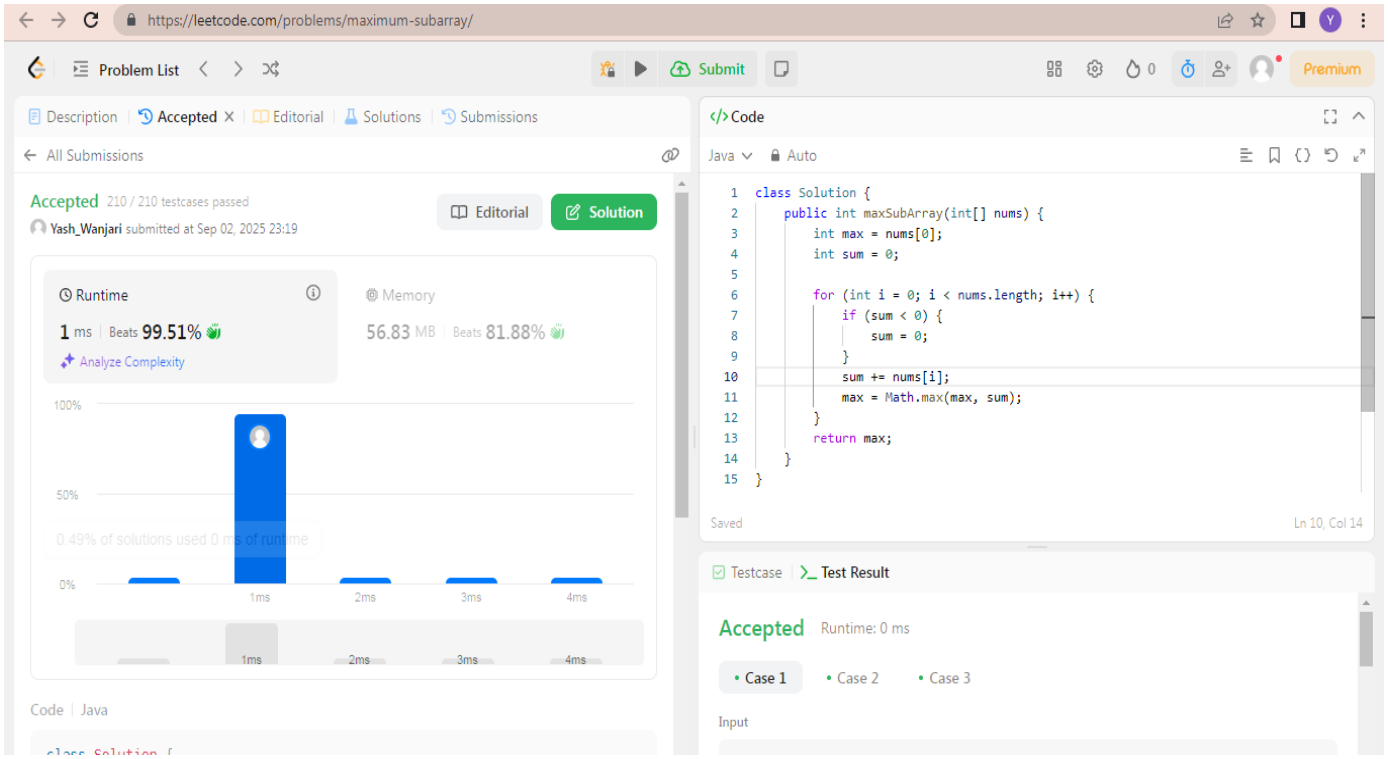
```
}

public static void main(String[] args) {
    int[][] tests = {
        {2, 1, 3, 4},
        {2, 2, 2, 2},
        {1, 5, 2, 3},
        {6, 7, 8},
        {1, 2, 3, 2, 1},
        {1, 1, 1, 1, 1},
        {4, 2, 3, 1},
        {},
        {1, 2, 3}
    };

    int[] limits = {5, 4, 5, 5, 5, 4, 5, 10, 0};

    for (int t = 0; t < tests.length; t++) {
        int[] arr = tests[t];
        int limit = limits[t];

        System.out.print("Case " + (t + 1) + ": ");

        if (arr.length == 0 || limit == 0) {
            System.out.println("No valid subarray");
```

```
            continue;
        }

        Result ans = maxSubarray(arr, 0, arr.length - 1, limit);

        if (ans.sum == Integer.MIN_VALUE) {
            System.out.println("No valid subarray");
        } else {
            System.out.print("Max sum = " + ans.sum + " with subarray [ ");
            for (int i = ans.start; i <= ans.end; i++) {
                System.out.print(arr[i] + " ");
            }
            System.out.println("]");
        }
    }
}
```

O/p



Leetcode:-


https://leetcode.com/problems/maximum-subarray/
Problem List
Submit
Premium
Description
Accepted
Editorial
Solutions
Submissions
All Submissions
Accepted 210 / 210 testcases passed
Yash_Wanjari submitted at Sep 02, 2025 23:19
Editorial
Solution
Runtime
1 ms | Beats 99.51%
Analyze Complexity
Memory
56.83 MB | Beats 81.88%
100%
50%
0%
1ms
2ms
3ms
4ms
Code | Java
Code
Java
Auto
class Solution {
public int maxSubArray(int[] nums) {
int max = nums[0];
int sum = 0;
for (int i = 0; i < nums.length; i++) {
if (sum < 0) {
sum = 0;
}
sum += nums[i];
max = Math.max(max, sum);
}
return max;
}
}
Saved
Ln 10, Col 14
Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Case 3
Input