**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.

The result is a degree two polynomial. Thus the algorithm has a complexity.

B1.)

greatestSequenceSum(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.

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.

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
| … |
|  |

To get to the base case, determine *k* for the base case of

|  |  |  |
| --- | --- | --- |
|  |  |  |

Simplify for the base case.

Substitute , thus this algorithm has complexity.