SH

Hi please see below and explain me first then
give me the complete code
Using dynamic programming
Solution idea
Let’s take an array maxSumEnding[] of size n, where maxSumEnding[i] denotes maximum subarray sum ending at index i. If we know the maxSumEnding[i - 1], we can easily calculate the maxSumEnding[i]. Here are some insights:

maxSumEnding[i] always include the value X[i].
If (maxSumEnding[i - 1] > 0), we need to include maxSumEnding[i - 1] to calculate maxSumEnding[i], because maxSumEnding[i - 1] + X[i] will be greater than X[i]. We update, maxSumEnding[i] = maxSumEnding[i - 1] + X[i]
If (maxSumEnding[i -1] < 0), we need to ignore maxSumEnding[i -1] to calculate maxSumEnding[i], because maxSumEnding[i - 1] + X[i] will be less than X[i]. We update, maxSumEnding[i] = X[i]
In more simple way, we can write: maxSumEnding[i] = max (maxSumEnding[i - 1] + X[i], X[i])
Finally, to get the max subarray sum of the whole array, we need to return the maximum value stored in array maxSumEnding[].
This problem falls under dynamic programming because we store the solution of sub-problems and solve the larger problem using the solution of smaller sub-problems.

Solution steps
We create an extra array maxSumEnding[] of size n.
We initialize array with the first value i.e. maxSumEnding[0] = X[0]
Now we traverse array from i = 1 to n - 1. At any ith iteration, we calculate the max subarray sum ending at index i and store it at position maxSumEnding[i].
If (maxSumEnding[i - 1] < 0), maxSumEnding[i] = X[i]
Else, maxSumEnding[i] = X[i] + maxSumEnding[i - 1]
We return the maximum stored in maxSumEnding[].
Solution code C++

```
int findMaxSubarraySum(int X[], int n)
{
   int maxSumEnding[n];
   maxSumEnding[0] = X[0];

   for (int i = 1; i < n; i = i + 1)
```

Regenerate response | Generate PNG | Download PDF | Share Link

```
  }

  int maxSubarraySum = INT_MIN;
  for (int i = 0; i < n; i = i + 1)
    maxSubarraySum = max(maxSubarraySum, maxSumEnding[i]);

  return maxSubarraySum;
}
```

This solution uses dynamic programming to find the maximum subarray sum of an array. It works by creating an extra array `maxSumEnding` of size n, where `maxSumEnding[i]` represents the maximum subarray sum ending at index i. The algorithm then iterates through the array and calculates the maximum subarray sum ending at each index i and stores it in `maxSumEnding[i]`.

If `maxSumEnding[i-1]` is greater than 0, then `maxSumEnding[i]` is calculated as the sum of the current element and the maximum subarray sum ending at the previous index, i.e., `maxSumEnding[i] = X[i] + maxSumEnding[i-1]`. If `maxSumEnding[i-1]` is less than or equal to 0, then the current element is considered as the starting point of a new subarray, i.e., `maxSumEnding[i] = X[i]`.

Finally, the function returns the maximum value stored in `maxSumEnding[]`. Below is the complete code in C++:

C++ Copy code

```cpp
#include <climits>

int findMaxSubarraySum(int X[], int n)
{
    int maxSumEnding[n];
    maxSumEnding[0] = X[0];

    for (int i = 1; i < n; i = i + 1)
    {
        if (maxSumEnding[i - 1] > 0)
            maxSumEnding[i] = X[i] + maxSumEnding[i - 1];
        else
            maxSumEnding[i] = X[i];
```

```
    }

    int maxSubarraySum = INT_MIN;
    for (int i = 0; i < n; i = i + 1)
        maxSubarraySum = max(maxSubarraySum, maxSumEnding[i]);

    return maxSubarraySum;
}
```

SH

hi can you show me the full code with inline comments showing algo and debug info

Sure! Here's the complete code in C++ with inline comments explaining the algorithm and adding some debugging information:

C++ Copy code

```
#include <iostream>
#include <climits>

using namespace std;

int findMaxSubarraySum(int X[], int n)
{
    int maxSumEnding[n]; // Declare an array to store the maximum subarray sum ending
    maxSumEnding[0] = X[0]; // Initialize the first element of the array

    for (int i = 1; i < n; i = i + 1)
    {
        if (maxSumEnding[i - 1] > 0) // If the previous maximum subarray sum is greate
            maxSumEnding[i] = X[i] + maxSumEnding[i - 1]; // add the current element t
        else
            maxSumEnding[i] = X[i]; // Otherwise, start a new subarray at the current
    }

    int maxSubarraySum = INT_MIN; // Initialize the maximum subarray sum to the smalle
    for (int i = 0; i < n; i = i + 1)
    {
```

```
            if (maxSumEnding[i] > maxSubarraySum) // Debug info: Print the value of maxSum
            {
                cout << "maxSumEnding[" << i << "] = " << maxSumEnding[i] << ", maxSubarra
            }
            maxSubarraySum = max(maxSubarraySum, maxSumEnding[i]); // Update the maximum s
        }

        return maxSubarraySum; // Return the maximum subarray sum
    }

    int main()
    {
        int X[] = {1, -2, 3, 4, -5, 8};
        int n = sizeof(X) / sizeof(X[0]);
        int maxSubarraySum = findMaxSubarraySum(X, n);
        cout << "The maximum subarray sum is " << maxSubarraySum << endl;
        return 0;
    }
```

The program first declares an array `maxSumEnding` of size `n` to store the maximum subarray sum ending at each index. It initializes the first element of this array to be the first element of the input array `X`.

It then iterates through the input array `X`, starting at index 1, and calculates the maximum subarray sum ending at each index using the formula `maxSumEnding[i] = max(maxSumEnding[i - 1] + X[i], X[i])`. This formula takes into account the fact that a maximum subarray sum ending at index i either includes the element at index i, or starts at some index j less than i and includes all elements between j and i.

After calculating the maximum subarray sum ending at each index, the program iterates through `maxSumEnding` and finds the maximum value. This value is the maximum subarray sum of the input array `X`.

In this code, I've also added some debugging information to print out the values of `maxSumEnding[i]` and `maxSubarraySum` during the loop that finds the maximum subarray sum. This can be useful for understanding how the algorithm works and verifying that it's working correctly.

SH ok explain how this program works with input as we go through the code neatly

ok explain how this program works with input as we go through the code neatly

Sure, let's walk through the code step-by-step for an example input to understand how the program works. Let's consider the input array `X` as:

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

Here's how the program works for this input:

```
int findMaxSubarraySum(int X[], int n)
{
    int maxSumEnding[n];
    maxSumEnding[0] = X[0];   // Initialize maxSumEnding with first element of X (i.e.

    // Traverse array X from index 1 to n-1
    for (int i = 1; i < n; i = i + 1)
    {
        // Calculate max subarray sum ending at index i
        if (maxSumEnding[i - 1] > 0)
            maxSumEnding[i] = X[i] + maxSumEnding[i - 1];
        else
            maxSumEnding[i] = X[i];
    }

    int maxSubarraySum = INT_MIN;

    // Find maximum subarray sum from all subarrays ending at different indices
    for (int i = 0; i < n; i = i + 1)
        maxSubarraySum = max(maxSubarraySum, maxSumEnding[i]);

    return maxSubarraySum;
}
```

1. The first line declares a variable `maxSumEnding` as an array of size `n`, which will

store the maximum subarray sum ending at each index of the input array `X`.

2. Then, the first element of the `maxSumEnding` array is initialized with the first element of `X`, since the maximum subarray sum ending at the first index is just the value at that index. In our example, the first element of `X` is `-2`, so `maxSumEnding[0]` is also set to `-2`.

```css
maxSumEnding[0] = X[0];
```

1. Next, the program loops through the remaining elements of `X` from index `1` to index `n-1`. For each index `i`, it calculates the maximum subarray sum ending at index `i` using the formula described in the solution idea.
   If the maximum subarray sum ending at the previous index `i-1` is greater than 0, it includes that sum in the calculation of the maximum subarray sum ending at index `i`. Otherwise, it ignores the maximum subarray sum ending at the previous index.
   For our example input array `X`, the loop proceeds as follows:
   - For `i = 1`, since `maxSumEnding[i-1]` is `-2` and less than 0, the maximum subarray sum ending at index `i` is just the value at index `i` itself, which is `1`. Therefore, `maxSumEnding[1]` is set to `1`.

     ```css
     if (maxSumEnding[i - 1] > 0)
         maxSumEnding[i] = X[i] + maxSumEnding[i - 1];
     else
         maxSumEnding[i] = X[i];
     ```

   - For `i = 2`, `maxSumEnding[i-1]` is `1` and greater than 0, so the maximum subarray sum ending at index `i` includes both the value at index `i` (which is `-3`) and the maximum subarray sum ending at index `i-1` (which is `1`). Therefore, `maxSumEnding[2]`

SH does our print which are the elements of max sub array

No, the program as it is currently written does not print the elements of the maximum subarray. It only returns the sum of the maximum subarray.

If you also want to print the elements of the maximum subarray, you can modify the program by keeping track of the start and end indices of the maximum subarray. You can do this by adding two additional variables `start` and `end` to keep track of the current subarray being considered as the maximum subarray. Whenever a new maximum subarray is found, you update `start` and `end` to the current start and end indices.

Here's an updated version of the program with inline comments to show the modifications:

c++ Copy code

```
#include <iostream>
#include <climits> // for INT_MIN

using namespace std;

void findMaxSubarray(int X[], int n)
{
    int maxSumEnding[n];
    int start = 0, end = 0;
    int maxSumSoFar = INT_MIN;
    maxSumEnding[0] = X[0];

    for (int i = 1; i < n; i = i + 1)
    {
        // calculate the maximum subarray ending at index i
        if (maxSumEnding[i - 1] > 0)
            maxSumEnding[i] = X[i] + maxSumEnding[i - 1];
        else {
            maxSumEnding[i] = X[i];
            start = i; // update start index for new subarray
        }

        // update maxSumSoFar if a new maximum subarray is found
        if (maxSumEnding[i] > maxSumSoFar) {
            maxSumSoFar = maxSumEnding[i];
            end = i; // update end index for new subarray
```

```
        }
    }

    // print the maximum subarray and its sum
    cout << "Maximum subarray: [";
    for (int i = start; i <= end; i = i + 1) {
        cout << X[i];
        if (i != end)
            cout << ", ";
    }
    cout << "]\n";
    cout << "Maximum subarray sum: " << maxSumSoFar << endl;
}

int main()
{
    int X[] = { -2, 1, -3, 4, -1, 2, 1, -5, 4 };
    int n = sizeof(X) / sizeof(X[0]);
    findMaxSubarray(X, n);
    return 0;
}
```

When you run this program with the input `X = { -2, 1, -3, 4, -1, 2, 1, -5, 4 }`, it will output:

yaml Copy code

```
Maximum subarray: [4, -1, 2, 1]
Maximum subarray sum: 6
```