CS 457, Fall 2019

Drexel University, Department of Computer Science

Lecture 1

Today’s Lecture

- The structure of the course (syllabus)
- Algorithms for a variety of problems
- How to measure the efficiency of an algorithm
- Asymptotic notation

Why study algorithms?

- What is an algorithm?
  - A well-defined **computational procedure** that takes some value(s) as **input** and produces some value(s) as **output**.
  - A **sequence of computational steps** that transform the **input** into the **output**.
- Goal of an algorithm: solve a **computational problem**
  - Sorting problem:
    - Input: (32, 20, 25, 10, 18, 1, 9)
    - Output: (1, 9, 10, 18, 20, 25, 32)
  - Shortest path, travelling salesman, knapsack, maximum flow …
- Algorithm **design and analysis techniques**
  - greedy, divide & conquer, randomization, dynamic programming…

Why study algorithms?

- Given a computational problem, e.g., the sorting problem
  - A specific input, e.g., (32, 20, 25, 10, 18, 1, 9) is a **problem instance**
- When is an algorithm **correct**?
  - When it computes the desired output on every problem instance
- When is an algorithm **efficient**?
  - Time efficient (main focus of this class)
  - Space efficient
  - Amenable to Parallelism
  - Not consuming too much bandwidth
  - Simple to code up…

Insertion Sort

INSERTION_SORT (*A*)

```
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted sequence A[1 .. j - 1].
    i = j - 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i - 1
    A[i + 1] = key
```

Execution:

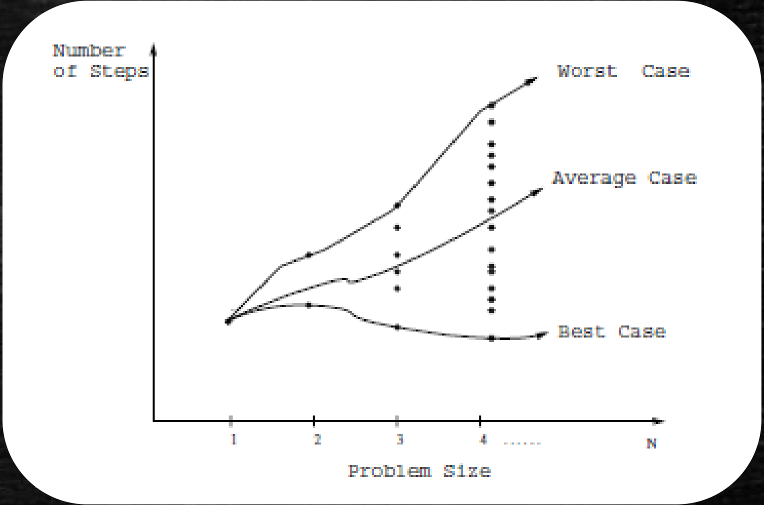
7 3 5 8 1 2
3 7
3 **5** 7
3 5 7 **8**
1 3 5 7 8
1 **2** 3 5 7 8

- What is the running time of this algorithm?
  - $O(n)$
  - $O(n \log n)$
  - $O(n^2)$
  - $O(\sqrt{n})$

How to measure the (time) efficiency

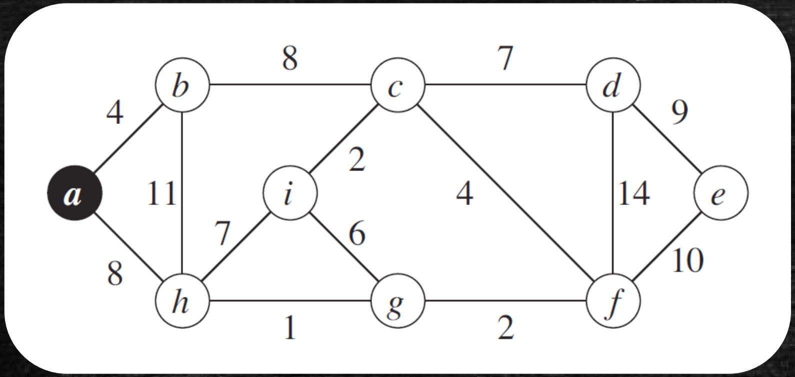
- The actual run-time of an algorithm depends on:
  - The **specs of the machine** being used
  - The **problem instance** at hand
    - Input size, e.g., number of values in a sorting instance
    - Even for a fixed input size, the run-time may vary by a lot! (e.g., sorting problem)

- How can we **compare** two algorithms?
  - Code and run experiments
  - Analyze the run-time as a function of the **input size**
    - Worst-case analysis
    - Best-case analysis
    - Average-case analysis



Minimum Spanning Tree

- The vertices of the graph below represent the 9 cities
- City a already has access to electricity, but the rest do not
- The weight of an edge is the cost of connecting the corresponding cities via cable
- What is the minimum cost cable network to achieve electrical coverage of all the cities?



- How much time does this algorithm need?
  - Depends on implementation of data structures used (graph representation, min priority queue)