Stefan Emmons

COSC-3020-01

Lab 05

10-15-2019

Part 2: Runtime analysis

To begin with an effective runtime analysis, let's take a look at the code in our function:

```
17  function convertToAdjList(adjMatrix) {
18      var adjList = [];
19      for(var i = 0; i < adjMatrix.length; i++) {
20          verInd = [];
21          for(var j = 0; j < adjMatrix.length; j++) {
22              if(adjMatrix[i][j] == 1)
23              {
24                  verInd.push(" -> " + [j]);
25              }
26              adjList["vertex " + i] = verInd;
27          }
28      }
29      return adjList;
30  }
```

Line 18 → Constant, θ(1)

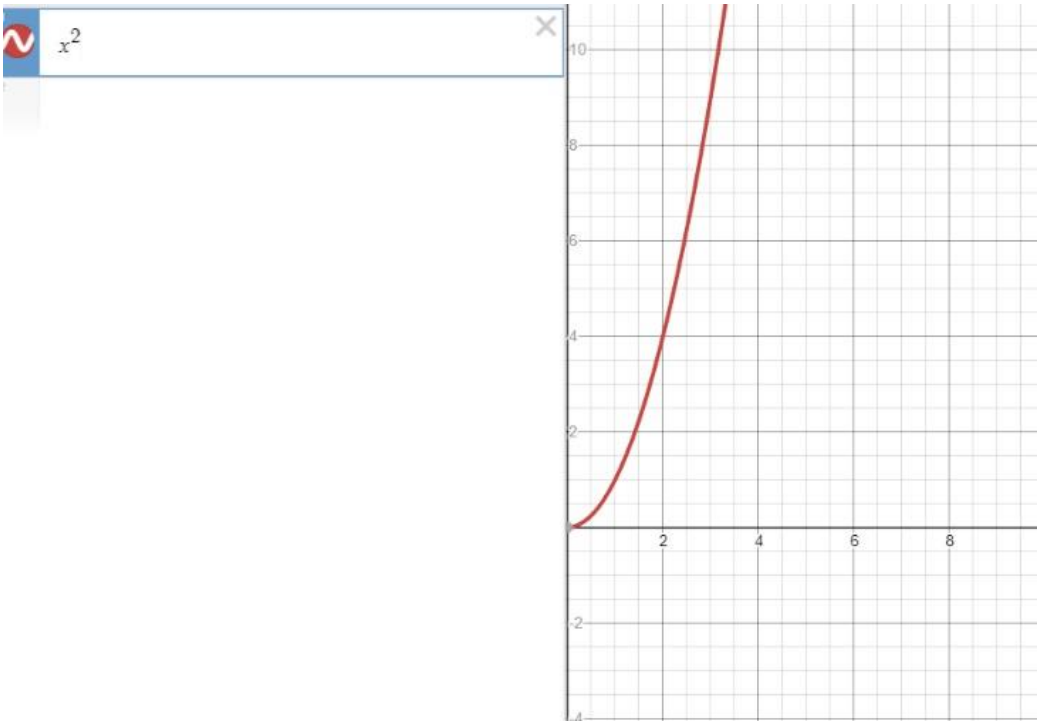
Line 19 → Linear, runs for the duration of the matrix, θ(|V|)

Line 20 → Constant, θ(1)

Line 21 → Linear, runs for the duration of the matrix, θ(|V|)

Lines 22-29 → Constant, θ(1)

This runtime complexity should be modeled asymptotically, meaning that we can drop all constants and only worry about the duration of the two loops used here. With this in mind, we can see that the runtime here is $\theta(|V|^2)$. This can be seen intuitively, as we use the entire length of the matrix twice for loop check conditions. To visualize this kind of growth, a graph is provided:



In answer to the second part of this question, for our particular conversion, it depends on both the number of vertices and edges. This is because we are observing the adjacency relations between all vertices and the subsequent edges that are, or are not connected to them.

The average-case time complexity for converting a list to a matrix would likely be $\theta(|V| \times |E|)$, as we would need to iterate over all vertices available, and check if edges exist by linearly iterating through our adjacency list. However, this would need to be done with two nested "for" loops, one for vertices, and one for edges.