# Group Assignment 1

Jing Huang
933-039-277
Xinwei Li
933-666-663

## I. Presudo-Code

1. Enumeration

```
ENUMERATION(a[1,...,n])
    define a result variable
    for i = 1:n
        for j = i:n
            define a current variable
            for k = j:n
                add two numbers together and store it in current
                take the max of current and result and store it in result
    return max sum found so far
```

2. Better Enumeration

```
BETTER_ENUMERATION(a[1, ..., n])
    for i = 1:n
     sum <- 0
     for j = i:n
        current += a[j]
        if sum < current OR i <- 0
            sum <- current
            keep max sum found so far
  return max sum found so far
```

3. Dynamic Programming

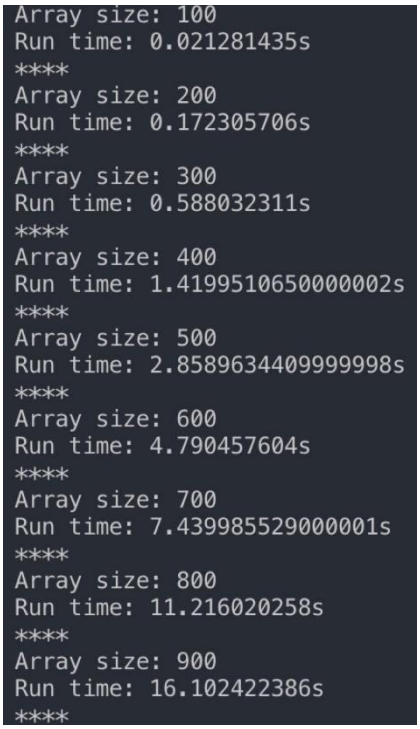
```
DYNAMIC(a[1,...,n])
    define a variable named current
    define a variable named result
    for i = 1:n
        current += a[i]
        if current less than 0 then set current to 0
```

take the max of current and result and store it in result
return max sum found so far

## II. Run-time Analysis

1. Enumeration

The run-time of Enumeration Algorithm is O(n^3), and the sum of it is $\sum_{i=1}^{n}\sum_{j=i}^{n}\sum_{k=j}^{n} 3k+2$, which means that running every size of input array is much slower, the output of runtime is below. As the picture shows that this algorithm is not an optimized one, it increases faster when it comes to 400 array size, which is nearly three times of 300 array size.




2. Better Enumeration

The run-time of Better Enumeration Algorithm is O(n^2). It means that running every size of input array is slower than O(n), which is the best run-time of maximum subarray, and the sum of it is $\sum_{i=1}^{n}\sum_{i=j}^{n} 2j+3$, the output of runtime is below. As the picture shows, run time increases rapid when it comes to 4000 array size, which is nearly twice of 3000 array size.

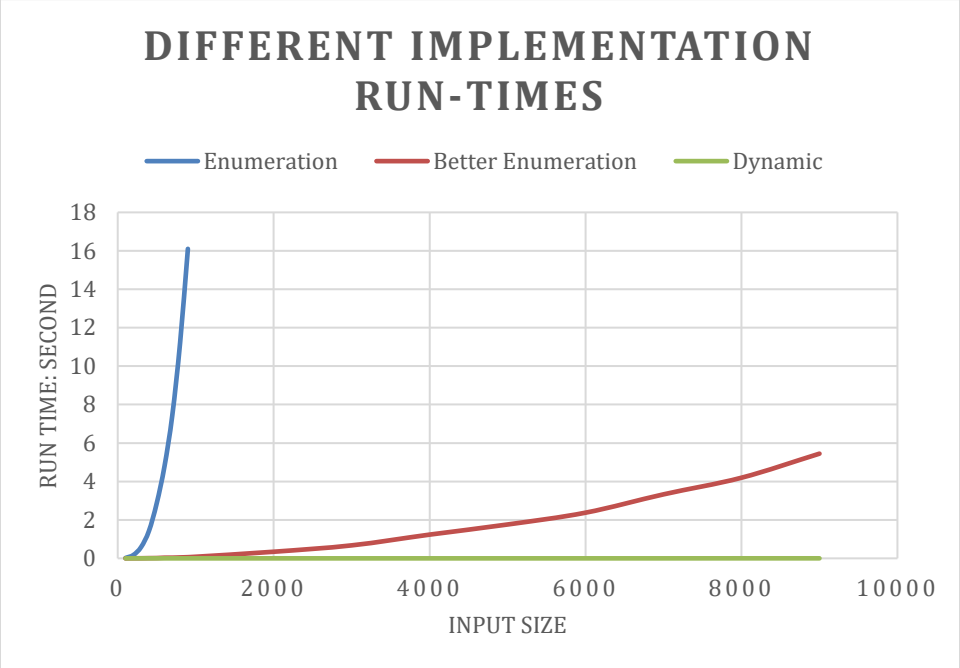
```
better_enumeration

Array size: 100
Run time: 0.0006323790000000024 s
****
Array size: 200
Run time: 0.004530359999999997 s
****
Array size: 300
Run time: 0.005592859999999998 s
****
Array size: 400
Run time: 0.011794885999999997 s
****
Array size: 500
Run time: 0.016973205000000005 s
****
Array size: 600
Run time: 0.034041493000000006 s
****
Array size: 700
Run time: 0.03671088799999998 s
****
Array size: 800
Run time: 0.048138226000000006 s
****
Array size: 900
Run time: 0.06147872400000001 s
****
Array size: 1000
Run time: 0.08355299100000002 s
****
Array size: 2000
Run time: 0.34362644900000006 s
****
Array size: 3000
Run time: 0.6754097010000001 s
****
Array size: 4000
Run time: 1.234526448 s
****
Array size: 5000
Run time: 1.7622367620000001 s
****
Array size: 6000
Run time: 2.3761689949999996 s
****
Array size: 7000
Run time: 3.3299819150000003 s
****
Array size: 8000
Run time: 4.195108458 s
****
Array size: 9000
Run time: 5.446601301999998 s
****
```

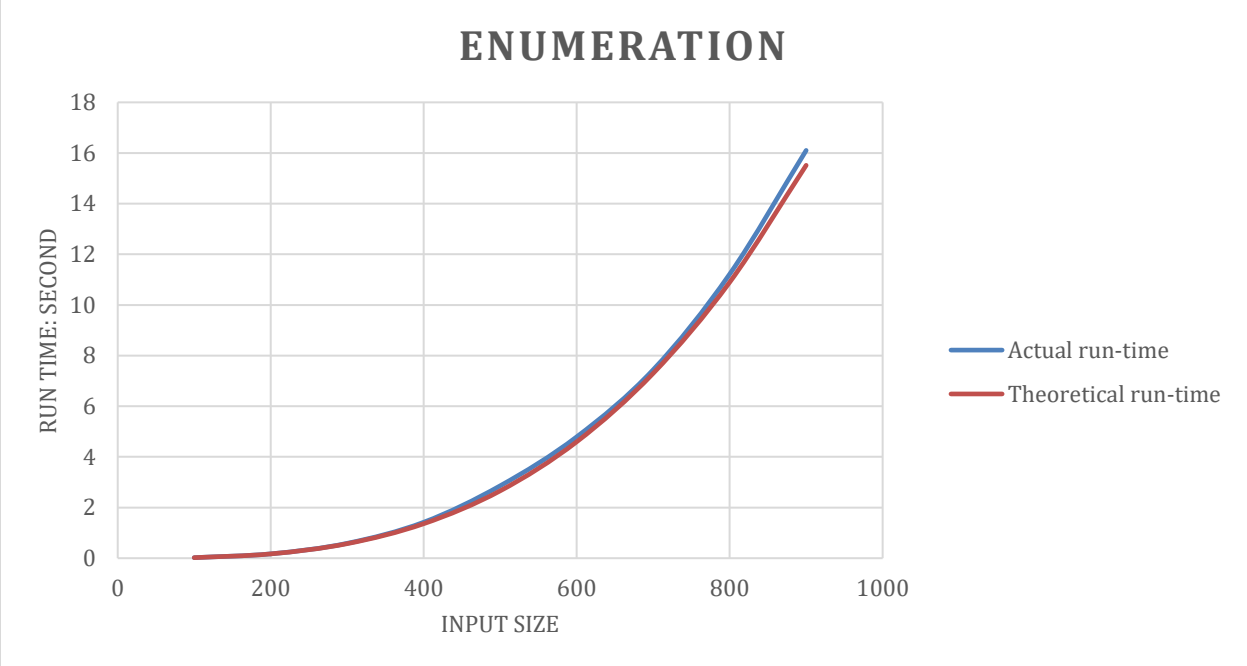
3. Dynamic Programming

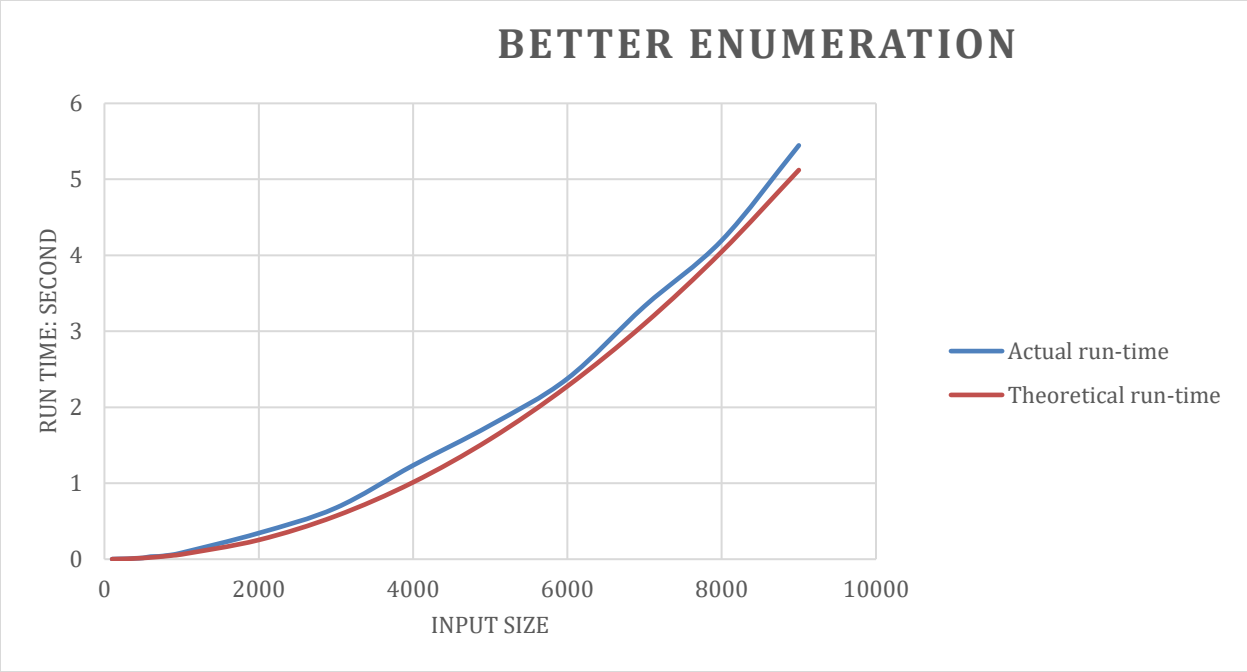
The run-time of Dynamic Programming Algorithm is O(n), which means that this is the best solution for maximum array problem among these three algorithms, and the sum of it is $\sum_{i=1}^{n} 3i + 2$ The output of runtime is below, as the picture shows that it increases gradually, and the time is much better than the Better Enumeration Algorithm.

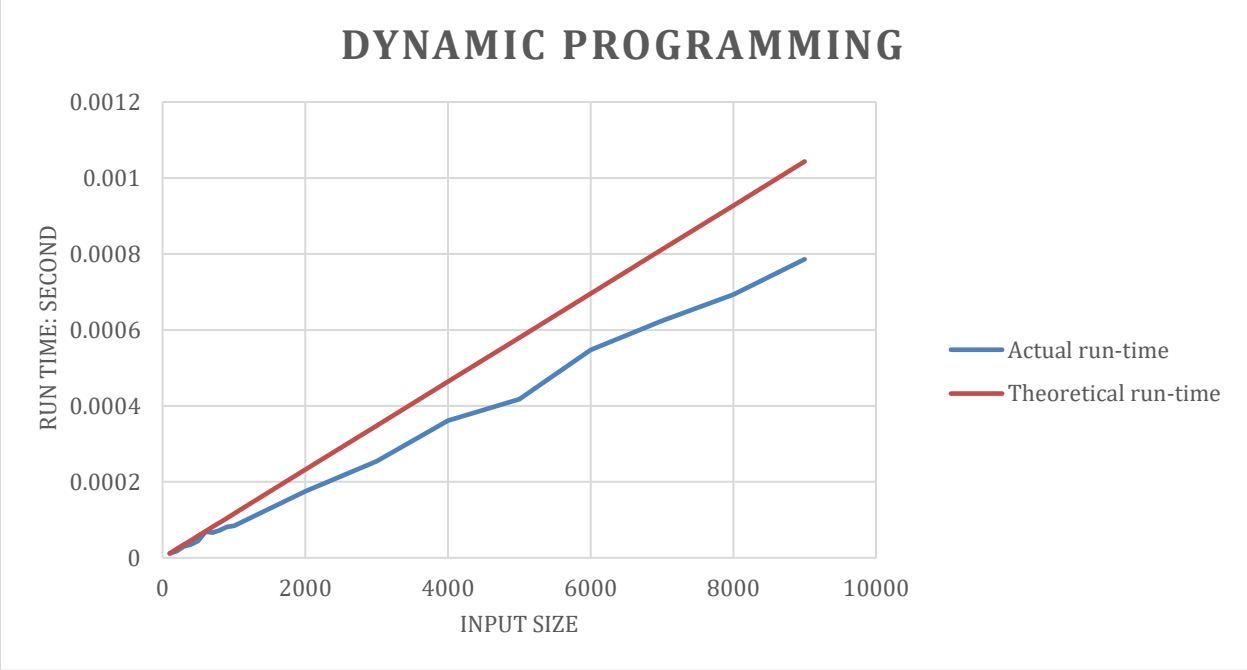
```
dynamic

Array size: 100
Run time: 1.1594000000003657e-05 s
****
Array size: 200
Run time: 1.762300000000827e-05 s
****
Array size: 300
Run time: 3.0145000000009192e-05 s
****
Array size: 400
Run time: 3.478199999999709e-05 s
****
Array size: 500
Run time: 4.40579999999996e-05 s
****
Array size: 600
Run time: 6.863799999999587e-05 s
****
Array size: 700
Run time: 6.631899999999524e-05 s
****
Array size: 800
Run time: 7.234799999999986e-05 s
****
Array size: 900
Run time: 8.115899999999732e-05 s
****
Array size: 1000
Run time: 8.394200000000351e-05 s
****
Array size: 2000
Run time: 0.0001753050000000006 s
****
Array size: 3000
Run time: 0.0002541449999999973 s
****
Array size: 4000
Run time: 0.000360812000000002 s
****
Array size: 5000
Run time: 0.00041785499999999476 s
****
Array size: 6000
Run time: 0.0005472470000000007 s
****
Array size: 7000
Run time: 0.0006237679999999968 s
****
Array size: 8000
Run time: 0.0006933340000000038 s
****
Array size: 9000
Run time: 0.0007860879999999904 s
****
```

## III. Experimental Run-time Analysis






We can know from the plot that the actual run-time of our implementation is similar to the theoretical run-time. And the most efficient algorithm implementation is the dynamic one, the second is the better enumeration, and the last one is the enumeration. There are some deviations because the compiler needs extra cost for processing.