# Chapter 17
# Searching and Sorting Algorithms

FORDHAM

CISC 5004 - Programming C++
Sam Kamens
Fall, 2022

# Searching and algorithms

- Algorithm
  - Sequence of steps for accomplishing a task
- Linear search algorithm
  - Starts from the beginning of the list and checks each element
  - Example
- Runtime
  - Time it takes for an algorithm to execute
  - Usually expressed in terms of the relationship to the size of the input
  - For linear search, with a list of size N, the worst case is N comparisons
  - “On the order of” N ⇒ O(N)

# Binary Search

- Requires the list to be sorted
- Illustration
- Algorithm
- Code
- Runtime
  - Search space is reduced by half at each iteration
  - Runtime is “logarithmic”: $O(\log_2 N)$

# Big O notation

- Big O notation
  - Mathematical way of describing how a function generally behaves
  - In relation to input size
- All functions with the same growth rate use the same Big O notation
  - Determining Big O notation
  - Composite functions
- Efficiency is most critical for large input sizes
  - Illustration
  - Graph
- Common categories

# Algorithm Analysis

- Worst-case analysis
  - Focus on the worst case
    - Can also evaluate best-case and average-case, but only if the properties of the data are known
  - Example
- Constant Time operations
  - Constant number of constant time operations is O(1)
- Nested Loops
  - Simplifies to a power of N for each level of nesting (2 levels ⇒ $N^2$, 3 levels ⇒ $n^3$, …)

# Sorting: Introduction

- Convert a list of elements into ascending (or descending) order
- Broken down into individual element swaps
  - Example

# Selection Sort

- Input treated as two parts (sorted part and unsorted part)
  - Algorithm selects which value to move from the unsorted to the end of the sorted part
  - Walkthrough
- What is the runtime?

# Insertion Sort

- Sorted and unsorted part
- Insert next value into the correct location in the sorted part
- Walkthrough
- Complexity?
- For **nearly-sorted lists**, runtime is O(N)
  - Illustration

# Quicksort

- Partition the input into low and high parts, and recursively sort each part
  - Pivot: Any value within the array
  - Partitioning
  - Sorting
- Complexity
  - Partitioning divides the input in half, half again, etc. (log N)
  - Each partition does at most N comparisons (N)
  - O(N logN)
- Worst case: all partitions are completely unbalanced
  - $O(N^2)$

# Merge Sort

- Divides the list in two halves
- Recursively sort each half
- Merge sorted halves
- Illustration
- Code
- Complexity: O(N logN)
- Requires additional space of O(N)

# C++ Examples

- Selection Sort
- Insertion Sort