| | Insertion | Selection | Bubble | Shell | Merge | Heap | Quick | Quick3 |
|---|---|---|---|---|---|---|---|---|
| Random |  |  |  |  |  |  |  |  |
| Nearly Sorted |  |  |  |  |  |  |  |  |
| Reversed |  |  |  |  |  |  |  |  |
| Few Unique |  |  |  |  |  |  |  |  |

# Comp 2540: Data Structures and Algorithms: Introduction

Prof. Jianguo Lu

January 6, 2025

# Overview

# Why we are here

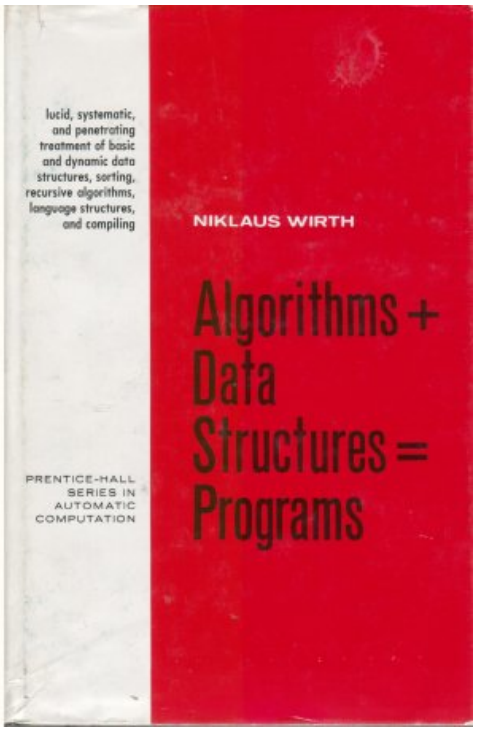
- *Data Structures and Algorithms* are fundamental.
- *Data Structures and Algorithms* are useful.
- *Data Structures and Algorithms* are fun!

# Why we are here

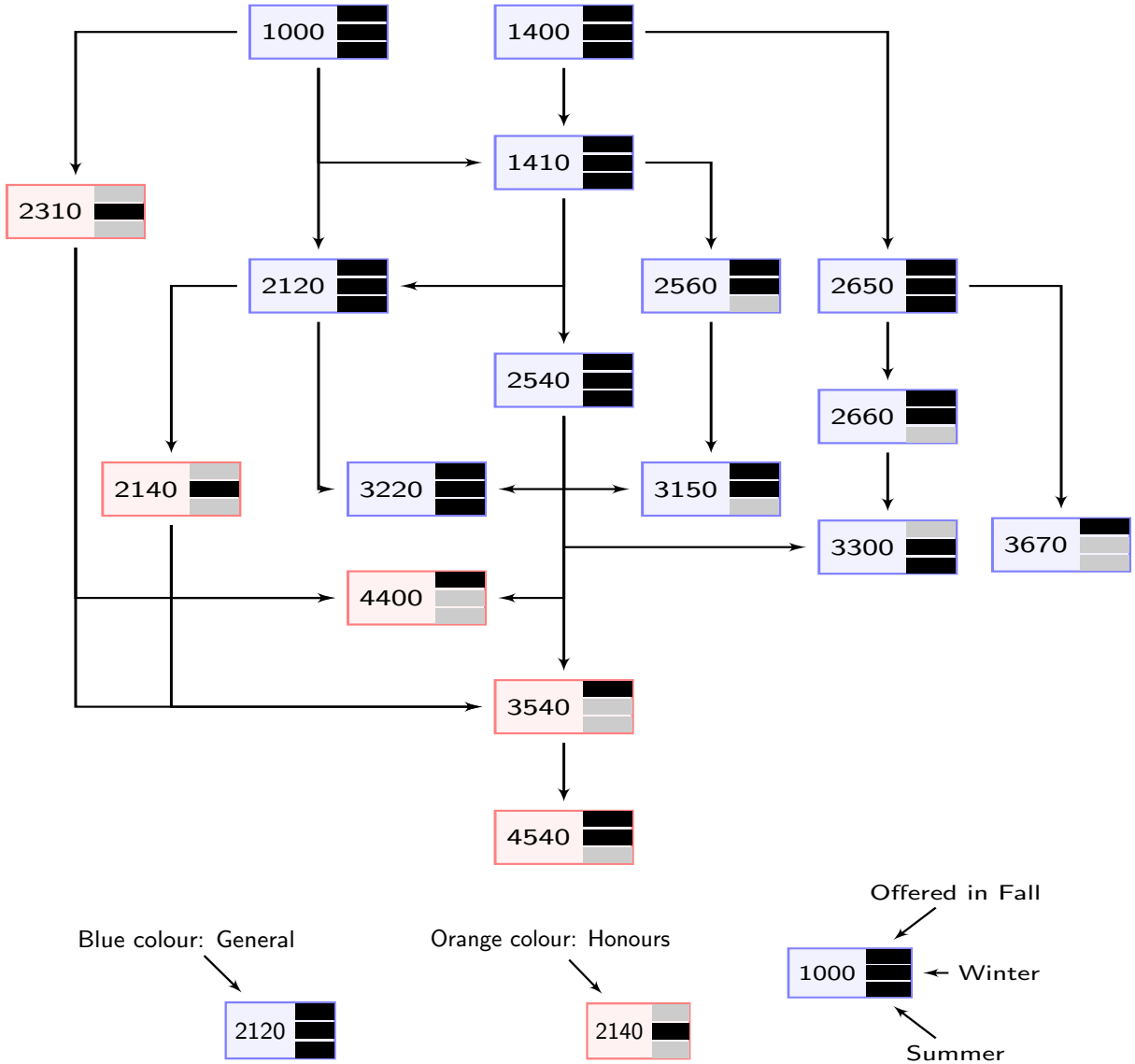
- *Data Structures and Algorithms* are fundamental.
- *Data Structures and Algorithms* are useful.
- *Data Structures and Algorithms* are fun!
- Comp-2540 is a required course.

# It is fundamental

Data structures and algorithms make up programs



# It is fundamental



# It is Useful

Must Do Coding Questions for Companies like Amazon, Microsoft, Adobe, ...

**Topic :**

- Arrays
- String
- Linked List
- Stack and Queue
- Tree and BST
- Heap
- Recursion
- Hashing
- Graph
- Greedy
- Dynamic Programming
- Divide and Conquer
- Backtracking
- Bit Magic

- Topics in coding interview preparation website
- Compare with our course content

# It is Useful

Must Do Coding Questions for Companies like Amazon, Microsoft, Adobe, ...

**Topic :**

- Arrays
- String
- Linked List
- Stack and Queue
- Tree and BST
- Heap
- Recursion
- Hashing
- Graph
- Greedy
- Dynamic Programming
- Divide and Conquer
- Backtracking
- Bit Magic

- Topics in coding interview preparation website
- Compare with our course content

| | |
|---|---|
| Week 1 | Introduction, Array and Linked List |
| Week 2, 3 | Algorithm Analysis. |
| Week 3 | Divide and conquer, merge sort. |
| Week 4 | Recursion and algorithm analysis of recursive programs |
| Week 5, 6 | Stacks and Queues, Priority queues and Heaps, Heap sort |
| Week 7 | Trees |
| Week 8 | Maps and Hash tables |
| Week 9 | More on Sorting algorithms |
| Week 10 | Binary search trees, AVL trees, Red-black trees |
| Week 11 | Graph algorithms, graph traversal, shortest path. |
| Week 12 | Dynamic programming, Review |

# It is Fun

Click to play the video for Insertion sort with Roman folk dance

# Data Structure

## Definition of Data Structure

- A data structure is a data organization, management and storage format
- The goal is to enable efficient access and modification.
- Consists of a collection of data values, and the operations that can be applied to the data

- Types of data structures

  Linear data structures : arrays, linked lists, stacks, queues
  Hierarchical structure : Trees, Binary search trees, graphs

# Abstract Data Types (ADT)

Data structures are often represented by Abstract Data Types

## Definition of ADT

- Define a data type using its operations and behavior
- Ignore (abstract away) the details of the implementation
- Examples: Stack, Queue, ...


## Stack ADT

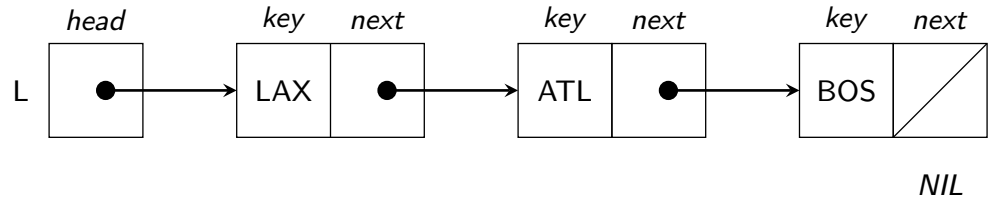
boolean empty() Tests if this stack is empty.

E peek() Looks at the object at the top of this stack without removing it.

E pop() Removes the object at the top of this stack and returns it.

E push(E item) Pushes an item onto the top of this stack.

# LinkedList ADT




## ADT for Singly LinkedList

size( ) : Returns the number of elements in the list.

isEmpty( ) : Returns true if the list is empty, and false otherwise.

first() : Returns (but does not remove) the first element in the list.

last( ) : Returns (but does not remove) the last element in the list.

addFirst(e) : Adds a new element to the front of the list.

addLast(e) : Adds a new element to the end of the list.

removeFirst( ) : Removes and returns the first element of the list.

# What is an "Algorithm"

## Definition of algorithm from Webster

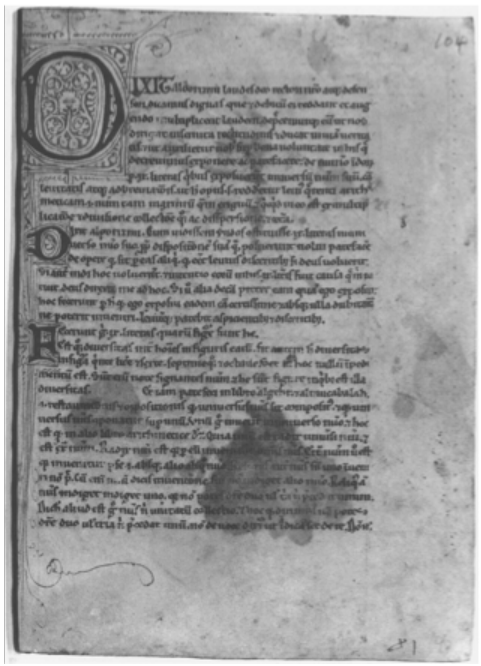
- A procedure for solving a mathematical problem in a **finite** number of steps that frequently involves **repetition** of an operation;
  - "There are several search engines, with Google, Yahoo and Bing being the biggest players. Each search engine has its own proprietary computation (called an "algorithm") that ranks websites for each keyword or combination of keywords". –Julie Brinton
- A step-by-step procedure for solving a problem or accomplishing some end.
  - "... sometimes you solve a problem by coming up with an algorithm of some kind. But sometimes you solve a problem in a very ad hoc sort of way". –William H. Huggins

## Definition of Algorithm from Wikipedia

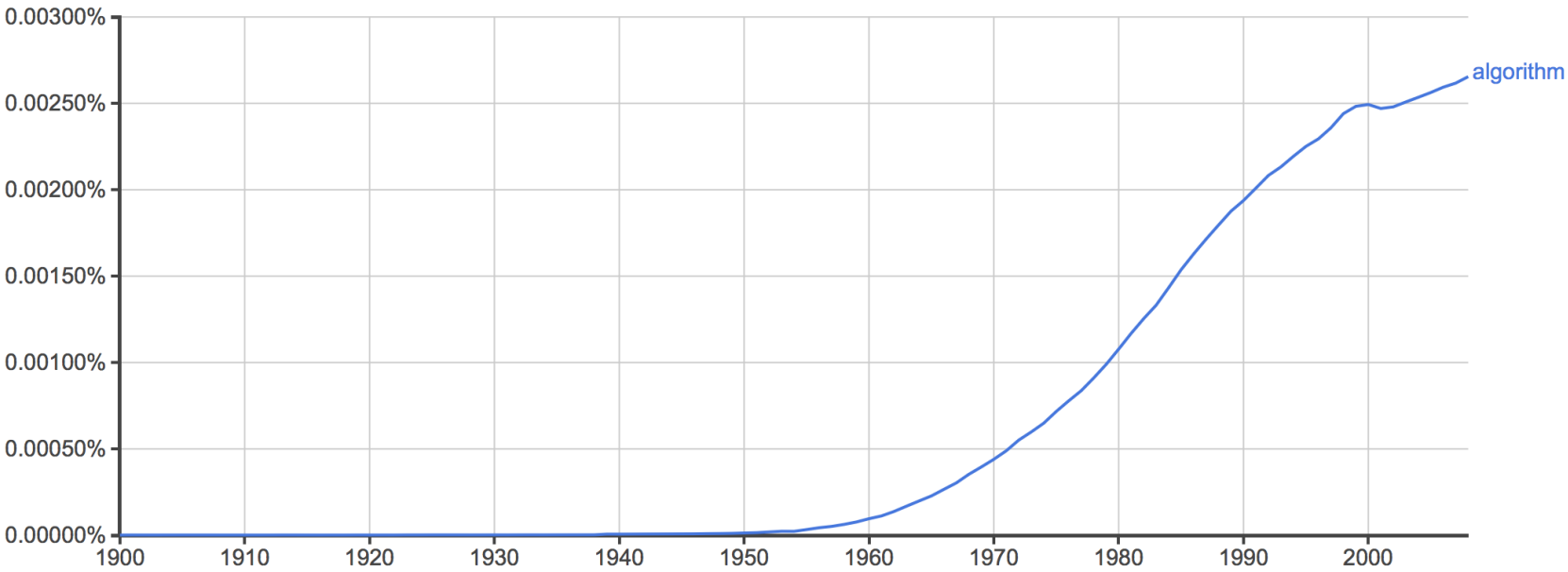
In mathematics and computer science, an algorithm is an **unambiguous** specification of how to solve a class of problems. Algorithms can perform calculation, data processing and automated reasoning tasks. — Wikipedia

# Etymology of "Algorithm"

- The word is from the Arabic name Al-Khowarizmi, a 9th century mathematician
- He wrote a book about how to multiply with Arabic numerals.
- "Algorisme" [old French] referred to just the Arabic number system, but eventually it came to mean "Algorithm" as we know today.



# “Algorithm” has become increasingly popular



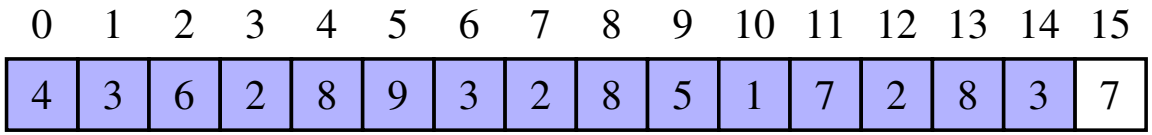

From Google Books NGram Viewer

# Algorithms vs. programs

- A computer program is an implementation of an algorithm on a computer
- A program is language dependent
- Algorithms are language in-dependent, often described using pseudo-code.
- Remember the classic book:

$$\textit{Algorithms} + \textit{data structures} = \textit{programs}$$

# Problem Specifications vs. Algorithms




## The sorting problem

- Input: A sequence of numbers A[1], A[2], ..., A[n]
- Output: A permutation(reordering) of A such that A is sorted. i.e.,

$$\text{for all i, j, if } i > j, \quad A[i] > A[j] \tag{1}$$

- This is the problem specification
- What are the algorithms?

# Selection Sorting Algorithm

**Algorithm 1:** Selection Sort

**Input:** Array A of length n
**Output:** Sorted A

```
1 for int i =0; i < n-1; i++ do
2     min= minimal element in
        array[i+1:n];
3     swap array[i] with min;
```

| | | | | | | |
|---|---|---|---|---|---|---|
| 4 | 6 | 7 | 1 | 2 | 5 | 3 |
| 1 | 6 | 7 | 4 | 2 | 5 | 3 |

# Selection Sorting Algorithm

**Algorithm 2:** Selection Sort

**Input:** Array A of length n
**Output:** Sorted A

```
1 for int i =0; i < n-1; i++ do
2     min= minimal element in
        array[i+1:n];
3     swap array[i] with min;
```

| | | | | | | |
|---|---|---|---|---|---|---|
| 4 | 6 | 7 | 1 | 2 | 5 | 3 |
| 1 | 6 | 7 | 4 | 2 | 5 | 3 |
| 1 | 2 | 7 | 4 | 6 | 5 | 3 |

# Selection Sorting Algorithm

**Algorithm 3:** Selection Sort

**Input:** Array A of length n
**Output:** Sorted A

```
1 for int i =0; i < n-1; i++ do
2     min= minimal element in
        array[i+1:n];
3     swap array[i] with min;
```

| | | | | | | |
|---|---|---|---|---|---|---|
| 4 | 6 | 7 | 1 | 2 | 5 | 3 |
| 1 | 6 | 7 | 4 | 2 | 5 | 3 |
| 1 | 2 | 7 | 4 | 6 | 5 | 3 |
| 1 | 2 | 3 | 4 | 6 | 5 | 7 |

# Selection Sorting Algorithm

**Algorithm 4:** Selection Sort

**Input:** Array A of length n
**Output:** Sorted A

```
1 for int i =0; i < n-1; i++ do
2     min= minimal element in
        array[i+1:n];
3     swap array[i] with min;
```

| | | | | | | |
|---|---|---|---|---|---|---|
| 4 | 6 | 7 | 1 | 2 | 5 | 3 |
| 1 | 6 | 7 | 4 | 2 | 5 | 3 |
| 1 | 2 | 7 | 4 | 6 | 5 | 3 |
| 1 | 2 | 3 | 4 | 6 | 5 | 7 |
| 1 | 2 | 3 | 4 | 6 | 5 | 7 |

# Selection Sorting Algorithm

**Algorithm 5:** Selection Sort

**Input:** Array A of length n
**Output:** Sorted A

```
1 for int i =0; i < n-1; i++ do
2     min= minimal element in
        array[i+1:n];
3     swap array[i] with min;
```

| | | | | | | |
|---|---|---|---|---|---|---|
| 4 | 6 | 7 | 1 | 2 | 5 | 3 |
| 1 | 6 | 7 | 4 | 2 | 5 | 3 |
| 1 | 2 | 7 | 4 | 6 | 5 | 3 |
| 1 | 2 | 3 | 4 | 6 | 5 | 7 |
| 1 | 2 | 3 | 4 | 6 | 5 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |

# Selection Sorting Algorithm

**Algorithm 6:** Selection Sort

**Input:** Array A of length n
**Output:** Sorted A

```
1 for int i =0; i < n-1; i++ do
2     min= minimal element in
        array[i+1:n];
3     swap array[i] with min;
```

| | | | | | | |
|---|---|---|---|---|---|---|
| 4 | 6 | 7 | 1 | 2 | 5 | 3 |
| 1 | 6 | 7 | 4 | 2 | 5 | 3 |
| 1 | 2 | 7 | 4 | 6 | 5 | 3 |
| 1 | 2 | 3 | 4 | 6 | 5 | 7 |
| 1 | 2 | 3 | 4 | 6 | 5 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |

# Selection Sorting Algorithm

**Algorithm 7:** Selection Sort

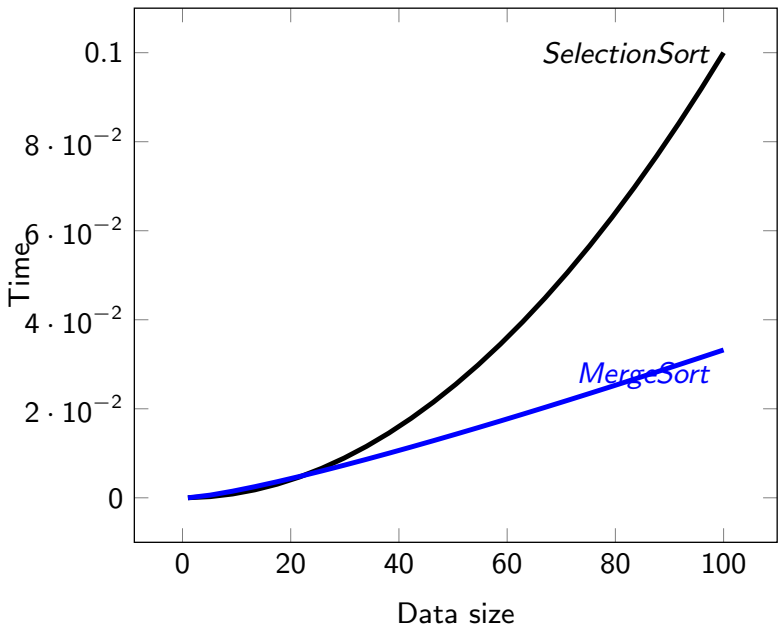
**Input:** Array A of length n
**Output:** Sorted A

```
1 for int i =0; i < n-1; i++ do
2     min= minimal element in
        array[i+1:n];
3     swap array[i] with min;
```

| | | | | | | |
|---|---|---|---|---|---|---|
| 4 | 6 | 7 | 1 | 2 | 5 | 3 |
| 1 | 6 | 7 | 4 | 2 | 5 | 3 |
| 1 | 2 | 7 | 4 | 6 | 5 | 3 |
| 1 | 2 | 3 | 4 | 6 | 5 | 7 |
| 1 | 2 | 3 | 4 | 6 | 5 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |

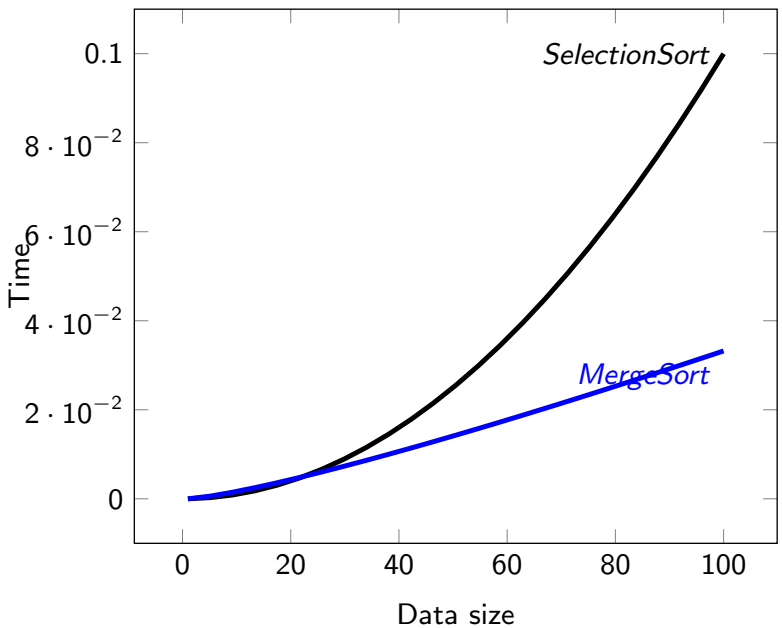
# Selection sort vs. Merge Sort

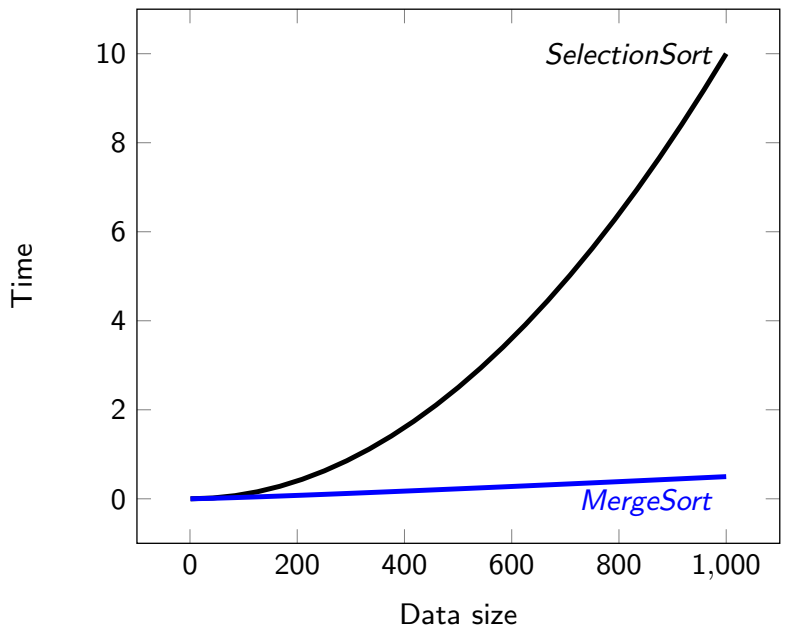
- Selection sort is slow compared with merge sort
- The difference is not big when the data size is small
- Makes a big difference when the data size is big



# Selection sort vs. Merge Sort

- Selection sort is slow compared with merge sort
- The difference is not big when the data size is small
- Makes a big difference when the data size is big



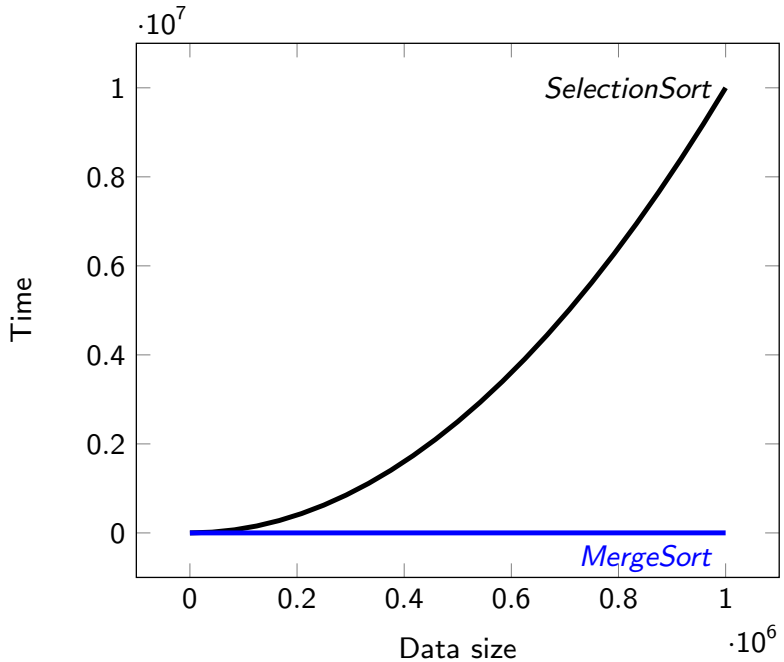



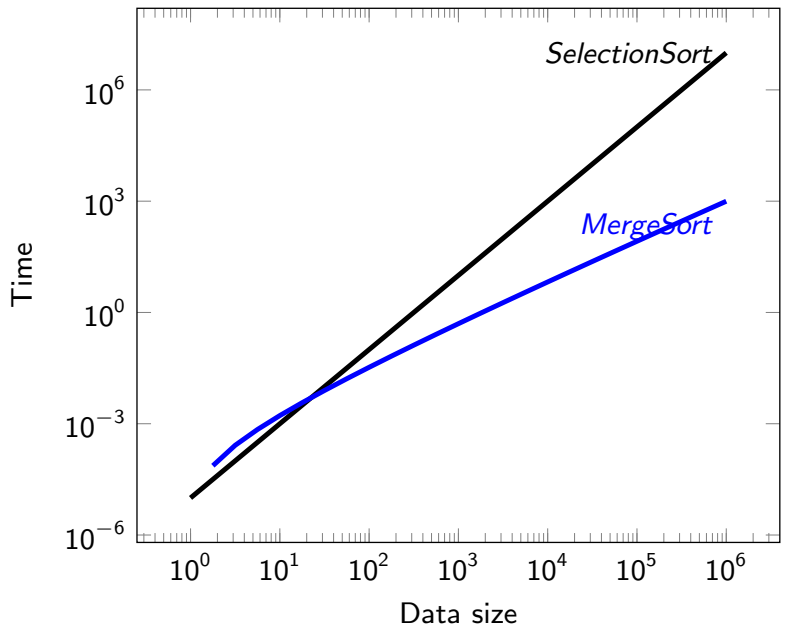
# Selection sort vs. Merge Sort

- Comparison when sort $10^6$ items

$$mergeSort < 1000 second \approx 0.27 hours$$

$$SelectionSort > 10^6 second \approx 277 hours > 10 days!$$



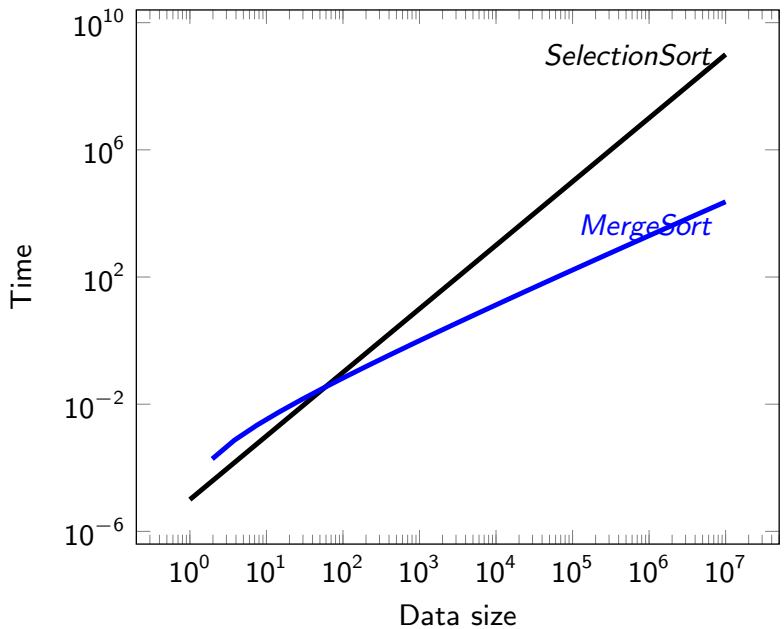



# Selection sort vs. Merge Sort

- Comparison when sort $10^7$ items

$$mergeSort < 11,000 \ second \approx 3 \ hours$$

$$SelectionSort > 10^9 \ second \approx 277,000 \ hours > 11,541 \ days \approx 31 \ years!$$

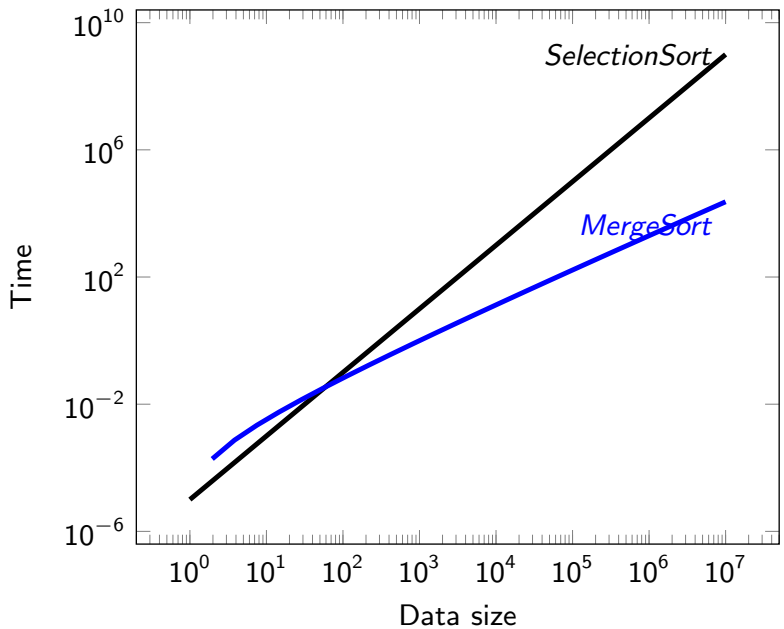


# Selection sort vs. Merge Sort

- Comparison when sort $10^7$ items

$$mergeSort < 11,000 \ second \approx 3 \ hours$$

$$SelectionSort > 10^9 \ second \approx 277,000 \ hours > 11,541 \ days \approx 31 \ years!$$



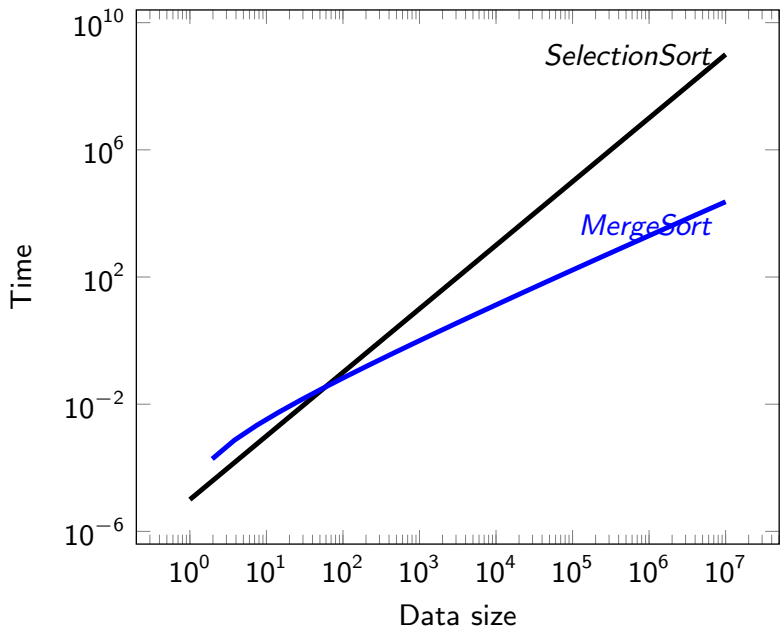

- How do we know that it will take 31 years?

# Selection sort vs. Merge Sort

- Comparison when sort $10^7$ items

$$mergeSort < 11,000 \ second \approx 3 \ hours$$

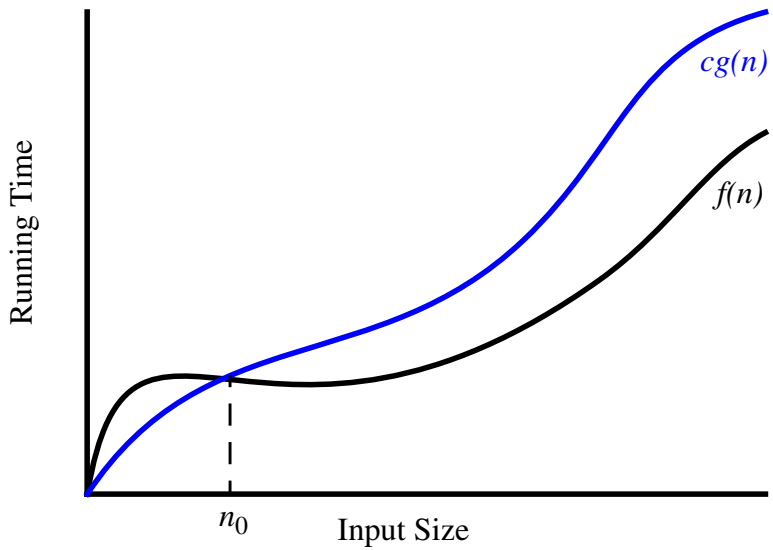
$$SelectionSort > 10^9 \ second \approx 277,000 \ hours > 11,541 \ days \approx 31 \ years!$$




- How do we know that it will take 31 years?
- We can **prove** using techniques in algorithm analysis.

## Definition: O(g(n))

$$O((g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0\} \quad (2)$$




$\Theta(n^2)$, $\Omega(n^2)$

# Algorithm analyses

- Order of growth of the running time
- gives a simple characterization of the algorithm's efficiency
- allows us to compare the relative performance of alternative algorithms
- Usually, an algorithm that is asymptotically more efficient will be the best choice for all but very small inputs

# We will learn many sorting algorithms


You need to use Acrobat Reader to view the animation. It can be downloaded here.

# Why sorting and searching algorithms?

- Direct applications: e.g.,
  - Google search, the best match(es) for a query
- Indirect applications: e.g., it is easier to get the top matches if data items are sorted.
  - e.g., word count example in our assignment

# And We will learn many data structures

- Heaps, Hash Table, Binary Search Trees, AVL tree, Graph, ...
- Used in sorting, searching and other algorithms

# And We will learn many data structures

- Heaps, Hash Table, Binary Search Trees, AVL tree, Graph, ...
- Used in sorting, searching and other algorithms

Why so many data structures?

# Example: How to find a value from an array quickly?

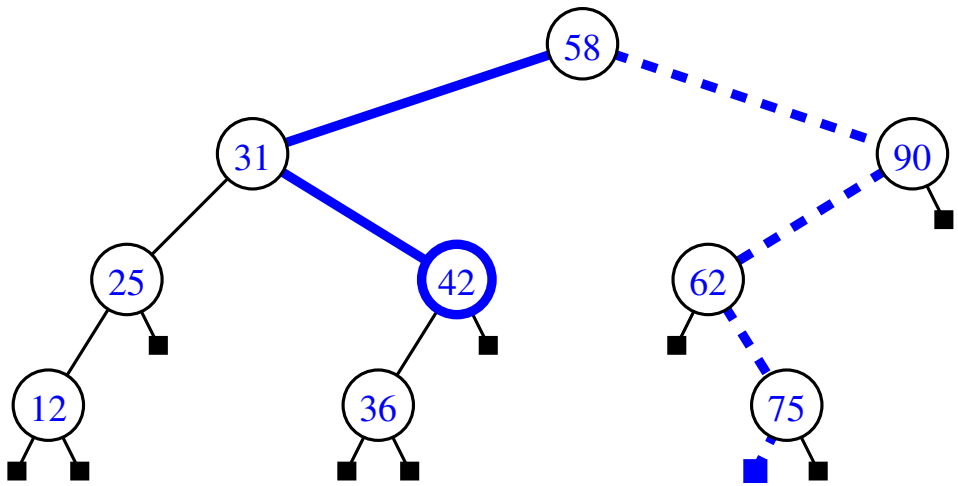
- Given a list of data items:

  `12, 25, 31, 58, 42, 36, 62, 75, 90.`

- Whether 42 is in the list?
- What is your algorithm?
- Can be sub-linear?
- Can be constant time?
- How can Google find your query result instantly?

# Binary search tree

- Elements stored in the left subtree are smaller.
- Elements stored in the right subtree are bigger.



# Heap

A (binary) heap is a binary tree that satisfies the following two properties:

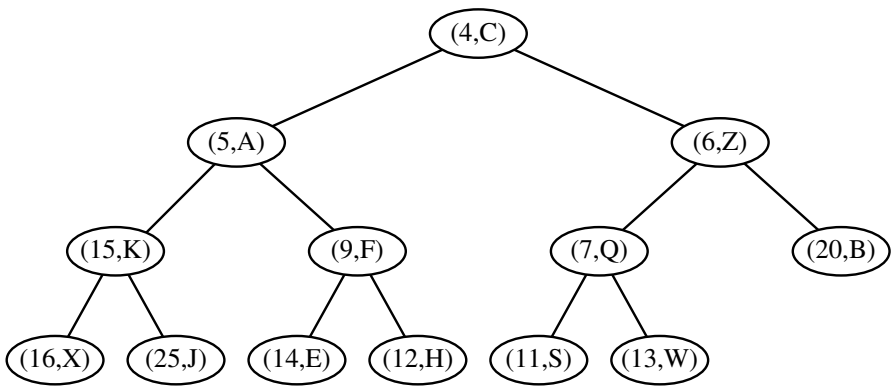
## Heap-Order Property

- Every node other than the root is no less than its parents

## Complete Binary Tree Property

A heap T with height h is a complete binary tree:

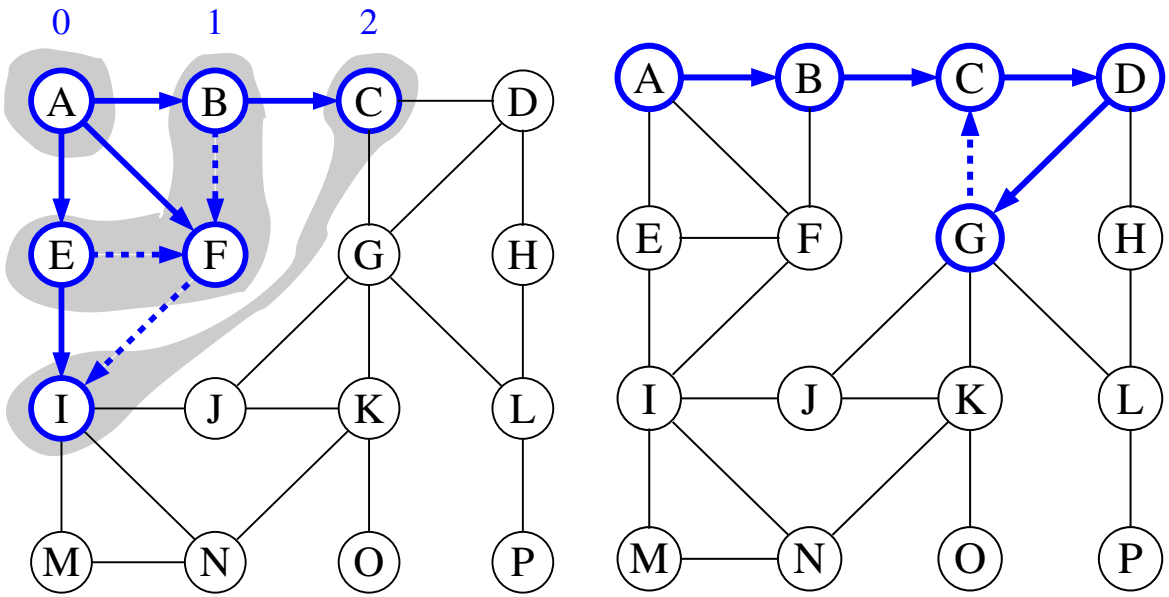
- levels 0,1, 2, . . . ,h-1 of T have the maximal number of nodes possible
- remaining nodes at level h reside in the leftmost possible positions.



# Graph algorithms

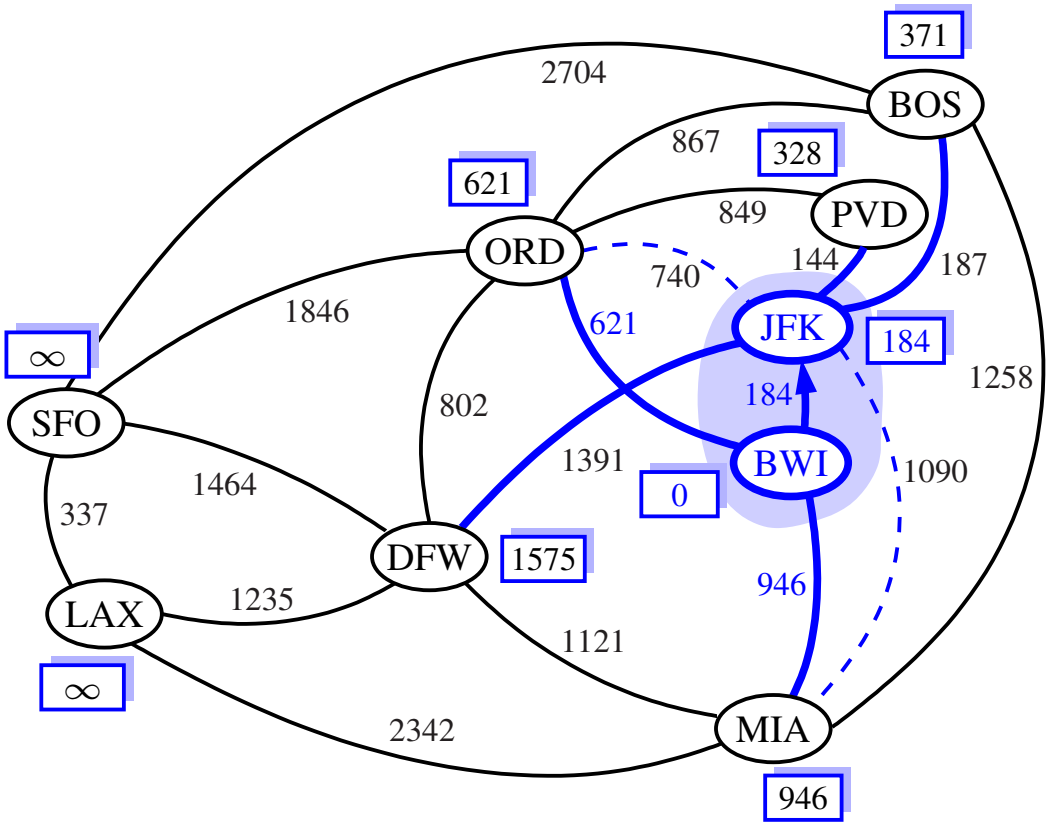
- Graph traversals, Depth-First Search, Breadth-First Search
- Shortest path

# Breadth first search




- Applications: e.g.,web crawling by Google
- Compare with Depth first search

# Shortest path



# How do we design algorithm

Divide and Conquer : breaking a problem into smaller ones. e..g. used in merge sort and quick sort.

Dynamic programming : Dijkstra's algorithm for the shortest path problem

Greedy algorithm : often used in optimization problems, like in Travelling Salesman Problem

# Labs

- Labs: Many sections. Monday and Wednesday
- Course web page : At BrightSpace website.
- Prerequisites : Comp-1000, Comp-1400, Comp-1410.
- Comp-2120. The textbook uses java.

# Evaluation Scheme

| | Time | Weight (%) |
|---|---|---|
| Lab Assignments | Every other week | 20 |
| Midterm 1 | Jan 29 | 15 |
| Midterm 2 | March 5 | 20 |
| Final Exam | TBA | 45 |
| **Total** | | 100 % |

Exam dates :

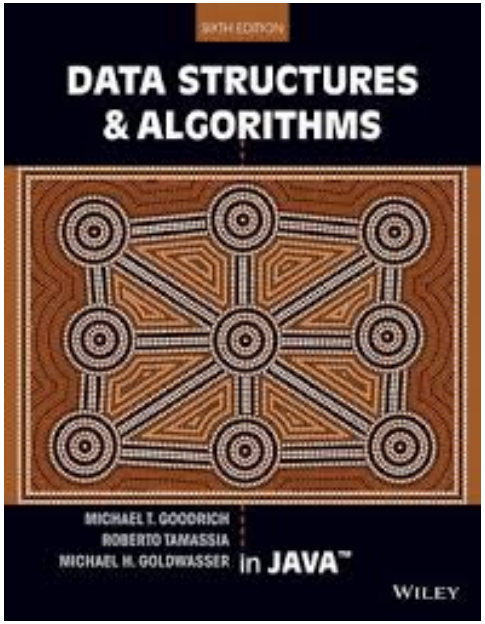
- Midterms: class time

Lab assignments due dates : Every other week.

# Labs

- Lab assignments will be submitted and marked during the labs.
- There will be 10 lab sessions. In each odd-numbered lab session (1, 3, 5, 7 and 9), the lab assignment will be explained, and students will start working on it.
- Lab assignments will be submitted in even-numbered lab sessions (2, 4, 6, 8 and 10). Each lab assignment is worth 8% of the course grade. If you finish earlier, namely in the corresponding odd-numbered session, you can submit the lab at that time and do not have to come to the next session (the corresponding even-numbered lab).
- The preferred programming language is Java. However, you can use a programming language of your choice, provided you complete the required tasks correctly.

# Text book




Data Structures and Algorithms in Java, 6th Edition Michael T. Goodrich, Roberto Tamassia, Michael H. Goldwasser ISBN: 978-1-118-77133-4. Jan 2014. 738 pages

# Take aways

- This course is about how to write good (efficient) programs/algorithms
- It is fundamental, useful, and fun (yet to be confirmed)
- What is a problem specification. (hint: sorting example)
- What is an algorithm. Selection sort example
- What is data structure (ADT). Binary search tree. Heap.
- Why is efficiency important?
- How do we know whether an algorithm is efficient: algorithm analysis.
- How to design efficient algorithms: algorithm design paradigms.
- Should I take this course?
- How can I score high in this course?