Week 4 – Take Home Assignment

- Finding the maximum sub array sum of a given number array (using left half and right half divide and conquer approach)

```cpp
// Include necessary libraries for input/output and handling vectors
#include <iostream>
#include <sstream>
#include <vector>
#include <limits>

// Using the standard namespace for cout and endl
using namespace std;
```

```cpp
// Function to compute the maximum subarray sum that crosses the midpoint
int countSum(vector<int> &numVect, int start, int mid, int end){
    // Initialize variables to store the sum of the left and right subarrays
    int sum = 0;
    int leftsum = std::numeric_limits<int>::min(); // set initial leftsum to the minimum value of int

    // Loop through the elements of the left subarray (from mid to start)
    for (int i= mid; i >= start; i--){
        sum = sum + numVect[i]; // add current element to the sum
        if (sum > leftsum){
            leftsum = sum;
        }
    }

    // Reset the sum variable for computing the sum of the right subarray
    sum = 0;
    int rightSum = std::numeric_limits<int>::min(); // set initial rightSum to the minimum value of int

    // Loop through the elements of the right subarray (from mid+1 to end)
    for (int i= mid+1; i <= end; i++){
        sum = sum + numVect[i]; // add current element to the sum
        if (sum > rightSum) {
            rightSum = sum;
        }
    }
// Function to compute the maximum subarray sum using divide and conquer approach
int maxSum(vector<int> &numVect, int start, int end){
    // Base case: if start index is greater than end index, return minimum int value
    if (start > end) {
        return std::numeric_limits<int>::min();
    }
    // Another base case: if start index equals end index, return the element at that index
    if (start == end){
        return numVect[end];
    }

    // Calculate the midpoint of the array
    int mid = (start + end)/2;

    // Recursively compute the maximum subarray sum of the left and right halves
    int leftSum = maxSum(numVect, start, mid);
    int rightSum = maxSum(numVect, mid + 1, end);

    // Compute the maximum subarray sum that crosses the midpoint
    int crossSum = countSum(numVect, start, mid, end);

    // Return the maximum of the three computed sums
    return max(max(leftSum, rightSum), crossSum);
}
```

```cpp
// Main function to read input, call maxSum function, and output the result
int main(){
    // Define a vector to store input numbers
    vector<int> numberVector;
    int k = 0;

    // Read a line of input containing space-separated numbers
    string input;
    getline(cin, input);

    // Create a string stream to extract numbers from the input line
    istringstream stream(input);
    int number;

    // Extract numbers from the string stream and add them to the vector
    while (stream >> number) {
        numberVector.push_back(number);
        k++;
    }

    // Call the maxSum function to compute the maximum subarray sum
    int output = maxSum(numberVector, 0, k-1);

    // Output the maximum subarray sum
    cout << output << endl;
}
```

- **Recurrence relation** ==> T(n) = 2 T (n / 2) + n

2) substitution method

our guese is $O(n) = n \log n$

$T(n) = 2 (cn \log n) + n$

$0 < n$ and $0 < \frac{n}{2} < n$

$T(1) = 2(c \log \frac{1}{2}) + 1 = 2c(\log 1 - \log 2) + 1$

$= (1 - 2c)$

$< 1 \quad c > \frac{1}{2}$

< 0

$= c_1 \log_1$

$T(n/2) = 2(c \frac{n}{2} \log n/2) + n/2$

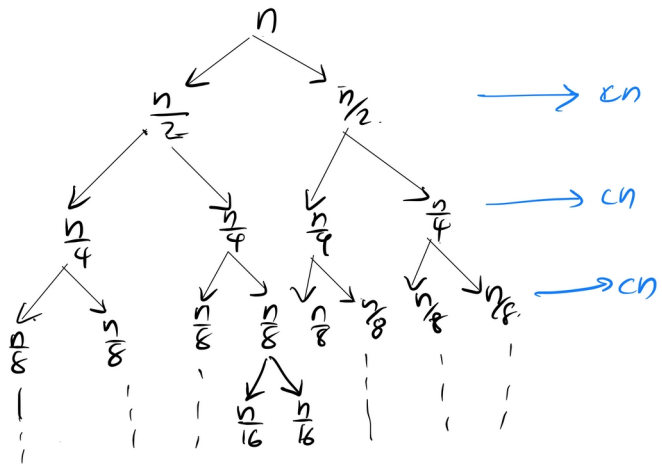
$= nc \log(n/2) + n/2 = nc \log n - cn \log 2 + n$

$= cn \log(n) + (1-c)n$

$\leq cn \log n \quad c \geq 1$

Tree method

$T(n) = 2T(n/2) + n$



Time ⇒ $n=4 \rightarrow 2cn$
$n=8 \rightarrow 3cn$
$n=16 \rightarrow 4cn$
$n=32 \rightarrow 5cn$
} nlogn.

Master method.

$T(n) = 2T(n/2) + n.$

$a=2 \quad b=2 \quad d=1$

$a \log b = 2 \log 2 = 2 > d \quad \therefore$ Case I

$T(n) = \Theta(n \log n)$