Maximum Subarray Sum

I just wanted to quickly go over my solution, and the others, for the Maximum Subarray Sum problem that I found on the UASCO Guid website. It's not the cleanest but it is O(n).

The problem: https://cses.fi/problemset/task/1643

Given an array of n integers, your task is to find the maximum sum of values in a contiguous, nonempty subarray. For instance {-1, 2, 3} is 5 as {2, 3} is the max subarray

Naive O(n^2)

For loop through the array and For each index loop through the sum of every possible subarray sum where that index is the leftmost element.

example: this code show the algorithm run on array {-1, 2, 3}

```
int Arr[] = {-1, 2, 3}
int n = length

int M = arr[0]; // max substring sum so far
int sum = 0; // current sum of whatever index we are on

for (int i = 0; i < n; i++) {
 sum = 0;
 for (int j = i; j < n; j++) {
  sum += arr[j];
  if (sum > M) {
   M = sum;
  }
 }
}
```

Now let's go over what the example is doing.

It starts by looping over each ith element in the array. For each ith element it calculates all possible subarrays starting at i.

As we loop through each of the possible sub arrays, we keep track of the highest sum that we have seen so far. This is done with the variable M. everytime we encounter a subarray that has a sum greater than M, that sum will be our new

M. This way at the end of the loops, M will contain the sum of the subarray with the largest sum, which is the answer.

Sadly the amount of loops we do is dictated by $\sum_{i=0}^{n} \sum_{j=i}^{n} 1$where *n* is the size of the array. When solved this comes out as on the order of n^2so this solution is a sluggish O(n^2). Can we do better?

O(n) - My solution using prefix sums

What is a prefix sum?

We can define a prefix sum as $prefix[i] = \sum_{n=0}^{i} arr[n]$ and thus $prefix[a] - prefix[b] = \sum_{n=b}^{a} arr[n]$. This means with a prefix sum we can in constant time calculate the sum between any two indices in arr[]. (learn prefix sums - https://usaco.guide/silver/prefix-sums?lang=cpp)

The code

We want to get the sum between two indices of the array so naturally we want to use a prefix sum. Through these sums any subarray sum starting at i ending at j (j $\geq$ i) can be calculated as prefix[j] - prefix[i]. That means to find the max subarray sum we need to maximize prefix[j] - prefix[i] and to do so we know that prefix[j] will be a local max and prefix[i] will be the lowest value before j.

After this the rest is easy we will loop through the prefix array and keep track of the current min value at that point. On each element we will minus the current element form that min to find the max subarray ending at that element. We go through every element in the list and find the one that has the biggest difference, which would tem be the biggest subarray sum.

```
// n is size
// arr[] is the array

// get prefix sum
int prefix[n];
prefix[0] = arr[0]
for (int i = 1; i<n; i++) {
  prefix[i] = arr[i] + prefix[i-1];
}

// finds max sum
int maxdiff = arr[0];
int minval = arr[0];
for(int i = 0; i < n; i++) {
  int cur = arr[i];
  // if cur and min have the largest diff update
  if (cur - minval > maxdiff) {
    maxdiff = cur-minval;
  }

  // if cur is less then minval make it the new min
  if (cur < minval) {
    minval = cur;
  }
}


cout << maxdiff << endl;
```

O(n) The normal solution - Kadane's algorithm

```
// arr[] is the array, n is the length

int maxsum[n]; // keeptrack of max subarray sum at each index
int maxval = arr[0]; // keep track of the overall max
for (int i = 1; i < n; i++) {

  // look at the options for max sum
  maxsum[i] = max(arr[i], arr[i] + maxsum[i-1]);

  // update overall max if this is new max
  if (maxsum[i] > maxval) {
    maxval = maxsum[i];
  }
}
cout << maxval << endl;
```

In this algorithm one checks for the max subarray sum for each index of i, but can reuse work done from previous indexes. How does this work? Let's define arr[] as the array and maxsum[] as the max subarray sum ending at each index. Say I want to find the max subarray sum at i. arr[i] on its own might be the max, but it could also be arr[i] + arr[i-1] + arr[i-2] ... Well in this case arr[i] + maxsum[i-1] would be the solution.

any arr[i] + arr[i-1] + arr[i-2] … can be thought of as arr[i] + (some subarray sum ending at i-1), let's call that sum *x*.

Since

$$\textbf{maxsum[i-1]} \geq \textbf{\textit{x}},$$

then

$$\textbf{arr[i] + maxsum[i-1]} \geq \textbf{arr[i] + x}$$

So for any i, maxsum[i] is either arr[i] or arr[i] + maxsum[i-1]