

TASK 1: Maximum Subarray

AIM

To find the contiguous subarray with the largest sum using brute force and optimized approach.

ALGORITHM

Brute Force:

1. Consider all possible subarrays.
2. Compute sum of each subarray.
3. Store the maximum sum.

Optimized (Kadane's Algorithm):

1. Initialize maxSum and currentSum with first element.
2. Traverse array and update currentSum = $\max(\text{arr}[i], \text{currentSum} + \text{arr}[i])$.
3. Update maxSum = $\max(\text{maxSum}, \text{currentSum})$.

PROCEDURE

1. Input an integer array.
2. Apply brute force and optimized algorithm.
3. Print maximum subarray sum.

PROGRAM (Java – Optimized)

```
class Solution {  
  
    public int maxSubArray(int[] nums) {  
  
        int maxSum = nums[0];  
  
        int currSum = nums[0];  
  
  
        for(int i = 1; i < nums.length; i++) {  
  
            currSum = Math.max(nums[i], currSum + nums[i]);  
  
            maxSum = Math.max(maxSum, currSum);  
  
        }  
  
        return maxSum;  
    }  
}
```

```
    }  
}
```

OUTPUT

The screenshot shows a code editor interface with a toolbar at the top. The toolbar has two tabs: 'Testcase' (with a checked checkbox) and 'Test Result'. Below the toolbar, there are three checkboxes labeled 'Case 1', 'Case 2', and 'Case 3', all of which are checked. The main area is divided into two sections: 'Input' and 'Output'. The 'Input' section contains the following code:

```
nums =  
[-2,1,-3,4,-1,2,1,-5,4]
```

The 'Output' section contains the number '6'.

RESULT

The program successfully finds the maximum subarray sum.

TASK 2: Birthday Bar

AIM

To determine the number of contiguous segments whose sum equals Ron's birth day and month.

ALGORITHM

1. Loop through array.
2. For each index, calculate sum of next m elements.
3. If sum equals d, increment count.

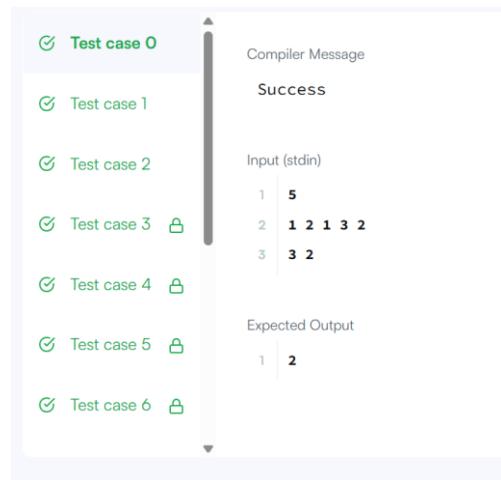
PROCEDURE

1. Read array, day (d), month (m).
2. Check each subarray of length m.
3. Print count.

PROGRAM (Java)

```
static int birthday(List<Integer> s, int d, int m) {  
    int count = 0;  
  
    for(int i = 0; i <= s.size() - m; i++) {  
  
        int sum = 0;  
  
        for(int j = i; j < i + m; j++) {  
  
            sum += s.get(j);  
  
        }  
  
        if(sum == d) count++;  
  
    }  
  
    return count;  
}
```

OUTPUT



The screenshot shows a list of test cases on the left and their results on the right. Test case 0 is successful. Test case 1 is successful. Test case 2 is successful. Test case 3 is successful, with input 5 and output 2. Test case 4 is successful. Test case 5 is successful, with input 1 2 1 3 2 and output 2. Test case 6 is successful.

Test Case	Result	Input (stdin)	Expected Output
Test case 0	Success		
Test case 1	Success		
Test case 2	Success		
Test case 3	Success	5	2
Test case 4	Success		
Test case 5	Success	1 2 1 3 2	2
Test case 6	Success	3 2	

RESULT

The program outputs the number of valid chocolate segments.

TASK 3: Max Subarray

AIM

To find maximum subarray and subsequence sums.

ALGORITHM

1. Use Kadane's algorithm for subarray sum.
2. For subsequence, sum all positive numbers.

PROCEDURE

1. Read test cases and arrays.
2. Apply Kadane's algorithm.
3. Print both results.

PROGRAM (Java)

```
static void maxSubarray(int[] arr) {  
  
    int maxSub = arr[0], curr = arr[0];  
  
    int maxSeq = 0;  
  
    boolean allNeg = true;  
  
    int maxVal = arr[0];  
  
  
    for(int x : arr) {  
  
        if(x > 0) { allNeg = false; maxSeq += x; }  
  
        maxVal = Math.max(maxVal, x);  
  
    }  
  
  
    for(int i = 1; i < arr.length; i++) {  
  
        curr = Math.max(arr[i], curr + arr[i]);  
  
        maxSub = Math.max(maxSub, curr);  
  
    }  
  
  
    if(allNeg) maxSeq = maxVal;  
  
    System.out.println(maxSub + " " + maxSeq);  
  
}
```

OUTPUT

The screenshot shows a code editor interface with several test cases. Test case 0 is a compiler message indicating success. Test cases 1 through 4 show input arrays and their corresponding output sums.

Test Case	Input (stdin)	Output
Test case 0	Compiler Message	Success
Test case 1		
Test case 2	1 6	6
Test case 3	2 1 3 1	1
Test case 4	4 6 5 -1 -2 -3 -4 -5 -6 6 2 7 1 -2 8 3 9 1 2 3	13

RESULT

The program prints maximum subarray and subsequence sums.

TASK 4: Maximum Sum Circular Subarray

AIM

To find maximum circular subarray sum.

ALGORITHM

1. Find normal max subarray using Kadane.
2. Find minimum subarray sum.
3. Circular sum = totalSum – minSubarray.
4. Answer = max(normalMax, circularMax).

PROCEDURE

1. Input array.
2. Apply Kadane's algorithm twice.
3. Print maximum sum.

PROGRAM (Java)

```
class Solution {  
  
    public int maxSubarraySumCircular(int[] nums) {  
  
        int total = 0;
```

```

int maxSum = nums[0], currMax = 0;

int minSum = nums[0], currMin = 0;

for(int n : nums) {

    currMax = Math.max(n, currMax + n);

    maxSum = Math.max(maxSum, currMax);

    currMin = Math.min(n, currMin + n);

    minSum = Math.min(minSum, currMin);

    total += n;

}

if(maxSum < 0) return maxSum;

return Math.max(maxSum, total - minSum);

}

```

OUTPUT

Testcase | Test Result

Case 1 Case 2 Case 3

Input

```
nums =
[1, -2, 3, -2]
```

Output

```
3
```

RESULT

The program returns maximum circular subarray sum.

TASK 5: String to Integer (atoi)

AIM

To convert a string to integer following given rules.

ALGORITHM

1. Remove leading spaces.
2. Check sign.
3. Convert digits until non-digit.
4. Handle overflow.

PROCEDURE

1. Input string.
2. Parse characters and compute integer.
3. Return integer value.

PROGRAM (Java)

```
class Solution {

    public int myAtoi(String s) {
        s = s.trim();
        if(s.length() == 0) return 0;

        int sign = 1, i = 0;
        long res = 0;

        if(s.charAt(0) == '-') { sign = -1; i++; }
        else if(s.charAt(0) == '+') i++;

        while(i < s.length() && Character.isDigit(s.charAt(i))) {
            res = res * 10 + (s.charAt(i) - '0');
            if(res * sign > Integer.MAX_VALUE) return Integer.MAX_VALUE;
            if(res * sign < Integer.MIN_VALUE) return Integer.MIN_VALUE;
        }
    }
}
```

```
i++;  
}  
  
return (int)(res * sign);  
}  
}
```

OUTPUT

The screenshot shows a code editor interface with a toolbar at the top. The toolbar has a checked checkbox labeled "Testcase" and an arrow icon labeled "Test Result". Below the toolbar, there are five checkboxes labeled "Case 1" through "Case 5", with "Case 1" being checked. A large input field below the toolbar contains the string "s = \"42\"". An output field below the input field contains the number "42".

Output

42

RESULT

The program correctly converts string to integer.

TASK 6: Alternating Characters

AIM

To find minimum deletions required to make string alternating.

ALGORITHM

1. Compare adjacent characters.
2. If same, increment deletion count.

PROCEDURE

1. Input string.
2. Traverse string and count duplicates.
3. Print deletions.

PROGRAM (Java)

```
static int alternatingCharacters(String s) {  
    int count = 0;  
  
    for(int i = 1; i < s.length(); i++) {  
  
        if(s.charAt(i) == s.charAt(i - 1)) count++;  
    }  
  
    return count;  
}
```

OUTPUT

✓ Test case 0	Compiler Message
✓ Test case 1	Success
✓ Test case 2	Input (stdin)
✓ Test case 3	1 5 2 AAAA 3 BBBBB
✓ Test case 4	4 ABABABAB 5 BABABA
✓ Test case 5	6 AAABBB
✓ Test case 6	Expected Output ▼ 1 3

RESULT

The program outputs required deletions.

TASK 7: Longest Substring Without Repeating Characters

AIM

To find length of longest substring without repeated characters.

ALGORITHM

1. Use sliding window with HashSet.
2. Move right pointer and remove duplicates from left.

PROCEDURE

1. Input string.
2. Apply sliding window technique.
3. Print maximum length.

PROGRAM (Java)

```
class Solution {  
  
    public int lengthOfLongestSubstring(String s) {  
  
        Set<Character> set = new HashSet<>();  
  
        int l = 0, max = 0;  
  
        for(int r = 0; r < s.length(); r++) {  
  
            while(set.contains(s.charAt(r))) {  
  
                set.remove(s.charAt(l++));  
  
            }  
  
            set.add(s.charAt(r));  
  
            max = Math.max(max, r - l + 1);  
  
        }  
  
        return max;  
    }  
}
```

OUTPUT

Testcase | [Test Result](#)

Case 1 Case 2 Case 3

Input

```
s =
"abcabcbb"
```

Output

```
3
```

RESULT

The program prints the longest substring length.

TASK 8: Find and Replace Pattern

AIM

To find words matching a pattern using bijection mapping.

ALGORITHM

1. For each word, map characters to pattern characters.
2. Ensure one-to-one mapping.
3. Store matching words.

PROCEDURE

1. Input words array and pattern.
2. Check mapping using HashMap.
3. Return matching words.

PROGRAM (Java)

```
class Solution {

    public List<String> findAndReplacePattern(String[] words, String pattern) {

        List<String> res = new ArrayList<>();

        for(String w : words) {
            if(match(w, pattern)) res.add(w);
        }

        return res;
    }

    private boolean match(String w, String p) {
        Map<Character, Character> m1 = new HashMap<>();
        Map<Character, Character> m2 = new HashMap<>();
```

```

for(int i = 0; i < w.length(); i++) {
    char c1 = w.charAt(i), c2 = p.charAt(i);
    if(m1.containsKey(c1) && m1.get(c1) != c2) return false;
    if(m2.containsKey(c2) && m2.get(c2) != c1) return false;
    m1.put(c1, c2);
    m2.put(c2, c1);
}
return true;
}
}

```

OUTPUT

Testcase | [Test Result](#)

Accepted Runtime: 0 ms

Case 1 Case 2

Input

```
words =
["abc", "deq", "mee", "aqq", "dkd", "ccc"]
```

RESULT

The program outputs all words matching the pattern.

TASK 9: String Matching in an Array

AIM

To find all strings that are substrings of other strings.

ALGORITHM

1. Compare each string with others.
2. If one string is found inside another, add to result.

PROCEDURE

1. Input array of strings.

2. Use nested loops to check substring.

3. Print result list.

PROGRAM (Java)

```
class Solution {  
  
    public List<String> stringMatching(String[] words) {  
  
        List<String> res = new ArrayList<>();  
  
        for(int i = 0; i < words.length; i++) {  
  
            for(int j = 0; j < words.length; j++) {  
  
                if(i != j && words[j].contains(words[i])) {  
  
                    res.add(words[i]);  
  
                    break;  
                }  
            }  
        }  
  
        return res;  
    }  
}
```

OUTPUT

Testcase | 

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

```
words =  
["mass","as","hero","superhero"]
```

TASK 10: Naive Pattern Search

AIM