OM-S20-03: Basics of Graphs and Complexity Theory

Aditya Bharti, Kritika Prakash

IIIT Hyderabad

10 Jan 2020

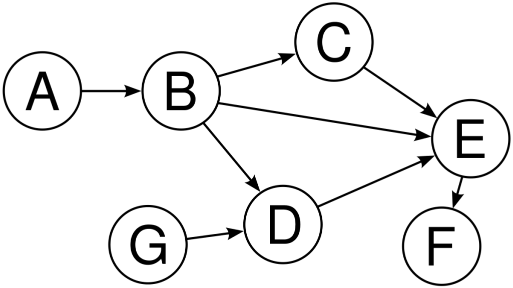
Graph Theory Basics

- What is a Graph?
- Graph Characteristics
- Graph Representations
- Types of Graphs
- Algorithms

What is a Graph?

A graph is an ordered pair G = (V, E) comprising

- V: a set of vertices (also called nodes or points)
- $E \subseteq \{(x, y) | (x, y) \in V^2\}$: a set of edges



Graph Characteristics

- Directedness of edges
- Degree of a vertex
  - Directed: In degree, Out degree
  - Undirected: Degree
- Edge weights
- Connectedness

Graph Representations

- Adjacency Matrix
- Adjacency List
- Incidence Matrix

Types of Graphs

- Simple Graphs
- Weighted Graphs
- Directed Graphs
- Connected Graphs
- Trees
- Bipartite Graphs
- Complete Graphs

Graph Algorithms

- **Minimum Vertex Cover**
  Minimum set of vertices that cover all the edges
- **Maximum Matching**
  Maximum set of disjoint edges
- **Shortest Path**
  Finding the shortest path between 2 vertices
- **Minimum Spanning Tree**
  Minimum set of edges which span the graph
- **Graph Flows**
  Maximum flow through the graph given start, end node

Computational Complexity

Complexity is a measure which evaluates the order of the count of operations, performed by a given or algorithm as a function of the size of the input data. To put this simpler, complexity is a rough approximation of the number of steps necessary to execute an algorithm.

- Asymptotic Complexity
- Complexity Notation
- Inherent Complexity of a Problem

Asymptotic Complexity

- How to analyze running time and space of algorithm
- Complexity analysis: asymptotic, empirical, others
- Different performance measures are of interest
  - Worst case (often easiest to analyze; need one ‘bad’ example)
  - Best case (often easy for same reason)
  - Average case

Complexity Notation

Let f, g be positive real valued functions defined on an unbounded subset of the real positive numbers.

- Big-O (O): $f(x) \in O(g(x))$ (upper bound) iff there exists a positive real number c and a real number x_0 such that

$$f(x) \leq c\ g(x) \text{ for all } x \geq x_0$$

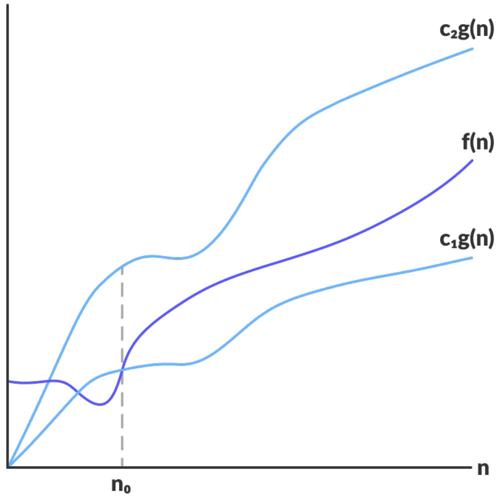
- Omega (Ω): $f(x) \in \Omega(g(x))$ (lower bound) iff there exists a positive real number c and a real number x_0 such that

$$f(x) \geq c\ g(x) \text{ for all } x \geq x_0$$

- Theta (Θ): $f(x) \in \Theta(g(x))$ (composite bound) iff there exist positive real numbers c_1, c_2 and a real number x_0 such that

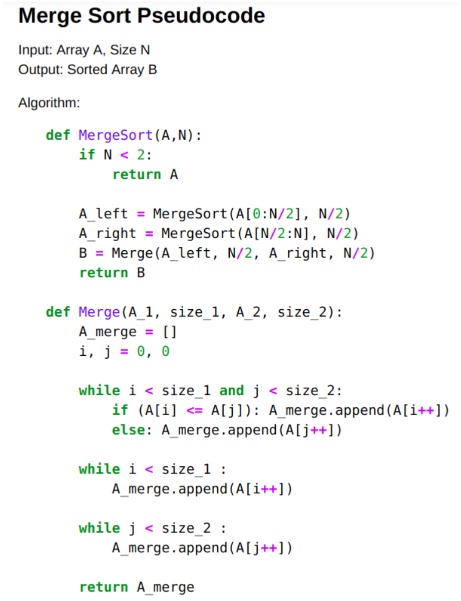
$$c_1\ g(x) \leq f(x) \leq c_2\ g(x) \text{ for all } x \geq x_0$$

Complexity Notation



$f(n) = \Theta(g(n))$

Example: Merge Sort



Merge Sort Pseudocode

Input: Array A, Size N
Output: Sorted Array B

Algorithm:

```
def MergeSort(A,N):
    if N < 2:
        return A

    A_left = MergeSort(A[0:N/2], N/2)
    A_right = MergeSort(A[N/2:N], N/2)
    B = Merge(A_left, N/2, A_right, N/2)
    return B

def Merge(A_1, size_1, A_2, size_2):
    A_merge = []
    i, j = 0, 0

    while i < size_1 and j < size_2:
        if (A[i] <= A[j]): A_merge.append(A[i++])
        else: A_merge.append(A[j++])

    while i < size_1 :
        A_merge.append(A[i++])

    while j < size_2 :
        A_merge.append(A[j++])

    return A_merge
```

Inherent Complexity of a Problem

Consider the case where we are trying to find the lower bound on the worst case of the sorting problem, where we are using comparisons to sort. Any sorting algorithm requires $\Omega(N \log N)$ comparisons.

- Given an input of N distinct numbers, choose permutations of N indices
- Algorithm independent proof using interactive approach
- Initially, possible number of answers (permutations) equals $N!$
- Each comparison reduces size of possible answer set by at most 2 (in the worst case input)
- $\log(N!) \in \Omega(N \log N)$