

MiniAssignment2

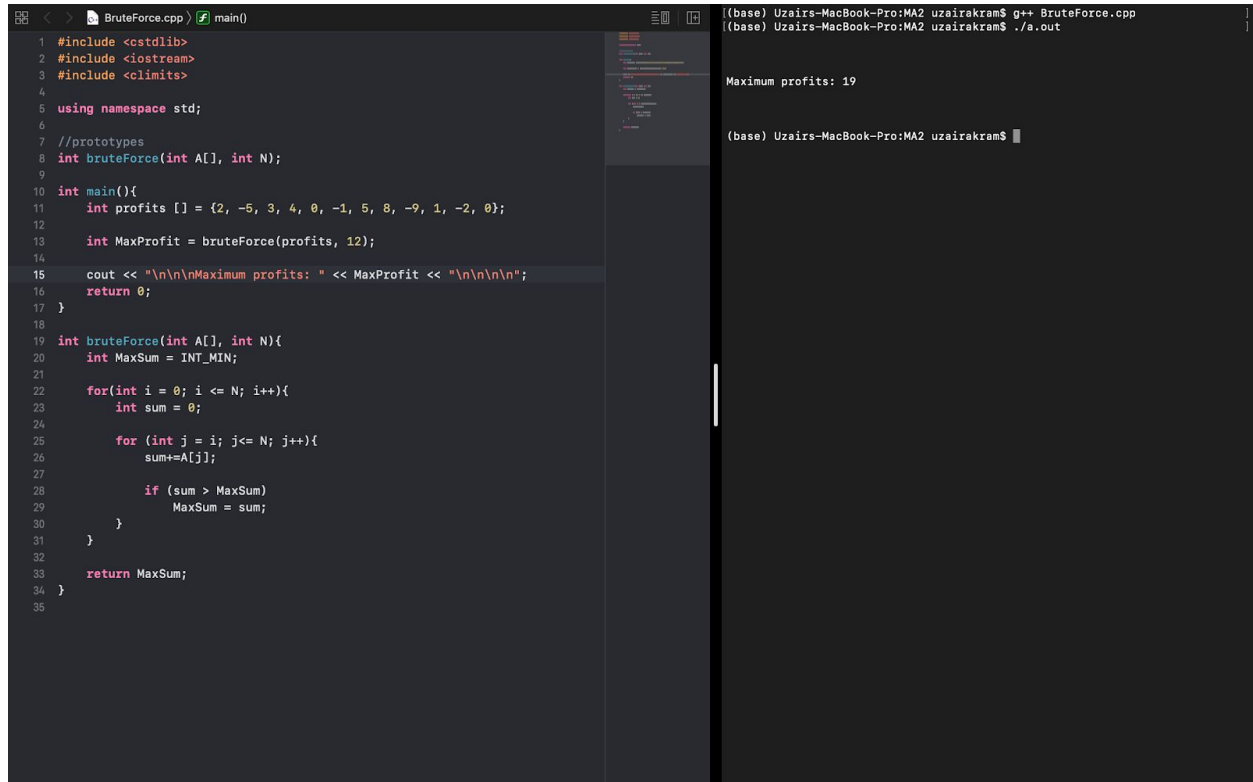
Brute Force

There are many solutions to the problem one is the brute force method by checking all possible subarrays. We can determine the subarray that produces maximum sum of its elements.

```
1 //Uzair Akram - Pseudo code
2 MaxSubarray(A[]){
3     int MaxSum = 0
4
5     for(int i = 0; i < A.size; i++){
6         int sum = 0
7
8         for (int j = i; j < A.size; j++){
9             sum+=A[j]
10
11             if sum > MaxSum
12                 MaxSum = sum
13         }
14     }
15
16     return MaxSum
17 }
18
```

The program would loop through each element starting at index zero and for each element it would sum its sub arrays and checking for a maximum sum between all sums of the subarrays of contiguous elements. The Algorithm has a time complexity of $O(N^2)$.

Implementation - BruteForce



```
1 #include <cstdlib>
2 #include <iostream>
3 #include <climits>
4
5 using namespace std;
6
7 //prototypes
8 int bruteForce(int A[], int N);
9
10 int main(){
11     int profits [] = {2, -5, 3, 4, 0, -1, 5, 8, -9, 1, -2, 0};
12
13     int MaxProfit = bruteForce(profits, 12);
14
15     cout << "\n\nMaximum profits: " << MaxProfit << "\n\n\n";
16     return 0;
17 }
18
19 int bruteForce(int A[], int N){
20     int MaxSum = INT_MIN;
21
22     for(int i = 0; i <= N; i++){
23         int sum = 0;
24
25         for (int j = i; j<= N; j++){
26             sum+=A[j];
27
28             if (sum > MaxSum)
29                 MaxSum = sum;
30         }
31     }
32
33     return MaxSum;
34 }
35
```

```
((base) Uzairs-MacBook-Pro:MA2 uzairakram$ g++ BruteForce.cpp
((base) Uzairs-MacBook-Pro:MA2 uzairakram$ ./a.out

Maximum profits: 19

((base) Uzairs-MacBook-Pro:MA2 uzairakram$
```

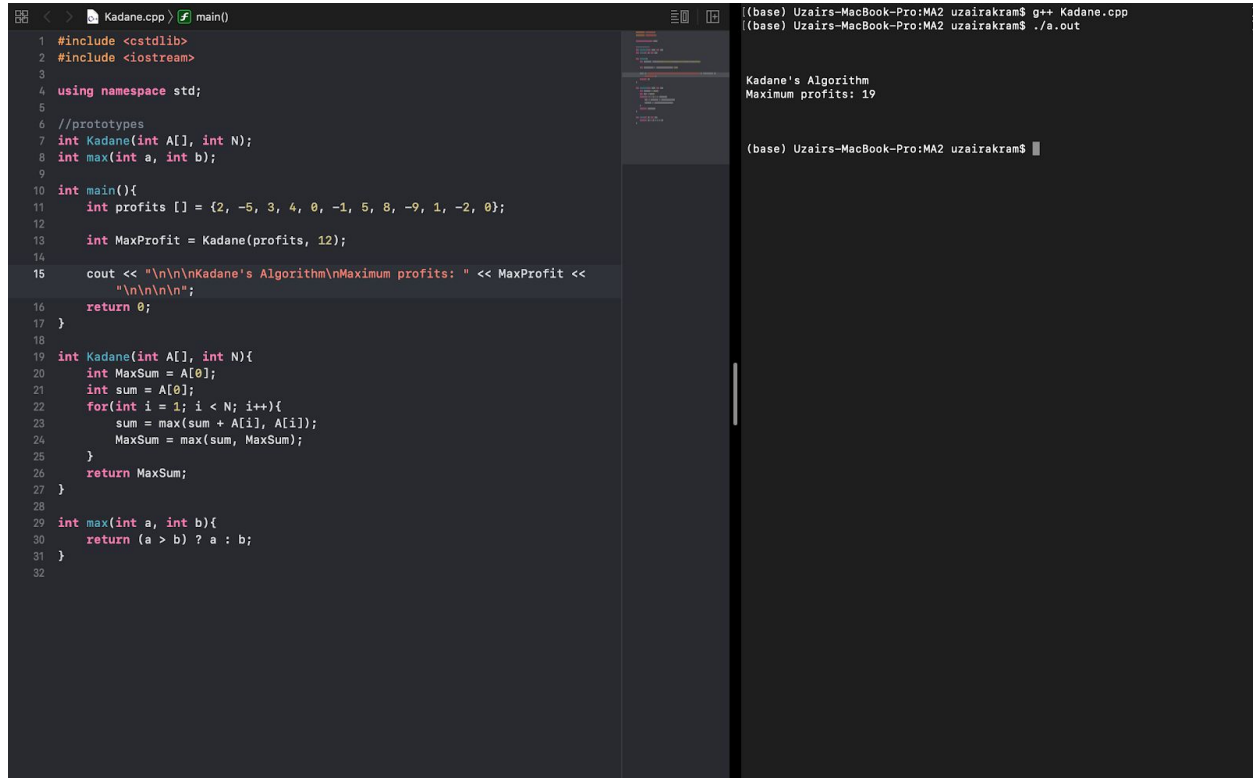
Produces the correct result. Output in terminal.

The problem can be solved using Kadane's Algorithm in $O(N)$ time. The Algorithm finds the maximum subarray of contiguous elements.

```
1 //Uzair Akram – Pseudo Code
2 Kadane(A[])
3     int MaxSum = sum = A[0]
4     for (int i = 1; i < A.size; i++)
5         sum = max(sum + A[i], A[i])
6         MaxSum = max(sum, MaxSum)
7     return MaxSum
8
```

Kadane's algorithm compares the previous sum plus the current element to the value of the current element and updates to the greater of the two. The sum ensures a sum of contiguous elements and maintains continuity by considering only two cases where the element is added to the array so the element added simply becomes an element of the subarray or the case when the element at index is greater than the element plus previous sum of contiguous elements which mean that the subarray would start at that element and begin a new subarray for producing greater sum. The sum is then compared with the previous max_sum and updated to the greater of the two. The result is the sum of the subarray that has the largest sum.

Implementation - Kadane



```
1 #include <cstdlib>
2 #include <iostream>
3
4 using namespace std;
5
6 //prototypes
7 int Kadane(int A[], int N);
8 int max(int a, int b);
9
10 int main(){
11     int profits [] = {2, -5, 3, 4, 0, -1, 5, 8, -9, 1, -2, 0};
12
13     int MaxProfit = Kadane(profits, 12);
14
15     cout << "\n\nKadane's Algorithm\nMaximum profits: " << MaxProfit <<
16         "\n\n\n";
17     return 0;
18 }
19
20 int Kadane(int A[], int N){
21     int MaxSum = A[0];
22     int sum = A[0];
23     for(int i = 1; i < N; i++){
24         sum = max(sum + A[i], A[i]);
25         MaxSum = max(sum, MaxSum);
26     }
27     return MaxSum;
28 }
29
30 int max(int a, int b){
31     return (a > b) ? a : b;
32 }
```

```
((base) Uzairs-MacBook-Pro:MA2 uzairakram$ g++ Kadane.cpp
((base) Uzairs-MacBook-Pro:MA2 uzairakram$ ./a.out

Kadane's Algorithm
Maximum profits: 19

((base) Uzairs-MacBook-Pro:MA2 uzairakram$
```