Group Programming Assignment 1

Prof. Xiaoli Fern

Due: Thursday, January 14 at 2PM

By:

Spike Madden

Nicholas Jake Jeffreys

Zachary Iverson

Algorithm 1: Enumeration

Pseudocode:

- Declare variables
- loop through the beginning of the array to the end of the array
    - loop again starting at the index of the first loop to end of the array
        - assign a variable called “range” to the value of the index of the second loop(inner loop) minus the index of the outer loop
            - range = index of inner loop - index of outer loop
        - set the “sum” variable to zero
            - sum = 0
        - another loop starting at zero through to the “range” variable value
            - add the value at the current index to the “sum” variable
                - sum = sum + array[i]
            - check to see if the value in “sum” variable is bigger than the value in the “biggest” variable
            - the “biggest” variable is the current biggest sum of the array
                - if the sum is bigger than biggest, replace bigger with the value in sum.
                    - biggest = sum
- Print out the variable “biggest”

Run-time analysis :

Sum - $max\sum_{i=0}^{size\ of\ array}\left(\sum_{m=i}^{size\ of\ array}\left(\sum_{j=0}^{range\ =\ m\ -\ i}\left(array[j]\right)\right)\right)$

Asymptotic Bound - O(n^3)

Algorithm 2: Better Enumeration

Pseudocode:

- Declare Variables
- Loop going through the beginning of the array to the end of the array
    - loop through going from the beginning of the array to the end of it
        - add the number of the index into a “current sum” variable
            - sum = sum + array[index[
        - check to see if the “biggest sum” variable is greater than the “current sum” variable
            - if it is bigger than replace the value in “biggest sum” variable with the value of “current sum variable”
                - biggest = sum

- Print the “biggest sum” variable

Run-time analysis:
Sum -

$$max \sum_{i=0}^{size\ of\ array} \left(\sum_{m=i}^{size\ of\ array} (array[m]) \right)$$

Asymptotic Bound - O(n^2)

Algorithm 3: Dynamic Programming

Pseudocode:

- Declare variables
- a loop starting from the begin of the array to the end of the array
    - check if value at that index if it is less than 0
        - if is less than 0, put the biggest sum in the current sum variable
        - it will either be 0 or the current sum plus the value in at the current index of the array into the current sum variable
            - sum = biggest number (which will be 0 or current sum + array[current index]
    - else if the the value at the index is greater than 0
        - put the max sum variable into the current sum
            - check if either the value at that current index is greater than the current sum plus the value at the current index in the array
                - sum = biggest number of value in the array[current index] or sum + array[current index]
    - check which of these variable “current biggest” variable and “ current sum” variable. have the greater value. Put that into the “current biggest” variable
- Print the “current biggest” variable after you are outside all of the loops

Run-time analysis:

$$max \sum_{i=0}^{size\ of\ array} (array[i])$$

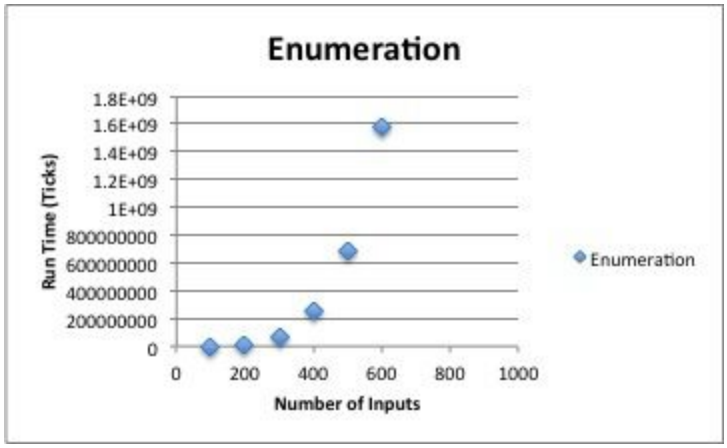
Sum -

Asymptotic Bound - O(n)

Experimental run-time analysis (Plots of Run Time with respect to Input Size)

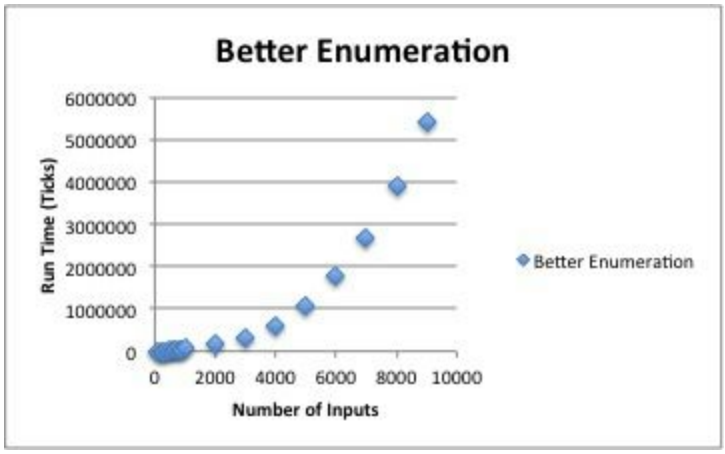
Enumeration

The graph below shows the enumeration algorithm run time relationship. As the number of inputs(X-axis) gets bigger by a constant 100, the run time(Y-axis) gets dramatically bigger. You can see this by the curve that the plots make on the graph.



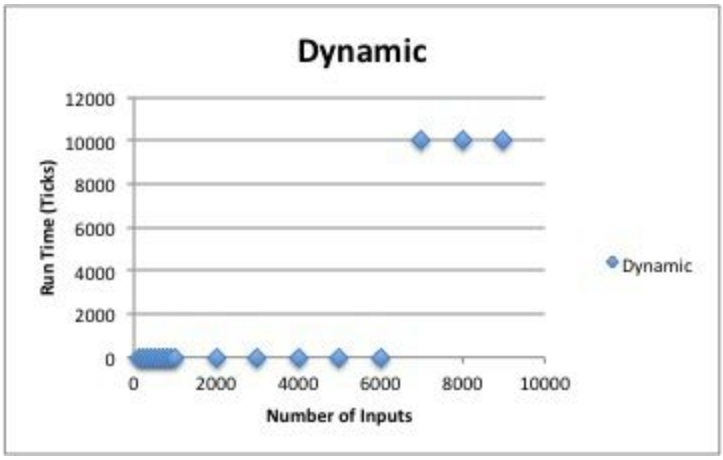
Better Enumeration

The graph below shows the Better Enumeration algorithm run time relationship. As the number of inputs(X-axis) gets bigger by a constant 100, the run time(Y-axis) gets consistently bigger.



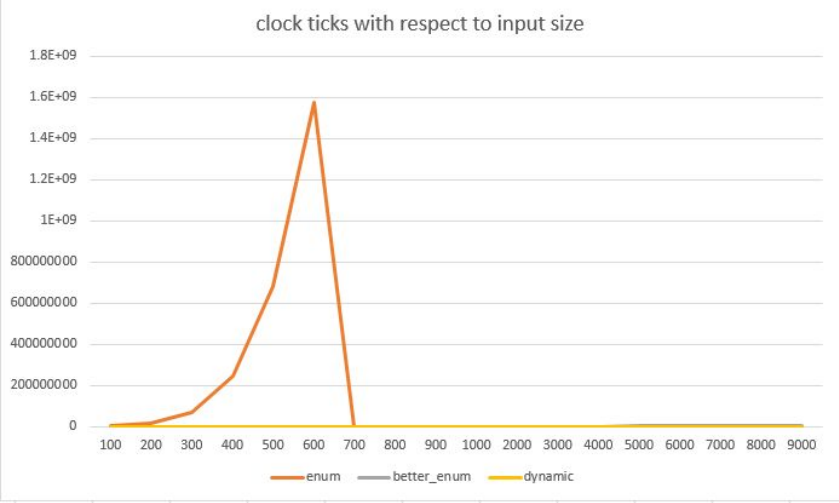
Dynamic

As the number of inputs (X-axis) increases at a constant rate, the runtime starts with a consistent 0 ticks on runtime, until it hit input sizes bigger than 6000. After it follows steady run time behavior at 10,000 ticks.

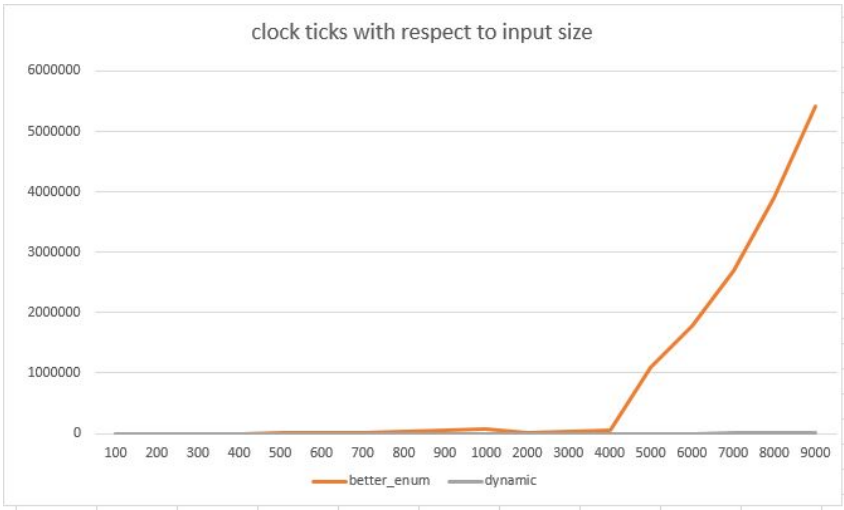


Comparison Charts:

Notes:
This chart below compares all three methods. Values for enumeration are far greater than those for better_enumeration and dynamic to the point that you can't see the other values. Program started producing garbage numbers after 600 inputs for enumeration algorithm because the variable overflowed.

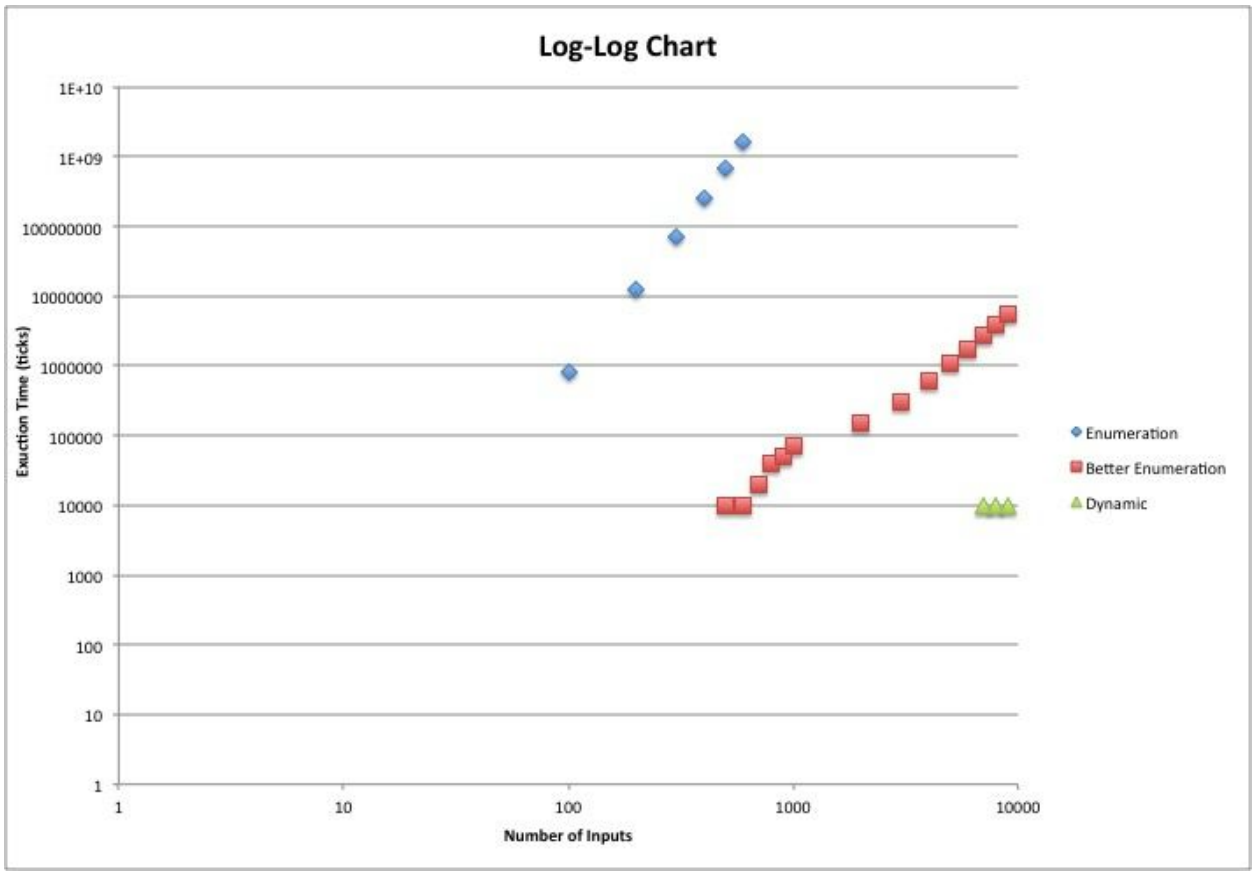


Notes:
This chart below only compares the better enumeration method and the dynamic method. Values for better_enumeration are far greater than the numbers from dynamic.



Log-Log charts:

Notes:
Enumeration method has the worst run time with the steepest slope making this approach quite ineffective when dealing with a large number of inputs. The dynamic algorithm has the best run time and appears to have a minimal slope showing that it is possible to quickly execute very large number of inputs with this approach.



Notes:

On the Log-Log chart below we found lines of best fit in order to compare the slopes. The Enumeration method has a slope of 3•10^6, the Better Enumeration method has a slope of 489, and the dynamic method has a slope of 0.758. These slopes show that the enumeration method has a very difficult time dealing with large inputs. The dynamic method is much more efficient for large input numbers. The Better Enumeration method lies in between these two slopes. These results agree with our predictions for asymptotic growth outlined at the beginning of our report. Not only were we right about the order but also about the scale to which they differed.