# CS 325
Programming Assignment 1
Kevin Stine

## Pseudocode:
### Algorithm 1: Enumeration
enumeration(array of integers)

max sum = 0

for i = 0 to length of array

for j = i to length of array

new sum = sum of array from i to j+1

if new sum > max sum

max sum = new sum

The first loop iterates through the array n times
The second loop iterates through the array n times as well

### Algorithm 2: Better Enumeration
betterEnumeration(array of integers)

max sum = 0

for i = 0 to length of array

old sum = 0

for j = i to length of array

if j = 0

newSum = array[i]

else

newSum = oldSum = array[j]

if newSum > newSum

maxSum = newSum

oldSum = newSum

This algorithm iterates through the array while saving the previously calculated subarray summation. The next summation can be computed using the previous summation + value at the next index

### Algorithm 3: Dynamic Programming
dynamicProgramming(array of integers)

best = 0

cur = 0

for i in array

cur = max(cur + i, 0)

best = max(best, cur)

This algorithm iterates through the array in one for loop, calculating the subarray using the maximum subarray at the previous position

**Run-time Analysis:**

1. Enumeration:
   a. This algorithm simply iterates through all elements of the array and compares the value of the sum through the array and the current maximum sum. If the new sum is greater than the current max sum, the max sum is changed to the new sum. Since this is a nested for loop, the runtime complexity is $O(n^2)$
2. Better Enumeration:
   a. This algorithm also iterates through all elements of the array in a nested for loop. It improves on the enumeration algorithm and retains the previously calculated subarray summation, causing the runtime complexity to be $O(1)$
3. Dynamic Programming
   a. This algorithm scans through the array values and computes the maximum subarray at each position. This algorithm only contains one for loop, therefore the runtime complexity is $O(n)$

**Experimental run-time analysis:**

| Array Size | Enumeration | Better Enumeration | Dynamic Progamming |
|---|---|---|---|
| 100 | 0.004149175 | 0.00116322 | 3.66E-05 |
| 200 | 0.022109008 | 0.003842902 | 7.23E-05 |
| 300 | 0.069082904 | 0.008721519 | 0.000106812 |
| 400 | 0.144295764 | 0.015424466 | 0.000141597 |
| 500 | 0.264735079 | 0.02410059 | 0.000180435 |
| 600 | 0.447789502 | 0.034599566 | 0.000215435 |
| 700 | 0.659637737 | 0.047356296 | 0.000244284 |
| 800 | 0.955656314 | 0.061542201 | 0.000279093 |
| 900 | 1.372023845 | 0.078342319 | 0.000311494 |
| 1000 | 1.887161684 | 0.096659851 | 0.000348639 |
| 1100 | 2.518984914 | 0.116932917 | 0.000378275 |
| 1200 | 3.210728073 | 0.13841095 | 0.000417948 |
| 1300 | 4.235153914 | 0.163273597 | 0.000456309 |
| 1400 | 5.125639558 | 0.189263821 | 0.000495434 |
| 1500 | 6.41123991 | 0.218218136 | 0.000519896 |
| 1600 | 7.495007372 | 0.246972108 | 0.000556684 |
| 1700 | 9.15910275 | 0.279858804 | 0.000585341 |
| 1800 | 10.83526502 | 0.313653302 | 0.000648212 |

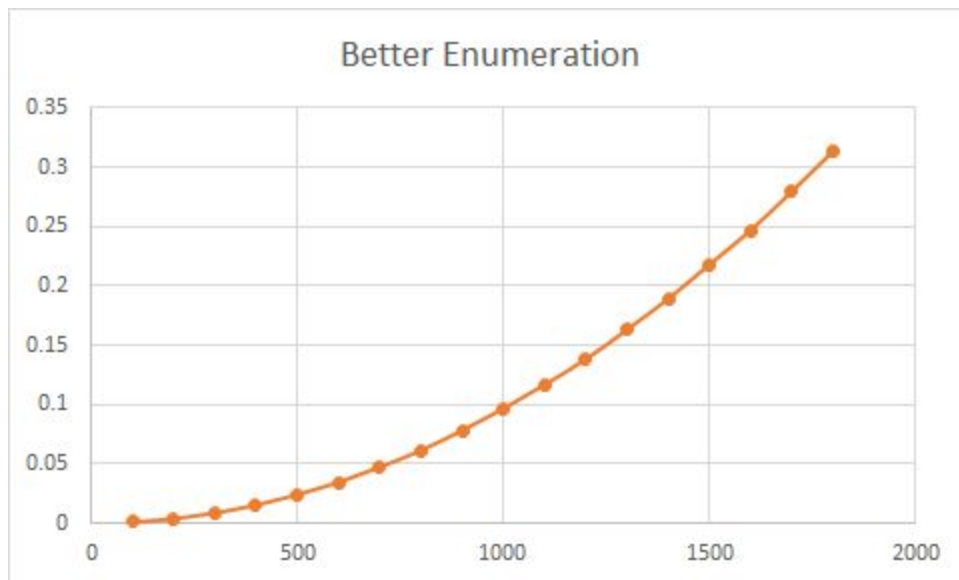**Table 1:** *Raw collected data*
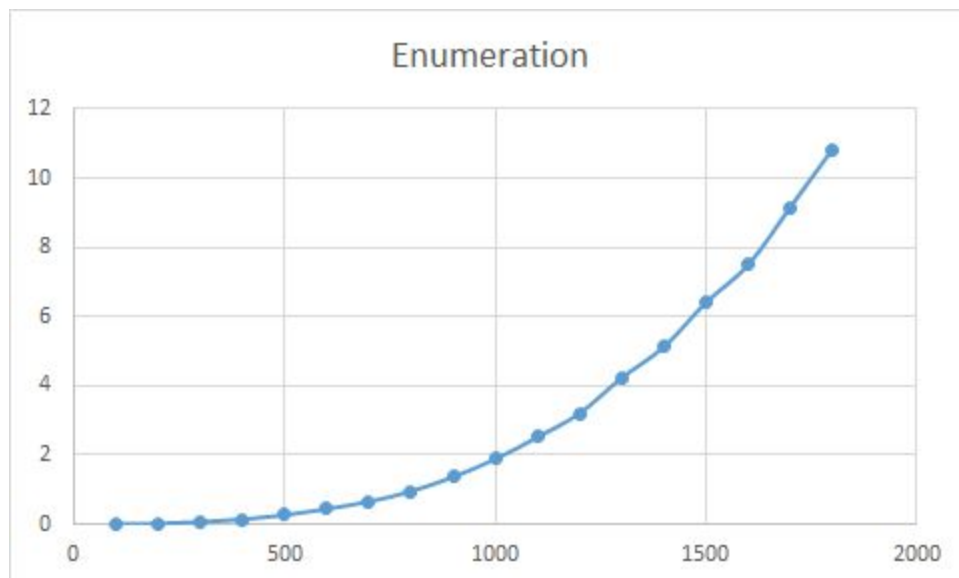
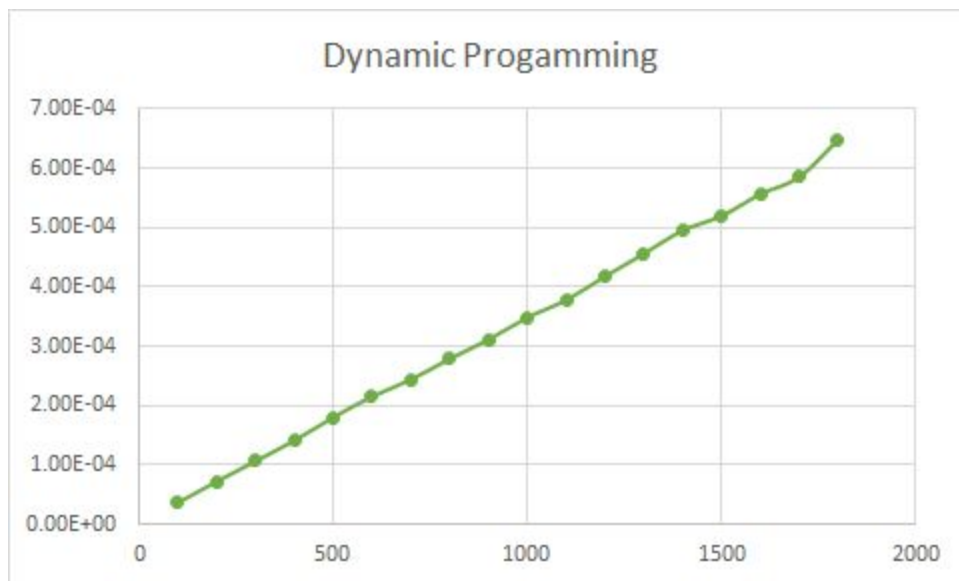**Figure 2:** *Better enumeration plot time vs. array size*



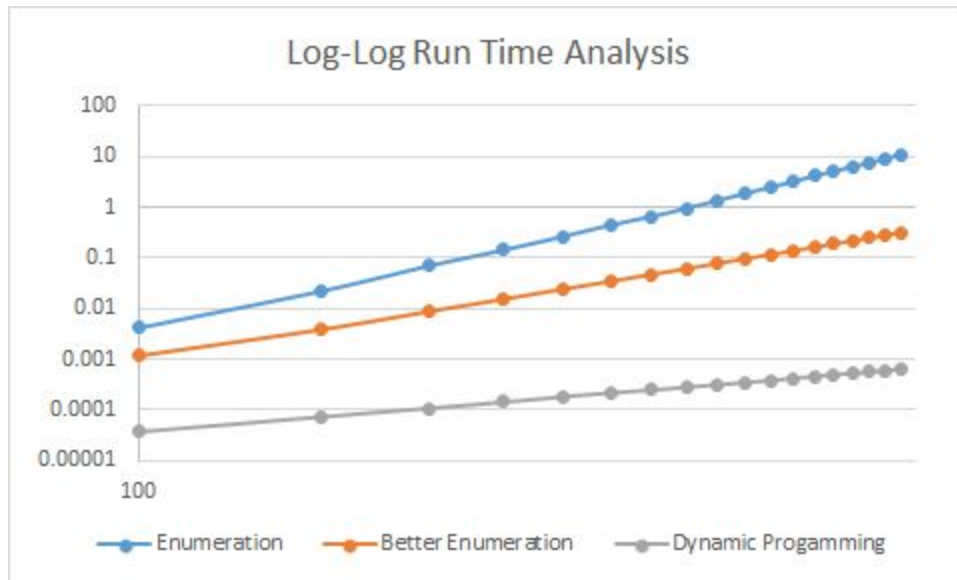**Figure 3:** *Dynamic programming plot time vs. array size*

**Figure 4:** *Log-log run-time analysis plot*

The results of the runtime analysis graph show that there was not much difference between asymptotic runtime and experimental runtime. The graph for Dynamic programming which should be $O(n)$ has a graph extremely close to what $O(n)$ looks like. The graph for enumeration which should be $O(n^2)$ also has a graph extremely similar to what $O(n^2)$ looks like.

| Algorithm | Slope |
|---|---|
| Enumeration | 5.932e-3 |
| Better Enumeration | 1.837e-4 |
| Dynamic Programming | 3.49089e-7 |