Project 1: Maximum Sum Subarray

#### Group 46

#### Group Members

David Brouillette

James Fitzwater

James Stallkamp

# Theoretical Run-time Analysis

## Enumeration

### Pseudo-Code

|  |
| --- |
| **on** enumerationAlgo(array)  **set** max **to** 0  **set** sumSoFar **to** 0  **set** start **to** 0  **set** end **to** 0  **repeat** **with** i **from** 0 **to** length of array **by** 1  **repeat** **with** j **from** i **to** length of array **by** 1  **repeat** **with** k **from** i **to** j **by** 1  **set** sumSoFar **to** sumSoFar + array[k]  if sumSoFar > max  **set** max **to** sumSoFar  **set** start **to** i  **set** end **to** k  **set** sumSoFar **to** 0  **end** **if**  **end** **repeat**  **end** **repeat**  **end** **repeat**  **return** max, array[start, end]  **end** max\_sub1 |

### Asymptotic Runtime

O(n^3). There are three(3) nested loops. This causes each loop to run exponentially more times then the loop it is inside of.   
This makes the T(n)=(n(n+1)(n+2))/6 = O(n^3)

## Better Enumeration

### Pseudo-Code

|  |
| --- |
| **on** betterEnumerationAlgo(array)  **set** max **to** 0  **set** start **to** 0  **set** end **to** 0  **repeat** **with** i **from** 0 **to** length of array **by** 1  **set** sumSoFar **to** 0  **repeat with** j **from** i **to** length of array **by** 1  **set** sumSoFar **to** sumSoFar + array[j]  if sumSoFar > max  **set** max **to** sumSoFar  **set** start **to** i  **set** end **to** j  **end** **if**  **end** **repeat**  **set** sumSoFar **to** 0  **end** **repeat**  **return** max, array[start, end]  **end** max\_sub2 |

### Asymptotic Runtime

O(n^2). There are two(2) nested loops, each running from a random number to the end of the array.   
This makes T(n)=n(n+1)/2 = O(n^2)

## 

## 

## Divide & Conquer

### Pseudo-Code

|  |
| --- |
| **on** getMaxCrossingSubarray(array, low, mid, high)  **set** leftSum **to** -infinity  **set** sum **to** 0  **set** maxLeft **to** NaN    **repeat** **with** i **from** mid **to** low **by** -1  sum = sum + (*item* i **of** array)  **if** sum > leftSum **then**  leftSum = sum  maxLeft = i  **end** **if**  **end** **repeat**    **set** rightSum **to** -infinity  **set** sum **to** 0  **set** maxRight **to** *none*    **repeat** **with** j **from** mid + 1 **to** high  sum = sum + (*item* j **of** array)  **if** sum > rightSum **then**  rightSum = sum  maxRight = j  **end** **if**  **end** **repeat**    **return** {maxLeft, maxRight, leftSum + rightSum}  **end** getMaxCrossingSubarray |

|  |
| --- |
| **on** divideConquerAlgo(array, low, high)  # Base Case of 1 element array  **if** high **is** **equal to** low **then**  **return** {low, high, *item* low **of** array}  **else**  **set** mid **to** int(**round** ((low + high) / 2) rounding *down*)  # return list for left half  **set** {leftLow, leftHigh, leftSum} **to** getMaxSubarray(array, low, mid)  # return list for right half  **set** {rightLow, rightHigh, rightSum} **to** getMaxSubarray(array, mid + 1, high)  # return list for crossing  **set** {crossLow, crossHigh, crossSum} **to** getMaxCrossingSubarray(array, low, mid, high)  # subarray on left has greatest sum  **if** leftSum ≥ rightSum **and** leftSum ≥ crossSum **then**  **return** {leftLow, leftHigh, leftSum}  # subarray on right has greatest sum  **else** **if** rightSum ≥ leftSum **and** rightSum ≥ crossSum **then**  **return** {rightLow, rightHigh, rightSum}  # subarray on across middle has greatest sum  **else**  **return** {crossLow, crossHigh, crossSum}  **end** **if**  **end** **if**  **end** getMaxSubarray |

### Asymptotic Runtime

O(n•lg(n)). This algorithm’s tree has a depth of lg(n) due to the input being divided by half until a single element input is reached. Each level or step has a sum up to n as the input is iterated over. Since the previous addition results are saved for use by the following n-th element, an addition of 1 is used at each iteration. This yields a constant time work.   
This makes the running time across lg(n) • n iterations yield O(n•lg(n))

## 

## 

## Linear Time

### Pseudo-Code

|  |
| --- |
| **on** linearAlgo(array):  **set** maxHere **to** 0  **set** maxSoFar **to** 0  **set** sum **to** 0  **set** start **to** 0  **set** end **to** 0  **set** startFinal **to** 0  **set** endFinal **to** 0  **repeat** **with** x **from** 0 **to** array length  **if** array[x] > maxHere + array[x]  **set** maxHere **to** array[x]  **set** start **to** x  **else**  **set** maxHere **to** maxHere + array[x]  **set** end **to** x  **end** **if**  **if** maxSoFar < maxHere  **set** maxSoFar **to** maxHere  **set** startFinal **to** start  **set** endFinal **to** end  **end** **if**  **end** **repeat**  **return** maxSoFar, array[startFinal, endFinal]  **end** max\_sub4 |

### Asymptotic Runtime

Since this algorithm only loops through the array one(1) time and there are no recursive calls, this makes T(n)=n. This is a linear relationship.

# Testing

### Process

Testing Process Here

# 

# 

# Experimental Analysis

## Enumeration

### 1.) Average Running Time Of Each n

### 2.) Plot of Average Running Times

### 

### 3.) Functional Relationship Model: Input Size and Time

### 4.) Discrepancies Between Running times

### 5.) Regression Model For Largest Input (n) To Be Solved In:

#### 10 Seconds

n =

#### 30 Seconds

n =

#### 60 Seconds

n =

### 6.) Log-Log Plot

### 

## 

## Better Enumeration

### 1.) Average Running Time Of Each n

### 

### 2.) Plot of Average Running Times

### 

### 3.) Functional Relationship Model: Input Size and Time

### 4.) Discrepancies Between Running times

### 5.) Regression Model For Largest Input (n) To Be Solved In:

#### 10 Seconds

n =

#### 30 Seconds

n =

#### 60 Seconds

n =

### 6.) Log-Log Plot

### 

### 

### Divide & Conquer

### 1.) Average Running Time Of Each n

### 2.) Plot of Average Running Times

### 

### 3.) Functional Relationship Model: Input Size and Time

### 4.) Discrepancies Between Running times

### 5.) Regression Model For Largest Input (n) To Be Solved In:

#### 10 Seconds

n =

#### 30 Seconds

n =

#### 60 Seconds

n =

### 6.) Log-Log Plot

### 

### 

## Linear Time

### 1.) Average Running Time Of Each n

### 

### 2.) Plot of Average Running Times

### 

### 3.) Functional Relationship Model: Input Size and Time

### 4.) Discrepancies Between Running times

### 5.) Regression Model For Largest Input (n) To Be Solved In:

#### 10 Seconds

n =

#### 30 Seconds

n =

#### 60 Seconds

n =

### 6.) Log-Log Plot

## Graph Presenting loglog Plot of All Four(4) Algorithms