Maximum subarray sum

Approach 1:

Intuition

The first intuition is to generate all subarrays and then compute sum of all subarrays.

Approach

Iterate i ->0 to n and j -> i to n and then k -> i(start) to j (end) to generate all subarray and compute sum inside k loop.

Complexity

- Time complexity:

```
O(n^3)
```

- Space complexity:

```
O(1)
```

Code

```
class Solution {
   public:
       int maxSubArray(vector<int>& nums) {
           int sum=0,maxSum=INT_MIN;
       for(int i=0;i<nums.size();i++)
       {

           for(int j=i;j<nums.size();j++)
           {
```

```
                sum=0;
                for(int k=i;k<=j;k++)
                {
                    sum+=nums[k];
                }
                maxSum=max(maxSum,sum);
            }
        }
        return maxSum;
        }
    };
```

Approach 2:

Intuition

The intuition is to maintain to pointers to mark the start and end and compute sum of numbers in the range.

Approach

Iterate i ->0 to n and j -> i to n and then k -> i(start) to j (end) to generate all subarray and compute sum inside k loop.

Complexity

- Time complexity:

```
O(n^2)
```

- Space complexity:

```
O(1)
```

Code

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int sum=0,maxSum=INT_MIN;
     for(int i=0;i<nums.size();i++)
     {
         sum=0;
         for(int j=i;j<nums.size();j++)
         {
             sum+=nums[j];
             maxSum=max(maxSum,sum);
         }
     }
     return maxSum;
    }
};
```

Approach 3:

Intuition

Using Kadane's Algorithm.

Approach

Run a loop from i -> 1 to n. Our motivate is to ignore sum if negative and start with a new subarray. Keep on adding values to sum variable maintain a maximum variable to store the maximum seen so far. If sum exceeds max then store sum in max. If sum is negative then set sum to 0 because we do not need negative sum and the sum needs to be continous as subarray is specified in the question so for starting the sum for a new subarray we need to set the sum to 0.

Complexity

- Time complexity:

```
O(n)
```

- Space complexity:

```
O(1)
```

Code

```
#include <bits/stdc++.h>
long long maxSubarraySum(int arr[], int n)
{
     long long sum=0,max=INT_MIN;
     for(int i=0;i<n;i++)
     {
         sum+=arr[i];
         if(sum>max)
            max=sum;
        if(sum<0)
            sum=0;
     }
     if(max<0)
        return 0;
     return max;
}
```