Philippe WS Lab 1

A1.)

```
Int maxSoFar = 0;
for( low = 1 to n ){
        int currentSum = 0;
        for( high= low to n ){
                currentSum += array[high];
                if( currentSum > maxSoFar )
                        maxSoFar = currentSum;
        }
}
```

A2.)

Low	# of Additions
1	(n-low+1) = n
2	(n-low+1) = n -1
…	
n	(n-low+1) = 1

Observing the number of additions shows a summation of iterative integers.

$$\sum_{low=1}^{n} (n - low + 1) = \frac{n(n+1)}{2}$$

The result is a degree two polynomial. Thus the algorithm has a $\theta(n^2)$ complexity.

B1.)

```
greestSequenceSum(array, sIndex, eIndex){          //startIndex & endIndex
        if(sIndex > eIndex) { return array[sIndex]; }
        int rMax, lMax, rSubMax, lSubMax, combinedMax, sum = 0;
        int mid = (sIndex+eIndex)/2;

        for( i = mid+1 to eIndex ){
                sum += array[ i ];
                if( sum > rMax) { rMax = sum; }
        }
        sum = 0;
        for( i = mid to sIndex ){
                sum += array[ i ];
                if( sum > lMax ) { lMax = sum; }
        }

        rSubMax = greatestSequenceSum(array, mid+1, eIndex);
        lSubMax = greatestSequenceSum(array, sIndex, mid);

        if( lSubMax >= rSubMax ) {
                combinedMax = lSubMax; }
        else {
                combinedMax = rSubMax; }

        if( combinedMax >= (lMax + rMax) {
                return combinedMax; }
        else {
                return (lMax + rMax); }
```

B2.)

The recurrence relation is similar to merge sort, but with a different base case.

$$M(n) = 2\left(M\left(\frac{n}{2}\right)\right) + n\,, \qquad M(1) = 0$$

B3.)

Input Size: n

Basic Operation: Addition

Cases: Best, Worst, and Average case have the same complexity and number of additions.

Derivation by back substitution.

$M(n) = 2\left(M\left(\frac{n}{2}\right)\right) + n$
$M(n) = 2\left(2 * M\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n$
$M(n) = 2^2\left(M\left(\frac{n}{4}\right)\right) + 2n$
$M(n) = 2^2\left(2 * M\left(\frac{n}{8}\right) + \frac{n}{4}\right) + n$
$M(n) = 2^3\left(M\left(\frac{n}{8}\right)\right) + 3n$
$\dots$
$M(n) = 2^k\left(M\left(\frac{n}{2^k}\right)\right) + kn$

To get to the base case, determine k for the base case of $M(1) = 0$

$\frac{n}{2^k} = 1$	$n = 2^k$	$\log(n) = k$

Simplify for the base case.

$$2^{\log(n)} * M(1) + \log(n) * n$$

Substitute $M(1) = 0$, thus this algorithm has $\theta(n * \log(n))$ complexity.