CS325- Analysis of Algorithms
Project-1
Maximum Sum Subarray

Team:
Lakshman Madhav Kollipara
Shajith Ravi
Paris Kalathas

Pseudocode:

```
Algorithm 1: Enumeration

MaxSubarray(a[1 to n])
   Max = 0
   For i =(1 to n)
        For j = (i to n)
             Sum = 0
             For k from i to j
                  Sum = Sum + a[k]
                  If Sum > Max
                        Max = Sum
   Return Max
```

```
Algorithm 2: Better Enumeration

MaxSubarray(a[1 to n])
     Max = 0
     For i = (1 to n)
         Sum = 0
         For j = (i to n)
              Sum = Sum + a[j]
              If Sum > Max
                   Max = Sum
     Return Max
```

Algorithm 3: Divide & Conquer

```
MaxSubarray(a ,low,high)
    if (l == h)
       return a[l];
    int m = (l + h)/2;
    a=maxSubarray        (a,        l,        m),
    b=maxSubarray      (a,      m+1,      h),
    c=maxCrossingSum (a, l, m, h)
    return max(a,b,c)

maxCrossingSum (a,1,m,h)
    bothMaxLeft = 0;
    sum = 0;
    for i = (low to m)
         sum += a[i];
         if (sum > bothMaxLeft)
              bothMaxLeft = sum;
    bothMaxRight = 0;
    sum = 0;
    for i = (m to high)
          sum += a[i];
          if (sum > bothMaxRight)
               bothMaxRight = sum
    bothMax = bothMaxRight + bothMaxLeft;
```

Algorithm 4: Linear Time

```
MaxSubarray (a[1 to n])
   maxsum =0
   maxsuffix =0
   for i = (1 to n)
   if maxsuffix + array[i] > 0
       maxsuffix += array[i]
   else maxsuffix = 0
   if maxsuffix > maxsum
       maxsum = maxsuffix
   return maxsum
```

Run-time Analysis

Enumeration
In this algorithm, we have 3 nested for loops and the innermost loop is of Θ(n).
Each loop go through the entire array, which gives $\Theta(n^2)$* Θ(n). The summation equals to

$$\sum_{i=1}^{n}\sum_{j=i}^{n}(1+\sum_{k=i}^{n}1)$$

Which is equal to $(n^3 + 3n^2 + 2n)/3$

The asymptotic running time of this algorithm is $\Theta(n^3)$.

Better Enumeration

In this algorithm, we have 2 nested for loops and the innermost loop is of Θ(n).
Each loop go through the entire array, which gives Θ(n)* Θ(n). The summation equals to

$$\sum_{i=1}^{n}\sum_{j=i}^{n}(1+1)$$

Which is equal to n(n-1)/2

The asymptotic running time of this algorithm is $\Theta(n^2)$.

Divide &Conquer

In this algorithm, we are splitting the array into two halves (n/2 each) and processing each recursively. And we need to process the case when the subarray crosses the midpoint. We can find crossing sum in linear time by finding the maximum sum starting from mid-point and ending at some point on left of mid, then find the maximum sum starting from mid + 1 and ending with sum point on right of mid + 1. Finally, combine the two and return.

$T(n) = 2T(n/2) + \Theta(n), n>1$

Solve using Masters Theorem, Case 2

T(n) = Θ(n lg n)

Linear Time

In this algorithm, if n = 1, since it returns immediately, we have Θ(1).

Keeping track of the maximum subarray (maxsub) and maximum current array (maxcurrent), this will take Θ(n). With the summation for each subarray, we end up having an asymptotic bound of Θ(n) with the n terms cancel out.

$$\sum_{i=1}^{n}(1+2)$$

Which is equal to n.
The asymptotic running time of this algorithm is Θ(n).

Proof of Correctness:

Pre-conditions:

A is the array of n elements of which we want to find the subarray which has the maximum value if we add its elements.

Post-conditions:

The maximum sequential sum value is returned.

Recursive part

Base case:

If lo == hi therefore only one element is in the array which means that the sum is the content of that element.

Inductive hypothesis:

We assume that algorithm 3 finds successfully the maximum subarray summation for the array of n elements.

Inductive step:

We want to show that the algorithm finds the maximum subarray summation for array with n+1 elements.

Left recursive call nL = midpoint -> lo = n+1/2 <= n

Right recursive call nR = midpoint + 1 -> hi = n+1/2 <= n

The recursive calls divide the array in half until they finally reach the base case where lo == hi and then they return the value of that element.

Loop Invariant for the maxCrossingSum function

Initialization:

Initially for the left subarray the variable i is equal to the midpoint and for the right subarray

is equal to midpoint + 1. The variables sum, bothMaxLeft/Right are all equal to zero.

Maintenance:

In each iteration we add to the sum the value of the element a[i] and then we check if it is greater than the bothMaxLeft/Right and store it if it is. The variable i decreases for the left array and increases for the right array for the next iterations.

Termination:

Eventually the loop terminates when i reaches the lo value for the left subarray and hi for the right subarray and by then the maximum sum is stored in the bothMaxLeft/Right variables.

$T(n) = 2T(n/2) + n$, $n > 1$

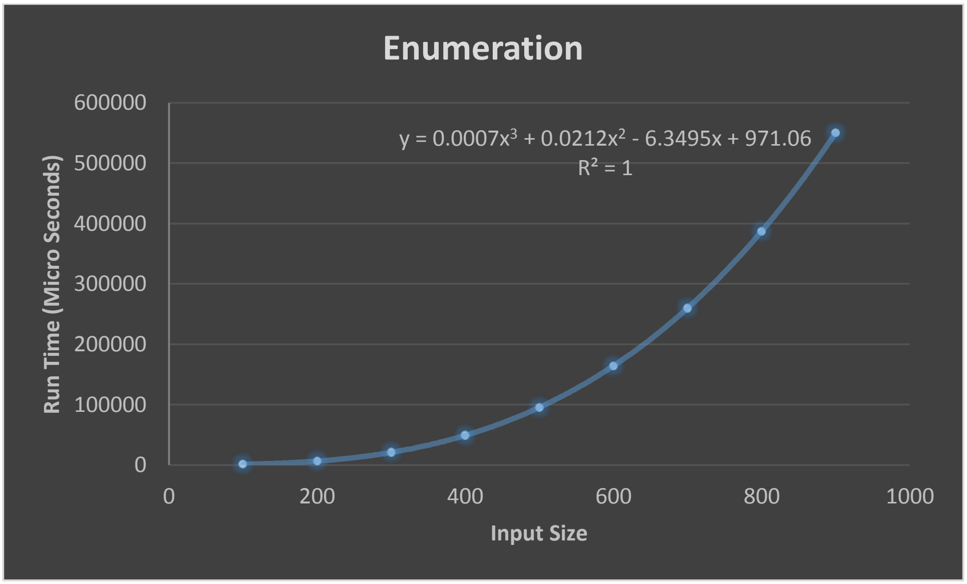
$T(n) = 1$, $n \leq 1$,

Testing:

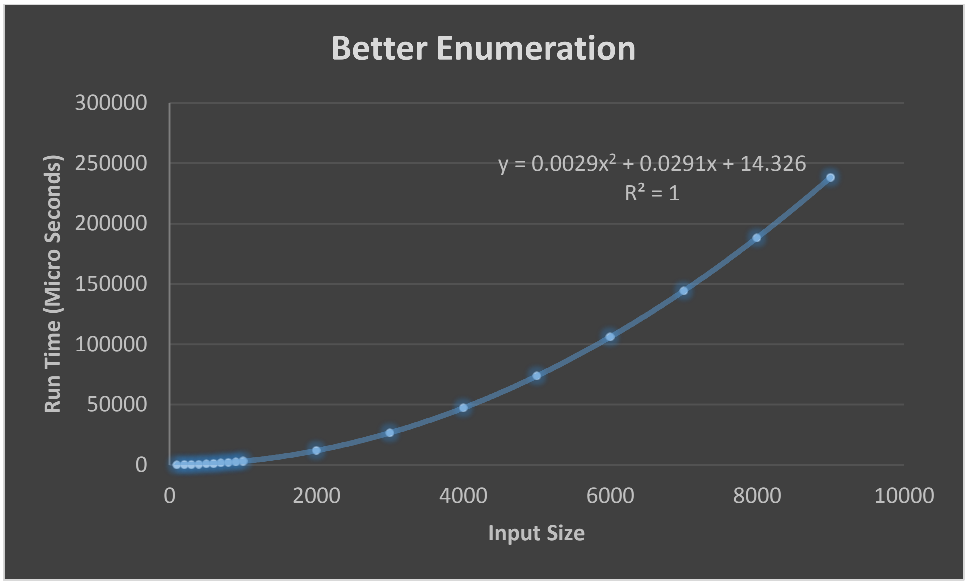
We randomly created 5 arrays and checked their Max sum from the web application, the instructor had included in the canvas and checked with the return values of all 4 Algorithms. If they are not equal, returns false and if they all are equal, return True. We also tested with your MSS_TestProblems.txt and verified output with MSS_testResults.txt. If I give empty array our algorithms takes it and does nothing. So I should add some exceptions and input validations but they are out of scope for this project.

Experimental Run-time Analysis:

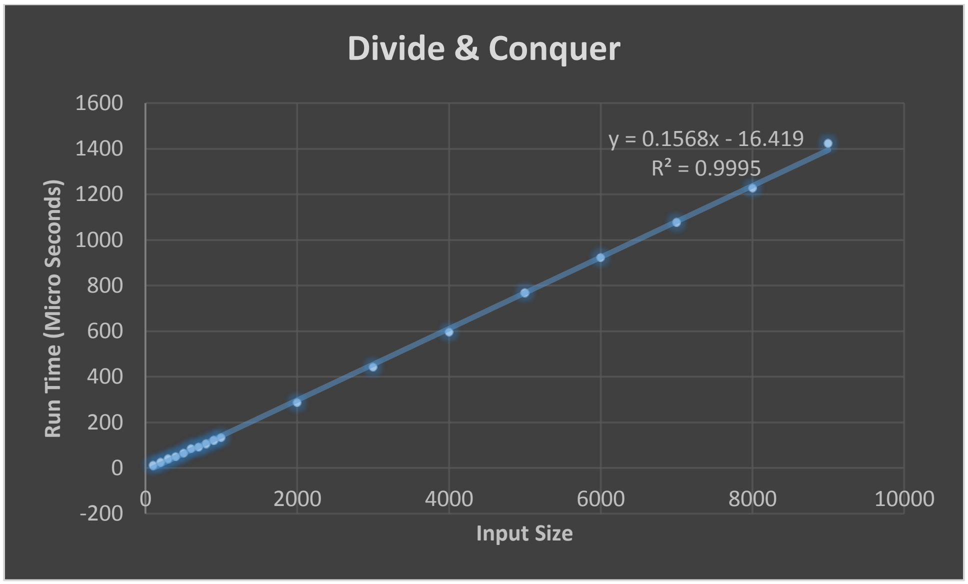
Array Size	Run Time (Micro Seconds)			
	Enumeration	Better Enumeration	Divide & Conquer	Linear Time
100	1302	31	11	1
200	6406	122	24	2
300	20909	269	39	3
400	49133	477	50	3
500	95293	750	65	4
600	164159	1075	84	5
700	260082	1455	92	6
800	387331	1917	106	7
900	550426	2492	121	7
1000		3017	134	8
2000		11869	286	16
3000		26526	443	25
4000		47140	595	32
5000		73636	768	40
6000		105914	922	54
7000		144168	1077	55
8000		188211	1227	65
9000		238310	1423	71



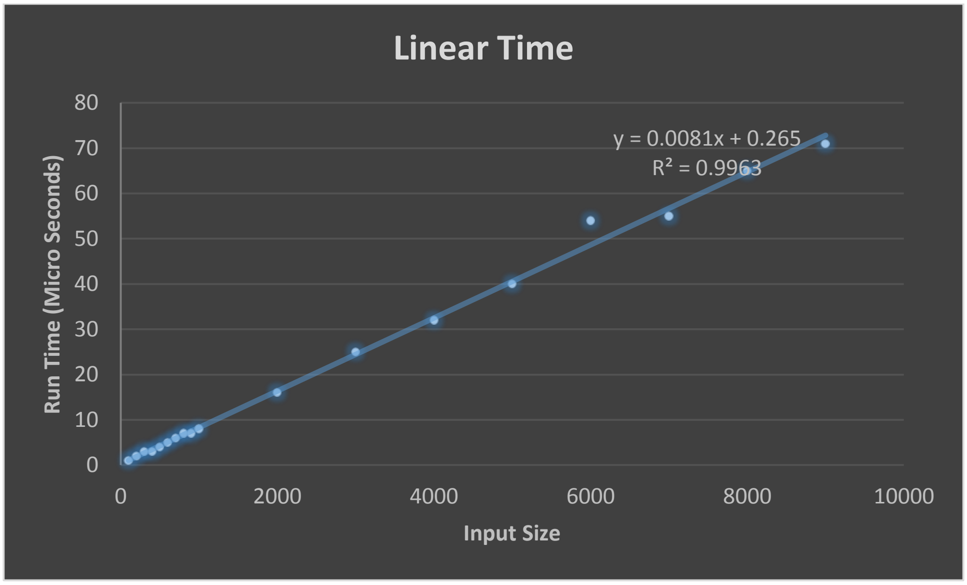
$y = 0.0027x^{2.8021}$ The Graph for Enumeration Looks Exponential (n^3).



$y = 0.0032x^{1.9906}$ The Graph for better enumeration looks exponential (n^2)

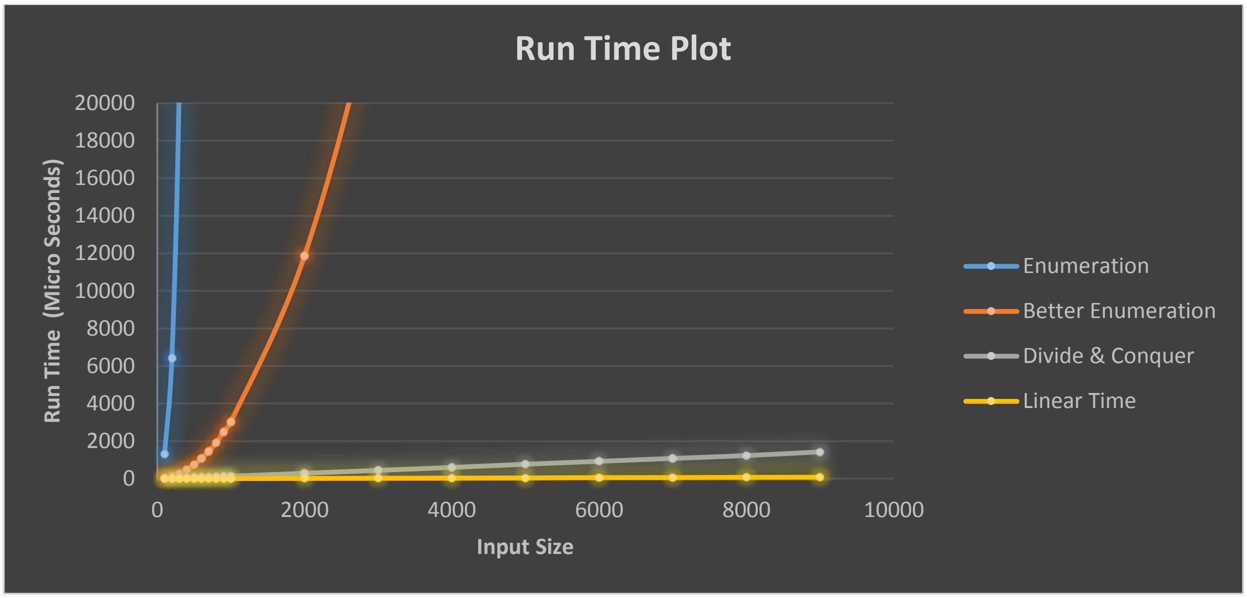


$y = 0.0834x^{1.0704}$ The Graph for Divide & Conquer looks linear up to some extent (n lg n)

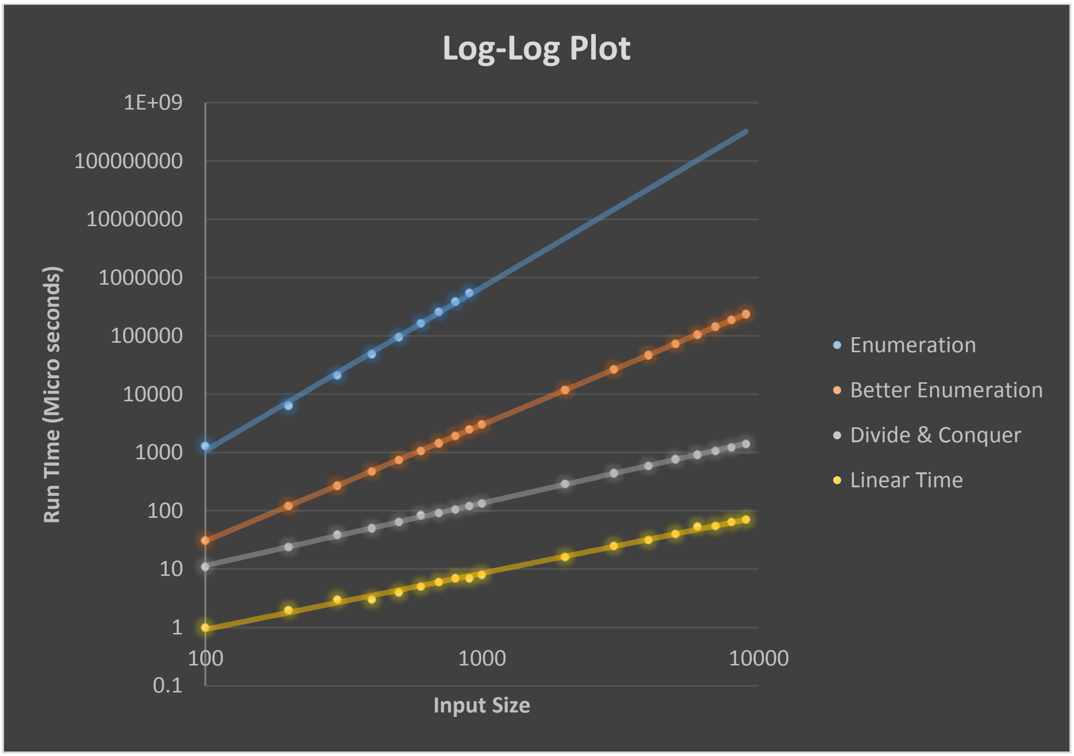


$y = 0.0109x^{0.9635}$ The Graph for Linear time looks linear till some extent and had small variations and again looks linear (n)

The Run Time of 10 minutes for Linear Time Algorithm is given by Input Size of 3127341.



The Y-axis can go till 6000000, But To show scope of all algorithms, plotted only till 20000.



From the Log-Log plot we get:

Experimental Runtime for the Enumerator algorithm is $\theta(n^{2.802})$. This is close to the theoretical runtime of $\theta(n^3)$.Experimental Run time for Better Enumeration algorithm is $\theta(n^{1.990})$. This is very close to the theoretical runtime of $\theta(n^2)$. Experimental runtime of Divide & Conquer is $\theta(n^{1.070})$. This is close to that of theoretical runtime $\theta(n \lg n)$. Experimental runtime of Linear Time is $\theta(n^{0.963})$. This is close to that of theoretical runtime $\theta(n)$.

The Log-Log plot also shows small discrepancies between the experimental and theoretical runtime data. Which might be some glitch in Micro seconds calculation and slope calculation and sometimes on number of processes running and Cache availability (Hit/Miss). But our experimental results are almost close to theoretical results.

Submitted By:

Lakshman Madhav Kollipara

Shajith Ravi

Paris Kalathas

For CS325_Project 1