Homework 3: Q2

**Name:** Ali Morshid, alimorsh

*Don’t forget to input your list of collaborators and sources on* ***AutoLab****.*

**Please submit this file as a PDF.**

# **Algorithm Idea**

Given an arbitrary vector V of length n:

|  |  |
| --- | --- |
| 0 | V1 |
| 1 | V2 |
| .  .  . | .  .  . |
| n-1 | Vn-1 |

We can also get an upper triangular matrix Un of length n from the given vector Vn:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| V0 | V1 | ... | ... | Vn-1 |
| 0 | V1 | ... | ... | Vn-1 |
| 0 | 0 | V2 | ... | Vn-1 |
| 0 | 0 | 0 | V3... | Vn-1 |
| 0 | 0 | 0 | 0 | Vn-1 |

In order to get our desired results we can iterate through the vector n^2 times using the following algorithm:

let x be our result variable

let Y be our result vector

**for(int i=0; i<n; i++)**

**x=0**

**for(int j=i; j<n; j++)**

**x = x + V[j];**

**Y[i] = x;**

to get the following result vector:

|  |  |
| --- | --- |
| 0 | V0+V1+...+Vn-1 |
| 1 | V0+V1+...+Vn-1 |
| .  .  . | .  .  . |
| n-1 | V0+V1+...+Vn-1 |

But this algorithm is unnecessarily complex given the unique structure of the upper triangular matrix. We can achieve a more efficient time complexity if we take advantage of its unique structure by iterating backwards from index n-1 to 0, we can use the same result variable x by only iterating n times through the vector. Since the last entry in Un is just:

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 0 | ...0 | Vn-1 |

Our result variable x is just Vn-1, and in the second to last entry in Un:

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | ...0 | Vn-2 | Vn-1 |

We can see that if we substitute our result variable x for Vn-1 and add only Vn-2, we can get the result we want outputted to our result vector.

1. **Algorithm Details**

Keeping the algorithm idea in mind, we can come to the conclusion that the best way to approach this problem is by iterating backwards through the input vector and keep the value of our result variable x through iterations and just add one more value at a time.

The following algorithm should do the trick:

//begin algo

**Let n be the input vector size**

**Let x be our result variable**

**Let V be our input vector**

**Let Y be our output vector**

**x=0;**

**for(int i=n-1; i>=0; i--)**

**x = x + V[i]**

**Y[i] = x;**

//end algo

So by making use of the unique structure of an upper triangular matrix, we were able to reduce the runtime quite exponentially.

1. **Big-Oh Analysis**

To analyze the time complexity of this algorithm, we first analyze the statements within the main loop then go outwards:

Let Sn be the statement number

S1: x=0 //assignment with constant runtime O(1)

S2: x = x + V[i] // assignment in O(1), addition in O(1) and query by index in O(1)

// O(1)+O(1)+O(1) = O(1)

S3: Y[i] = x //query by index O(1), assignment O(1)

//O(1)+O(1) = O(1)

So all of the statements run in constant time or O(1), so we move on to the main loop which runs from n-1 to 0 or in O(n) time

Let g(n) be the algorithm we used and let T(n) be the runtime for g(n):

T(n) = runtime of main loop x (runtime of statements inside the main loop)

**= O(n) x (O(1)+O(1)) = O(n)**

Now we add the statements outside of the main loop

**= O(n) + O(1) = O(n)**

**We conclude that the algorithm runs in O(n) time.**

1. **Big-Omega Analysis**

Since there is no way for the algorithm to run in less times than O(n) because we have to iterate through every value in the input vector inside the main loop, and we cannot possibly break out of the loop early, we can conclude that the lower bound of the algorithm runtime is Ω(n)

And since the algorithm runs in both O(n) and Ω(n), we can conclude that it also runs in Θ(n) time.