

+ New chat

Array Manipulation Sol  

## Factorial Calculation Functions

💬 Create int arrays.

Learning Data Structures

Student Record Object Creation

Java program for collection pe


 New chat

Print prime numbers (Python)

 Clear conversations

 Upgrade to Plus

NEW

 Dark mode

[🔗 Updates & FAQ](#)

[→ Log out



```
long arrayManipulation(int n, vector<vector<int>> queries) {
```

```
int no_of_operations = queries.size();
vector<long> final_array(n,0);
long max = 0;

for (int i=0; i<no_of_operations; i++){
    for (int j=queries[i][0]; j<=queries[i][1]; j++){
        final_array[j-1] += queries[i][2];
        if (final_array[j-1]>max){
            max = final_array[j-1];
        }
    }
}

return max;
}
```



This code takes in an integer  $n$ , which represents the length of an initial array, and a vector of vector queries. Each query is represented as a vector with 3 integers: a starting index, an ending index, and a value. The code performs a series of operations on the initial array

queries are performed.

Next, the code iterates over each query in the `queries` vector using a `for` loop. For each query, the code iterates over a range of indices in the `final_array`. The range of indices is determined by the starting and ending indices in the query vector. For each index in this range, the value in `final_array` at that index is updated by adding the value in the third position of the query vector.

Finally, the code checks if the updated value at each index is greater than the current maximum value. If it is, the updated value becomes the new maximum value.

The code returns the maximum value found in the `final_array` after all of the queries have been performed.

This implementation of the array manipulation problem using nested loops has a time complexity of  $O(n*m)$ , where  $n$  is the length of the array and  $m$  is the number of queries. However, there is an optimized solution using prefix sums that has a time complexity of  $O(n+m)$ .

🔄 Regenerate response



To optimize the time complexity of this algorithm, we can use the concept of prefix sums.




Instead of iterating through each index in the range of indices for each query and updating the corresponding value in the `final_array`, we can use the prefix sum technique to update the values in the `final_array` in constant time.

The prefix sum technique involves creating an auxiliary array that holds the cumulative sum of the original array up to each index. This can be done in linear time by iterating through the original array and updating the auxiliary array at each index.

Once we have the auxiliary array, we can process each query by simply adding the query value to the starting index of the final\_array and subtracting the query value from the index after the ending index. This can also be done in constant time.

After processing all the queries, we can iterate through the `final_array` and compute the cumulative sum up to each index to get the resulting array.


🔄 Regenerate response

```
long arrayManipulation(int n, vector<vector<int>> queries) {
    vector<long> final array(n+1,0);
```

```
// Update final_array using prefix sum technique
for (int i=0; i<queries.size(); i++){
    int a = queries[i][0];
    int b = queries[i][1];
    int k = queries[i][2];
    final_array[a] += k;
    if (b+1 <= n) final_array[b+1] -= k;
}

// Compute cumulative sum of final_array
long max = 0;
long sum = 0;
for (int i=1; i<=n; i++){
    sum += final_array[i];
    if (sum > max) max = sum;
}

return max;
}
```

 Regenerate response



[ChatGPT Mar 14 Version](#). Free Research Preview. Our goal is to make AI systems more natural and safe to interact with. Your feedback will help us improve.