HomeWork 04

Rohan Jawed
21L-5625

(Q1)

```
MaxSubArrSum (int A[], int n){
        globalEnd = 0
        globalStart = 0
        global Sum=A[0]
        MaxSum = A[0]

    for(i=0; i<n-1; i++){
        if(MaxSum + A[i]> A[j])
            MaxSum += A[i]    }
        else {
            Max Sum = A[i]
            global Start = i    }
        if( globalSum <Max Sum)
            globalSum= Max Sum
            globalend = i

    return global Sum
}
```

(Q2)

Dry Run for Brute-force

2	-4	3	4	-3	5	-5	6	-1
1	2	3	4	5	6	7	8	9

i	j	subArrSum	globalSum
1	1	2	2
2	2	-4	2
2	1	-2	2
3	3	3	3
3	2	-1	3
3	1	1	3
4	4	4	4
4	3	7	7
4	2	3	7
4	1	5	7
5	5	-3	7
5	4	1	7
5	3	4	7
5	2	0	7
5	1	2	7
6	6	5	7
6	5	2	7
6	4	6	7
6	3	9	9
6	2	5	9
6	1	7	9

7	7	-5	9
7	6	0	9
7	5	-3	9
7	4	1	9
7	3	4	9
7	2	0	9
7	1	2	9
8	8	6	9
8	7	1	9
8	6	6	9
8	5	3	9
8	4	7	10
8	3	10	10
8	2	6	10
8	1	8	10
9	9	-1	10
9	8	5	10
9	7	0	10
9	6	5	10
9	5	2	10
9	4	6	10
9	3	9	10
9	2	5	10
9	1	7	10

maxSum is <u>10</u> in both cases

(Q3) <u>Kadane's Algo</u>

i	maxSum[i]	globalSum	globalEnd
2	-2	2	2
3	3	3	3
4	7	7	4
5	4	7	4
6	9	9	6
7	4	9	6
8	10	10	8
9	9	10	8

—(Q4)—

Time Complexity = O(1)

```
globalSum = A[1]
presentSum = A[1]

for (i=2 ; i<N ; i++)
    if (presentSum + A[i] > A[i])
      presentSum += A[i]
    else
      presentSum = A[i]
      globalSum = max(globalSum,
                        presentSum)

    return globalSum
```

It uses a single array so its time complexity will be O(1).