🔢 Integer Overflow Handling

- ☑️ 1LL * a * b: Ensures multiplication stays in long long
- ☑️ LLONG_MIN, LLONG_MAX: Use for initializing min/max of long long
- ☑️ INT_MIN, INT_MAX: For int range checks

🔁 Backtracking Tips

- Pass vector<int>& temp by **reference**
- Use push_back() → recursive call → pop_back() (undo step)
- Avoid copying vectors in recursion (memory efficient)

✅ Structure: Find All Subsequences (with if branching)

```
void findSubsequences(vector<int>& nums, int index, vector<int>& temp) {
  // Base case: we've considered all elements
  if (index == nums.size()) {
    // Do something with the current subsequence
    return;
  }

  // ☑️ Include current element
  temp.push_back(nums[index]);
  findSubsequences(nums, index + 1, temp);
  temp.pop_back(); // backtrack

  // ❌ Exclude current element
  findSubsequences(nums, index + 1, temp);
}
```

💉 Example Usage:

```
int main() {
```

```
    vector<int> nums = {1, 2, 3};
    vector<int> temp;
    findSubsequences(nums, 0, temp);
    return 0;
}
```

[Maximum Product of First and Last Elements of a Subsequence]

```
// Recursive helper to explore subsequences of size 'm'
void find(vector<int>& nums, int m, int index, vector<int>& temp, long long &maxprod) {
    if (temp.size() == m) {
        // Only multiply when you have 'm' elements in the temp vector
        long long pro = 1LL * temp[0] * temp[m - 1]; // use 1LL to avoid overflow
        maxprod = max(maxprod, pro);
        return;
    }

    // Base case: if index goes out of bounds
    if (index == nums.size()) return;



    // Include current element
    temp.push_back(nums[index]);
    find(nums, m, index + 1, temp, maxprod);

    // Exclude current element (backtrack)
    temp.pop_back();
    find(nums, m, index + 1, temp, maxprod);
}

// Main function
```

```
long long maximumProduct(vector<int>& nums, int m) {
    long long maxprod = LLONG_MIN;  // Set to minimum to handle all cases
    vector<int> temp;
    find(nums, m, 0, temp, maxprod);
    return maxprod;
}
```

IN JAVA:

```
class MaxHolder {
    long value = Long.MIN_VALUE;
}

public static void find(int[] nums, int m, int index, int[] temp, int tempIndex, MaxHolder max) {
    if (tempIndex == m) {
        long prod = 1L * temp[0] * temp[m - 1];
        max.value = Math.max(max.value, prod);
        return;
    }
    if (index == nums.length) return;

    temp[tempIndex] = nums[index];
    find(nums, m, index + 1, temp, tempIndex + 1, max);

    find(nums, m, index + 1, temp, tempIndex, max);
}

public static long maximumProduct(int[] nums, int m) {
    int[] temp = new int[m];
    MaxHolder max = new MaxHolder();
```

```
    find(nums, m, 0, temp, 0, max);
    return max.value;
}
```

PREFIX SUM:

prefix[i] = prefix[i-1] + arr[i]

Sum of the elements in arr in the Range of [l ... r] = Prefix[r] - prefix[l-1]

C++:

```
// Function to build prefix sum array
vector<int> prefixSum(vector<int>& nums) {
    int n = nums.size();
    vector<int> prefix(n);
    prefix[0] = nums[0];
    for (int i = 1; i < n; ++i) {
        prefix[i] = prefix[i - 1] + nums[i];
    }
    return prefix;
}

// Function to get sum in range [l, r]
int optimizedSum(vector<int>& prefix, int l, int r) {
    if (l == 0) return prefix[r];
    return prefix[r] - prefix[l - 1];
}
```

Time Complexity: O(n), as we are traversing the array only once.
Auxiliary Space: O(n), to create the array prefxSum[] of size n.

JAVA:

```
import java.util.*;
public class PrefixSumExample {
  // Method to calculate prefix sum
  public static int[] prefixSum(int nums[]) {
    int n = nums.length;
    int prefix[] = new int[n];
    prefix[0] = nums[0];
    for (int i = 1; i < n; i++) {
      prefix[i] = prefix[i - 1] + nums[i];
    }
    return prefix;
  }
  // Method to get sum from index l to r (inclusive)
  public static int optimizedSum(int[] prefix, int l, int r) {
    if (l == 0) return prefix[r];
    return prefix[r] - prefix[l - 1];
  }
  public static void main(String[] args) {
    int nums[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    int prefix[] = prefixSum(nums);
    System.out.print("Prefix Sum Array: ");
    for (int num : prefix) {
      System.out.print(num + " ");
    }
    System.out.println();
    int l = 0, r = 9;
    System.out.println("Optimized Sum from index " + l + " to " + r + ": " +
optimizedSum(prefix, l, r));
  }
}
```

Find count of number of subarrays with sum == k

Brute Force — $O(n^3)$

```
int countSubarraysBrute(vector<int>& nums, int k) {
   int n = nums.size(), count = 0;
   for(int i = 0; i < n; i++) {
      for(int j = i; j < n; j++) {
         int sum = 0;
         for(int l = i; l <= j; l++) {
            sum += nums[l];
         }
         if(sum == k) count++;
      }
   }
   return count;
}
```

BETTER: $O(n^2)$

```
int countSubarraysBetter(vector<int>& nums, int k) {
   int n = nums.size(), count = 0;
   for(int i = 0; i < n; i++) {
      int sum = 0;
      for(int j = i; j < n; j++) {
         sum += nums[j];
         if(sum == k) count++;
      }
   }
   return count;
}
```

Optimal — O(n) using Prefix Sum + HashMap

अगर किसी पॉइंट पर prefix[j] - prefix[i] = k हो, तो इसका मतलब i+1 से j तक की subarray का योग k है।

```
int countSubarrays(vector<int> &arr, int k) {
    int count = 0;  // To store the total number of valid subarrays
    int sum = 0;    // To store the running prefix sum
    unordered_map<int, int> prefixsum;

    prefixsum[0] = 1;  // Initialize with sum 0 to handle subarrays starting from index 0

    for (int i = 0; i < arr.size(); i++) {
        sum += arr[i];  // Update prefix sum with current element

        // Check if there is a prefix sum such that (sum - k) exists in map
        // If yes, it means there exists a subarray ending at index i with sum = k
        if (prefixsum.find(sum - k) != prefixsum.end()) {
            count += prefixsum[sum - k];  // Add frequency of (sum - k) to count
        }

        prefixsum[sum]++;  // Record the current prefix sum for future matches
    }

    return count;  // Return total count of subarrays with sum == k
}
```

❓ **Jab aap dekhte ho:**

sum - k

Toh iska matlab hota hai:

"Kya pehle kahin koi aisa subarray tha jiska sum (sum - k) tha?"

🔍 **Kyu check kar rahe ho sum - k?**

- **sum hai ab tak ka total sum (index 0 se current index tak).**
- **Agar pehle kahin sum - k mila tha,**
  toh iska matlab hai ki us point ke baad se lekar ab tak ka subarray ka sum exact k hoga.

🔢 **Ab samjho: sum - k == 0 kya matlab?**

Agar sum - k == 0, toh matlab:
Index 0 se current index tak ka pura subarray ka sum exactly k hai.

Longest Subarray with Sum K

BRUTE FORCE:O(N^2) TLE

```
int longestSubarray(vector<int>& nums, int target) {
    int n=nums.size();
    int maxlen=INT_MIN;

    for(int i=0;i<n;i++){
      int sum=0;
      int len=0;
      for(int j=i;j<n;j++){
        sum+=nums[j];
        len++;
        if(sum==target){
          maxlen=max(maxlen,len);  or maxlen=max(maxlen,j-i+1);
        }
      }
    }
    return maxlen==INT_MIN?0:maxlen;
  }
```

OPTIMIZED:

Using Hash Map and Prefix Sum - O(n) Time and O(n) Space

```
int longestSubarray(vector<int>& nums, int k) {
    unordered_map<int, int> prefixIndex;
int sum = 0, maxLen = 0;

for (int i = 0; i < nums.size(); i++) {
  sum += nums[i];

  // If sum is exactly k, from index 0 to i or // Check if the entire prefix sums to k

  if (sum == k) {
    maxLen = i + 1;
  }

  // If (sum - k) is seen before, update maxLen
  if (prefixIndex.find(sum - k) != prefixIndex.end()) {
    int len = i - prefixIndex[sum - k];
    maxLen = max(maxLen, len);
  }

  // Only store first occurrence of sum as We want the longest subarray whose sum is exactly k
  if (prefixIndex.find(sum) == prefixIndex.end()) {
    prefixIndex[sum] = i;
  }
}

return maxLen;
}
```

Minimum Size Subarray Sum i.e Smallest Subarray with Sum ≥ K (Leetcode)

Brute force O(N^2) TLE:

```
int minSubArrayLen(int target, vector<int>& nums) {
    int n=nums.size();
    int minlen=INT_MAX;

    for(int i=0;i<n;i++){
      int sum=0;
      int len=0;
      for(int j=i;j<n;j++){
        sum+=nums[j];
        len++;
        if(sum>=target){
          minlen=min(minlen,len);
          break;
        }
      }
    }
    return minlen==INT_MAX?0:minlen;
  }
```

OPTIMAL:(Use sliding window (two pointers) Time Complexity: O(n) Space Complexity: O(1)

```
int minSubArrayLen(int target, vector<int>& nums) {
    int n=nums.size();
    int minlen=INT_MAX;
    int i=0,j=0;
    int sum=0; // Window ka current sum
```

```
    while(j<n){
        sum+=nums[j]; // j pointer ka element window mein include kiya
        while(sum>=target){ // window ko chota jitna kar sake karte jayenge
            minlen=min(minlen,j-i+1);
            sum-=nums[i];
            i++;
        }
        j++;
    }

    return minlen==INT_MAX?0:minlen;
}
```