Group Project 3
Dylan Camus (931-769-466), Mihai Dan (932-146-380), Braden Ackles (932-115-298)
Analysis of Algorithms

Task 1.

Method 1:

```
Set A[] and B[] as the two halves of the array
Set tempA[] equal to the sum of the suffices of A[]
Set tempB[] equal to the sum of the prefixes of B[]
Declare a currentMin integer variable
Declare a tempMin integer variable
For i = 0 to the length of tempA[]
        For j = 0 to the length of tempB[]
                tempMin = | tempA[i] – tempB[j] |
                if tempMin is less than currentMin
                        currentMin = tempMin
                        lowIndex = i
                        highIndex = j
Return [lowIndex, highIndex, currentMin]
```

Runtime for this algorithm is $O(n^2)$.

Method 2:

```
Set A[] and B[] as the two halves of the array
Set tempA[] equal to the sum of the suffices of A[]
Set tempB[] equal to the sum of the prefixes of B[]
Sort tempA[] and tempB[]
Declare a currentMin integer variable
Declare a tempMin integer variable
For i = 0 to the length of tempA[]
        For j = 0 to the length of tempB[]
                tempMin = | tempA[i] – tempB[j] |
                if tempMin is less than currentMin
                        currentMin = tempMin
                        lowIndex = i
                        highIndex = j
                else if currentMin is less than tempMin
                        break
Return [lowIndex, highIndex, currentMin]
```

Runtime for this algorithm is $O(n^2)$.

<u>Method 3:</u>

Set A[] and B[] as the two halves of the array
Set tempA[] equal to the sum of the suffices of A[]
Set tempB[] equal to the sum of the prefixes of B[]
Declare a currentMin integer variable
Declare a tempMin integer variable
Negate values that pertain to tempB[]
MergeSort tempA[] and tempB[] into C[][]
For i = 0 to the length of C[][]
    If i and i+1 do not belong to the same array
        tempMin = C[i][] – C[i+1][]
        if tempMin is less than currentMin
            currentMin = tempMin
            lowIndex = the index that corresponds in tempA[]/tempB[] from C[][]
            highIndex = the index that corresponds in tempA[]/tempB[] from C[][]
Return [lowIndex, highIndex, currentMin]

Runtime for this algorithm is O(nlogn).

Task 2.

closestToZero(A[]){
    if the input size is 1
        Return {A[0], 0, 0}
    Divide A[] equally into A1[] and A2[]
    Return [minimum of closestToZero(A1[]), closestToZero(A2[]), suffixPrefix(A[])]

Task 3.

Method 1:

$$T(n) = 2T\left(\frac{n}{2}\right) + n^2$$

By the Master theorem, a = 2 b = 2 d = 2

$$log_2 2 < 2$$

$$\text{Therefore, } T(n) = \theta(n^2)$$

Method 2:

$$T(n) = 2T\left(\frac{n}{2}\right) + n^2$$

By the Master theorem, a = 2 b = 2 d = 2

$$log_2 2 < 2$$

$$\text{Therefore, } T(n) = O(n^2)$$

For this algorithm, best case runtime would be O(n), assuming the loop breaks out after the first iteration. The worst case runtime would be $O(n^2)$, as shown above.

Method 3:

$$T(n) = 2T\left(\frac{n}{2}\right) + nlog(n)$$

$$= 4T\left(\frac{n}{4}\right) + nlog(n) + nlog\left(\frac{n}{2}\right)$$

$$= 2^k T\left(\frac{n}{2^k}\right) + n \times \sum_{i=0}^{k-1} \log\left(\frac{n}{2^i}\right)$$

$$= 2^k T\left(\frac{n}{2^k}\right) + n \times \sum_{i=0}^{k-1} \log(n - i)$$

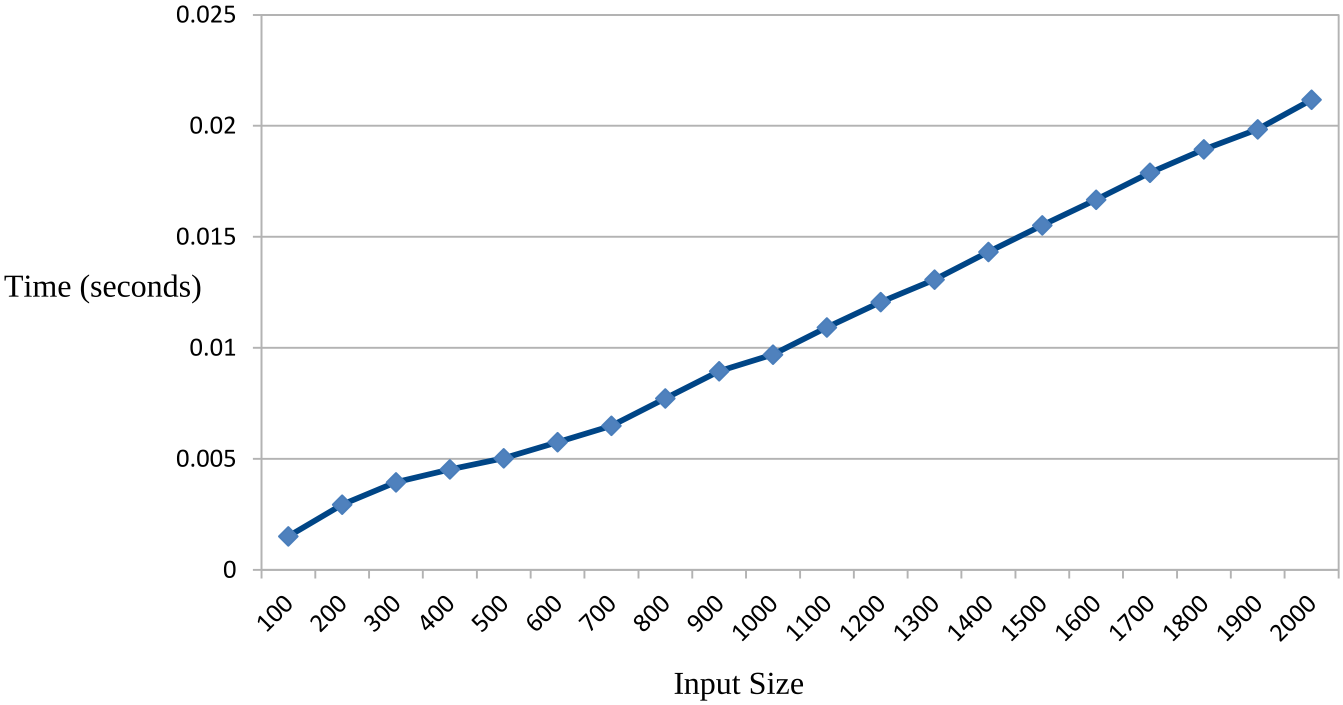
$$\text{Because we know that } \frac{n}{2^k} = 1 \text{ and } k = \log(n)\text{, we can simplify to}$$

$$= n + n \times \sum_{i=0}^{k-1} \mathrm{k} - \mathrm{i}$$

$$= O\left(n \frac{\log(n)\log(n+1)}{2}\right)$$

$$\text{Which gives us a runtime of } O\left(nlog^2(n)\right)$$

Task 6.


Time (seconds)
0.025
0.02
0.015
0.01
0.005
0
100
200
300
400
500
600
700
800
900
1000
1100
1200
1300
1400
1500
1600
1700
1800
1900
2000
Input Size

Extra Credit:

There is an algorithm the runs better than these. It starts by sorting the array which can be done in O(nlog(n)). After that we have a few cases.

1. Case 1: they are all positive numbers. We know the sub-array closest to 0 will be the sub-array of the first two values because they are the smallest due to the sort.
2. Case 2: just like if they are all positive if they are all negative they are the last two in the array because those will be the smallest number.
3. Case 3: there are both positives and negatives:
    a. You evaluate every combination of negative numbers that can be a sub string (for example, -5 -1) so in this case you'd have (-5, -2), (-2)
    b. From there, for each of those, you evaluate each possible sub-array from left to right
        i. So if you're array was -5, -1, 2 , 7, 12, 18

-5-2 = -7	-5-2+1 = -6	-5-2+1 + 7 = 1	-5-2+1+7+12=13	-5-2+1+7+12+18=31
-2+1=-1	-2+1+7=6	-2+1+7+12=13	-2+1+7+12+18=-36	

        ii. Making sure to keep track of the smallest number obtained as well as the length of the sub string to obtain it for example in case of a tie
        iii. You can stop iterating through once you have reached a number to start your new sub-array with that is greater than the number achieved. Because you won't be able to get a number smaller than that if you add anything to it.
        iv. So in this case we have a tie 1 and -1 are equally close to zero but since we track the number of items in the sub away -1 wins because its sub-array is 2 and 1 has a sub-array size of 4.
4. Iterating through the array will only cost O(n), so the most costly part of this will be the sorting. The worst case scenario this algorithm will run in O(nlog(n)). So this algorithm will run faster than the other algorithms.