Kadane's Algorithm

Tags	algorithm dynamic_programming
Companies	Adobe Amazon Facebook/Meta Goldman Sachs Google Hike Linkedin MetLife Microsoft Ola Cabs Oracle Paypal Samsung Snapdeal Teradata Visa Walmart Yahoo flipkart
Created time	@October 14, 2022 7:10 AM
Last edited time	@October 19, 2022 6:10 PM
URL	

Task :-

Given an integer array we have to find a contiguous sub-array with maximum sum.

```
example-
Input-
{-2, 1,-3, 4,-1, 2, 1,-5, 4}
output -
6
explanation -
{-2, 1,-3, 4,-1, 2, 1,-5, 4}
[ 0  1  2 |3  4  5  6| 7  8]
In this case it is from index 3 to 6 {4, -1, 2, 1} i.e. 6.
```

Solution:-

Brute force solution - firstly we will look at a obvious, accurate but inefficient approach which will be iterating whole array with nested loops and getting all possible contiguous sub-array summing them and returning max of that sum values.

all possible contiguous sub-array

```
{-2}, = -2
{-2,1} = -1
{-2,1,-3} = -4
{-2,1,-3,4} = 0
{-2,1,-3,4,-1} = -1
{-2,1,-3,4,-1,2} = 1
{-2,1,-3,4,-1,2,1} = 2
{-2,1,-3,4,-1,2,1,-5} = -3
{-2,1,-3,4,-1,2,1,-5,4} = 1
```

```
{1} = 1
{1,-3} = -2
{1,-3,4} = 2
{1,-3,4,-1} = 1
{1,-3,4,-1,2} = 3
{1,-3,4,-1,2,1} = 4
{1,-3,4,-1,2,1,-5} = -1
{1,-3,4,-1,2,1,-5,4} = 3
```

```
{-3} = -3
{-3,4} = 1
{-3,4,-1} = 0
{-3,4,-1,2} = 2
{-3,4,-1,2,1} = 3
{3,4,-1,2,1,-5} = -2
{-3,4,-1,2,1,-5,4} = 2
```

```
{4} = 4
{4,-1} = 3
{4,-1,2} = 5
{4,-1,2,1} = 6 // answer
{4,-1,2,1,-5} = 1
{4,-1,2,1,-5,4} = 5
```

```
{-1} = -1
{-1,2} = 1
{-1,2,1} = 2
{-1,2,1,-5} = -3
{-1,2,1,-5,4} = 1
```

```
{2} = 2
{2,1} = 3
{2,1,-5} = -2
{2,1,-5,4} = 2
```

```
{1} = 1
{1,-5} = -4
{1,-5,4} = 0
```

```
{-5} = -5
{-5,4} = -1
```

```
{4} = 4
```

and we got the answer. i.e. 6

This method will give as O(n^2) time complexity which is not necessarily a bad solution but as the input will grow it can be an expensive task,

but it can be reduced we can notice in the brute method many times we are iterating and array in inner loop even if that sub array can no longer can possibly give answer.

for example when we are adding a contiguous array if any element turns whole sum to negative it doesn't matter how many element we add in front of that sub-array it will not give the answer.

like in first inner loop while we were adding elements we got a negative sum in first element it self i.e.

```
{-2}, = -2
```

now if we add next element we will get {-2,1} = -1 and if set sum to 0 then add we will get sum as 1

we will get a better answer if we eliminate current element and all element before that (in this case -2 is first element)

Lets take an another example in same question with bigger sub-array.

{-2,1,-3,4,-1} = -1

now when we added -1 in this sub-array we got a negative sum now even though the sum was 0 ({-2,1,-3,4} = 0) before that it still could be a possible answer but once the sum became negative all sum of that inner-loop sub-array will have a negative effect like

with {-2,1,-3,4,-1} adding next will be {-2,1,-3,4,-1,2} = 1 without it, it will be {2} = 2 which is clearly more that {-2,1,-3,4,-1,2} = 1 still if we add further more element hoping it might get better we will get {-2,1,-3,4,-1,2,1} = 2 and without sub-array{-2,1,-3,4,-1} sum will be {2,1} = 3

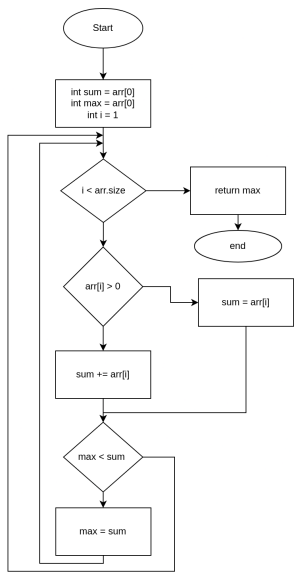
in fact every element will have more potential to give answer if we restart sum from 0 whenever the sum goes negative and because we want a contiguous array we cannot add element before that.

so that conclude to argument that we are sometimes iterating some sub-array even it is useless further, so we have to find a way to ignore that calculations and optimise our solution to make our code more efficient.

lucky for us we don't have to reinvent the wheel, we already have a well described and well tested algorithm i.e. **Kadane's algorithm**

In this algorithm we will start by making two variables currentSum and maxSum) and initialise both with the 0th element of array after that we will iterate the array from 1 to size-1 as we iterate we will add current element to sum then check if sum is bigger than 0 (non-negative) if sum becomes negative we will set sum to zero after all of that we will check if current sum is bigger than max-sum so we will always have max sum till end of loop.

```
int maxSubArray(vector<int>& nums) {
  int currentSum = nums[0];
  int maxSum = nums[0];
  for(int i = 1;i < nums.size();i++)
  {
    if(currentSum < 0)
    {
      currentSum = nums[i];
    }
    else
    {
      currentSum += nums[i];
    }
    maxSum = max(maxSum,currentSum); //updating max
  }
  return maxSum;
}
```



dry run with same input

```
{-2, 1,-3, 4,-1, 2, 1,-5, 4}
i = 0 sum = -2 max = -2 || sum = {-2, 1,-3, 4,-1, 2, 1,-5, 4} max = {-2, 1,-3, 4,-1, 2, 1,-5, 4}
i = 1 sum =  1 max =  1 || sum = {-2, 1,-3, 4,-1, 2, 1,-5, 4} max = {-2, 1,-3, 4,-1, 2, 1,-5, 4}
i = 2 sum = -2 max =  1 || sum = {-2, 1,-3, 4,-1, 2, 1,-5, 4} max = {-2, 1,-3, 4,-1, 2, 1,-5, 4}
i = 3 sum =  4 max =  4 || sum = {-2, 1,-3, 4,-1, 2, 1,-5, 4} max = {-2, 1,-3, 4,-1, 2, 1,-5, 4}
i = 4 sum =  3 max =  4 || sum = {-2, 1,-3, 4,-1, 2, 1,-5, 4} max = {-2, 1,-3, 4,-1, 2, 1,-5, 4}
i = 5 sum =  5 max =  5 || sum = {-2, 1,-3, 4,-1, 2, 1,-5, 4} max = {-2, 1,-3, 4,-1, 2, 1,-5, 4}
i = 6 sum =  6 max =  6 || sum = {-2, 1,-3, 4,-1, 2, 1,-5, 4} max = {-2, 1,-3, 4,-1, 2, 1,-5, 4}
i = 7 sum =  1 max =  6 || sum = {-2, 1,-3, 4,-1, 2, 1,-5, 4} max = {-2, 1,-3, 4,-1, 2, 1,-5, 4}
i = 8 sum =  5 max =  6 || sum = {-2, 1,-3, 4,-1, 2, 1,-5, 4} max = {-2, 1,-3, 4,-1, 2, 1,-5, 4}
```