Day 39/180

Binary Search Hard Interview Problem 📑

Kadane's Algorithm

```
class Solution{
    public:
        long long maxSubarraySum(int arr[], int n) {
            long long maxi = INT_MIN, prefix = 0;
            // Iterate through the array elements.
            for (int i = 0; i < n; i++) {
                // Add the current element to the current subarray sum.
                prefix += arr[i];

                // Update maxi with the maximum of the current subarray
sum and maxi.
                maxi = max(prefix, maxi);

                // If the current subarray sum becomes negative, reset it
to 0,
                // as including negative sum would only decrease the
overall sum.
                if (prefix < 0)
                    prefix = 0;
            }

            // Return the maximum subarray sum.
            return maxi;
        }
};
```

Code Explanation and Complexity

1. maxi Variable:
    - It keeps track of the maximum subarray sum encountered so far.
    - Initialized to INT_MIN to ensure the first element is considered as a potential maximum.
2. prefix Variable:
    - Represents the current sum of the subarray.
    - Accumulates the sum as the loop iterates through the array.

3. Inside the Loop:
    - prefix += arr[i];: Adds the current element to the current subarray sum.
    - maxi = max(prefix, maxi);: Updates maxi with the maximum of the current subarray sum and the previously recorded maximum.
    - if (prefix < 0) prefix = 0;: If the current subarray sum becomes negative, resets it to 0.
        - Ensures considering a new subarray if the previous one has a negative sum.
4. After the Loop:
    - The function returns the maximum subarray sum (maxi).
5. Time Complexity:
    - The time complexity is O(N), where N is the number of elements in the array.
        - The algorithm processes each element of the array once.
6. Space Complexity:
    - The space complexity is O(1).
        - The algorithm uses a constant amount of extra space regardless of the input size.

Maximum difference between two elements such that larger element appears after the smaller number

```
#include <bits/stdc++.h>
using namespace std;

/* The function assumes that there are
at least two elements in array. The
function returns a negative value if the
array is sorted in decreasing order and
returns 0 if elements are equal */
int maxDiff(int arr[], int arr_size)
{
     // Maximum difference found so far
     int max_diff = arr[1] - arr[0];

     // Minimum number visited so far
     int min_element = arr[0];
     for(int i = 1; i < arr_size; i++)
     {
          // Update max_diff if a larger difference is found
          if (arr[i] - min_element > max_diff)

               max_diff = arr[i] - min_element;

          // Update min_element if a smaller element is found
          if (arr[i] < min_element)
               min_element = arr[i];
     }

     return max_diff;
}

int main()
{
    // Example array
    int arr[] = {1, 2, 90, 10, 110};
    int n = sizeof(arr) / sizeof(arr[0]);

    // Function calling
    cout << "Maximum difference is " << maxDiff(arr, n);

    return 0;
}
```

Code Explanation and Complexity

1. maxDiff Function:
    - Iterates through the array and keeps track of the maximum difference (max_diff) and the minimum element encountered so far (min_element).
    - Updates max_diff if a larger difference is found.
    - Updates min_element if a smaller element is encountered.
2. Time Complexity:
    - O(n), where n is the size of the array, as the algorithm iterates through the array once.
3. Space Complexity:
    - O(1), indicating a constant amount of extra space used regardless of the input size.
4. Handling Sorted Arrays:
    - The function assumes there are at least two elements in the array. It returns a negative value if the array is sorted in decreasing order and returns 0 if elements are equal.
5. Example Usage:
    - The provided example array is {1, 2, 90, 10, 110}.
    - The function is called with this array, and the maximum difference is printed.

Maximum prefix sum for a given range

```cpp
#include <vector>
#include <climits>

class Solution {
public:
    vector<int> maxPrefixes(int a[], int L[], int R[], int N, int Q) {
        int j = 0;
        vector<int> ans;

        // Iterate through each query
        while (Q != 0) {
            int prefix = 0;
            int maxi = INT_MIN;

            // Iterate within the specified range [L[j], R[j]]
            for (int i = L[j]; i <= R[j]; i++) {
                prefix += a[i];
                maxi = max(maxi, prefix);
            }

            // Move to the next query
            j++;

            // Store the maximum prefix-sum for the current query
            ans.push_back(maxi);

            // Reduce the count of remaining queries
            Q--;
        }

        return ans;
    }
};
```

Code Explanation and Complexity

1. maxPrefixes Function:
    - Parameters: a[] - array of integers, L[] and R[] - arrays representing the ranges for each query, N - size of the array, Q - number of queries.
    - Returns a vector containing the maximum prefix-sum for each query.
2. Initialization:

    - j is the index used to iterate through the queries.
    - ans is a vector to store the results of each query.
3. Query Iteration (while loop):
    - Iterates through each query until Q becomes 0.
4. Range Iteration (for loop):
    - For each query, initializes prefix and maxi to 0 and INT_MIN, respectively.
    - Iterates through the specified range [L[j], R[j]] in the array.
    - Updates prefix by adding the current element to it.
    - Updates maxi with the maximum of the current prefix sum and the previous maxi.
5. Storing Results:
    - After processing the range for a query, moves to the next query (j++).
    - The maximum prefix-sum for the current query is stored in the ans vector.
    - The count of remaining queries is reduced (Q--).
6. Result:
    - The function returns the vector ans containing the maximum prefix-sum for each query.
7. Time Complexity:
    - Expected time complexity is O(N*Q), where N is the size of the array and Q is the number of queries.
    - The code iterates through each query and processes the specified range in the array.
8. Space Complexity:
    - Expected auxiliary space complexity is O(1).

Equal Sums

```
class Solution {
public:
    vector<int> EqualSum(int a[], int n) {
        int minDiff = INT_MAX, minDiffIndex = -1, subArray = 1, totalSum
= 0, prefix = 0;

        // Calculate total sum of the array
        for (int i = 0; i < n; ++i)
            totalSum += a[i];

        // Iterate through the array to find the minimum difference
        for (int i = 0; i < n; ++i) {
            prefix += a[i];

            // Calculate the sum on the right side of the current element
            int rightSum = totalSum - prefix;
            // Calculate the absolute difference between left and right sums
            int diff = abs(rightSum - prefix);

            // Update minimum difference information if the current difference is smaller
            if (diff < minDiff) {
                minDiff = diff;
                minDiffIndex = i + 1;

                // Determine the subarray in which the new element will be included
                if (rightSum >= prefix)
                    subArray = 1;
                else
                    subArray = 2;
            }
        }

        // Return the result as a vector
        return {minDiff, minDiffIndex + 1, subArray};
    }
};
```

Code Explanation and Complexity

- EqualSum Function:
    - Parameters: a[] - array of integers, n - size of the array.
    - Returns a vector containing three values: minimum difference, its index, and the subarray number.
- Initialization:
    - minDiff, minDiffIndex, and subArray are used to track the minimum difference, its index, and the subarray number.
    - totalSum is calculated as the sum of all elements in the array.
    - prefix is used to calculate the sum of elements occurring before the current index.
- Total Sum Calculation:
    - Iterates through the array to calculate the total sum.
- Minimum Difference Calculation:
    - Iterates through the array to find the minimum difference between the sum of elements occurring before and after the current index.
    - Updates the minimum difference information if the current difference is smaller.
    - Determines the subarray in which the new element will be included based on the comparison of right and left sums.

- Result:
    - Returns the result as a vector containing the minimum difference, its index, and the subarray number.
- Time Complexity:
    - O(n), where n is the size of the array.
    - The code iterates through the array twice, and each iteration takes O(n) time.
- Space Complexity:
    - O(1).
    - The code uses a constant amount of extra space regardless of the input size. The space required is independent of the array size.