Project 1

Theoretical Run-time Analysis:

**Algorithm 1, Enumeration, Pseudocode:**

MSASumEnum(List [1…n])

currentMax = -∞

begin = -1, end = -1

FOR i from 0 to n

FOR j from i+1 to n

sum = 0

FOR k from i to j

sum = sum + List [k]

IF sum > currentMax

currentMax = sum

begin = i

end = j

END IF

RETURN List [begin, end], currentMax

**Algorithm 1 Theoretical Run-Time:**

There are three nested loops, i, j, and k. The k-loop is bound by O(j-i); the j-loop is bound by O(n\*n) and the outer i-loop is bound by O(n\*n\*n). Thus, the theoretical run-time could be summed as: T(n) =

=

= need to finish this…

**Algorithm 2, Better Enumeration, Pseudocode:**

MSABetterEnum(List [0…n])

currentMax = -∞

begin = -1, end = -1

FOR i from 0 to n

sum = 0

FOR j from i to n

sum = sum + List [j]

IF sum > currentMax

currentMax = sum

begin = i

end = j+1

END IF

RETURN List[begin, end] currentMax

**Algorithm 2 Theoretical Run-Time:**

T(n) =

=

=

=

=

**Algorithm 3, Divide and Conquer, Pseudocode:**

MSADivideAndConquer(List[0…n], begin, end)

IF begin >= end

RETURN .begin = -1, .end = -1, .sum = -∞

END IF

IF begin == end - 1

RETURN .begin = begin, .end = end, .sum = List[begin]

END IF

mid = (begin + end) / 2

Subarray left = MSADivideAndConquer(list, begin, mid)

Subarray right = MSADivideAndConquer(list, mid, end)

Subarray crossing = MaxCrossingSubarray(list, begin, mid)

IF left.sum >= right.sum AND left.sum >= crossing.sum

RETURN left

ELSE IF right.sum >= left.sum AND right.sum >= crossing.sum

RETURN right

ELSE

RETURN crossing

// Helper Function

MaxCrossingSubArray(List[0…n], low, mid, high)

begin = -1, end = -1

left\_highest\_sum = -∞

left\_sum = 0

FOR i from mid-1 to 0

left\_sum = left\_sum + A[i]

IF left\_sum > left\_highest\_sum

left\_highest\_sum = left\_sum

begin = i

END IF

right\_highest\_sum = -∞

right\_sum = 0

FOR i from mid to high

right\_sum = right\_sum + List[i]

IF right\_sum > right\_highest\_sum

right\_highest\_sum = right\_sum

end = i + 1

STRUCT Subarray sub

.begin = begin

.end = end

.sum = left\_highest\_sum + right\_highest\_sum

RETURN sub

**Algorithm 3 Theoretical Run-Time:**

This algorithm uses the divide and conquer method, thus its recurrence must be solved in order to determine the theoretical run-time. The MSADivideAndConquer function has the base case of n=0 and n=1, which would immediately return out of the function and hence take constant time: T(0) = O(1) and T(1) = O(1).

The function recursively calls itself twice when n > 1 and iterating over n/2 items per recursion. Furthermore, the function calls its helper function, MaxCrossingSubArray, which has two non-nested for-loops—this is the non-recursive work. Thus,

T(n) =

Solving using the Master Method: a=2, b=2, f(n) = n, log22=1, nlog2(2) = n1

Comparing f(n) = n to nlogb(a) = n, f(n) = Θ(n) indicates that Case 2 applies

T(n) = Θ(nlogb(a) \* lg n)

= Θ(n1 \* lg n)

= Θ(n lg n)

**Algorithm 4, Linear Time, Pseudocode:**

MSALinear(List[0…n])

Subarray current, max

FOR i from 0 to n

IF current.sum < 0

current.sum = List[i]

current.begin = i

current.end = i+1

ELSE

current.sum = current.sum + List[i]

current.end = i+1

END IF

IF current.sum > max.sum

max = current

END IF

RETURN max

**Algorithm 4 Theoretical Run-Time:**

This algorithm has one loop, the i-loop, that runs from 0 to n and there are only constant time operations within the loop. Thus, the time complexity could be summed as:

T(*n*) =

=

=