Design & Analysis of Algorithms

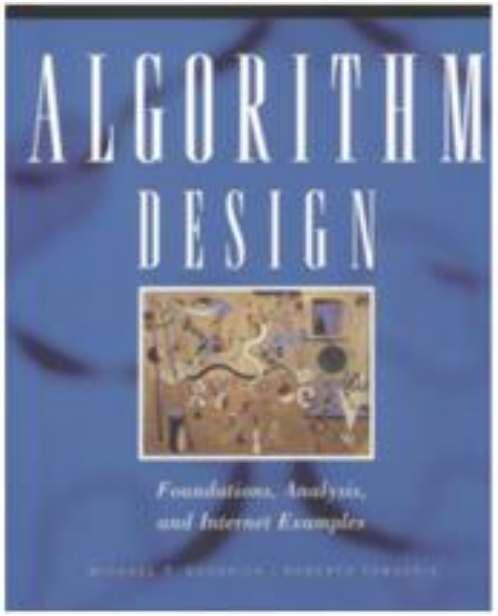
- **Course Website**:
  - moodle.uoz.edu.krd

- **Instructor**: Abdulhakeem Othman Mohammed
  - Office Hours: Monday 10:30–12:30, and by appointment
  - Email: a.mohammed@uoz.edu.krd
  - Questions? (Piazza)

Books

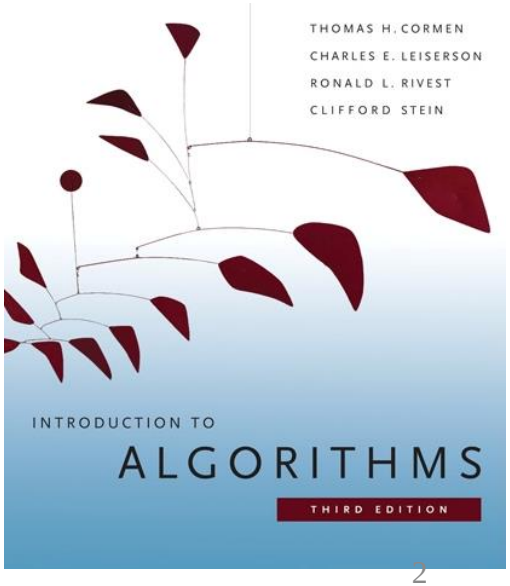
- **Textbook**:

Algorithm Design: Foundations, Analysis, and Internet Examples, by Michael T. Goodrich and Roberto Tamassia, 1st edition, Wiley, 2001



- **An excellent reference**:

Introduction to Algorithms, 3rd Edition, by T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, MIT, 2009.



Course Requirements

- **Homework 10%**
  - Good preparation for exams
  - Homework is weighted based on different problems
- **Quiz 10%**
  - Good preparation for exams
- **Midterm Exams 40% (Two midterms, each 20%)**
  - (closed book, no calculators, one sheet (both sides) of notes)
  - During Tutorial Time.
- **Final Exam 40%**
  - (closed book, no calculators)
- **Participation 5% (Extra Credits)**
  - Engagement in class and on Piazza

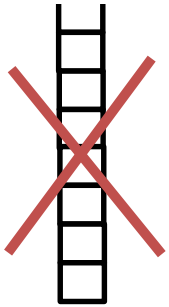
Late Policy

- Late Policy
  - Homework must be turned in by the end of class on the due-date.
  - Unexcused late homework is not accepted.
  - Missed exams and missed homework are only excused if absence was essential and can be fully documented.

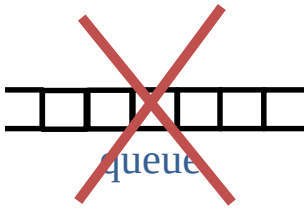
Tools you need

Example: Design an inventory system which can quickly find an item.

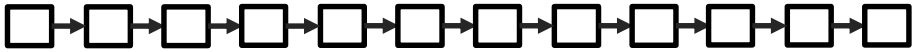
- What data structure to use?



stack

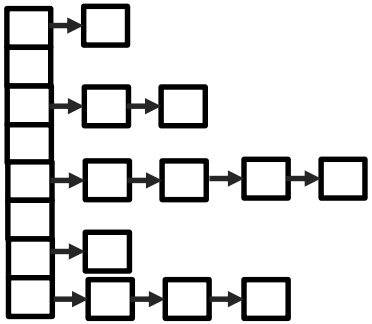


queue

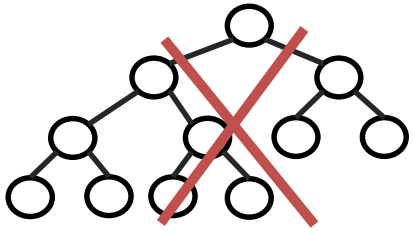


linked list

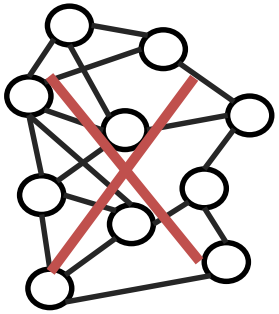
array



hash table



tree



graph

Tools you need

Example: Design an inventory system which can quickly find an item.

- What approach to take?

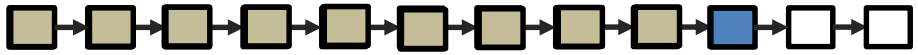
Brute force

Dynamic programming

Divide and conquer

Greedy method

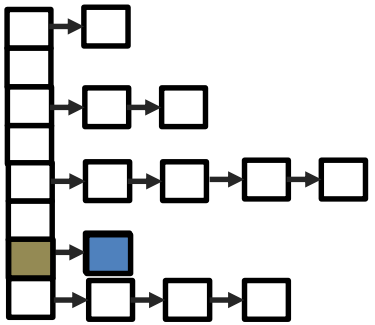
Prune and search



linked list

array

- Are there any existing algorithms that could be used/modified?



hash table

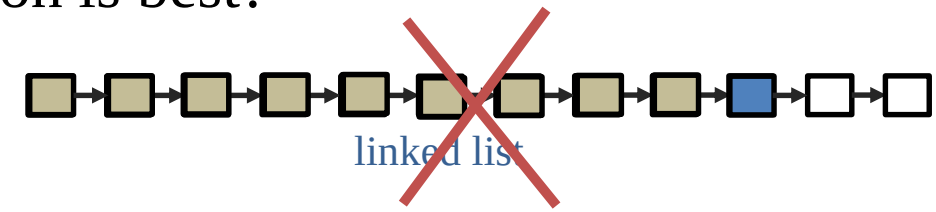
Tools you need

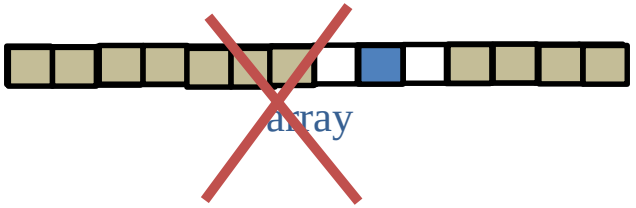
Example: Design an inventory system which can quickly find an item.

- How to determine which solution is best?
- Does it work as required?

  Rationalization

  Proof of correctness



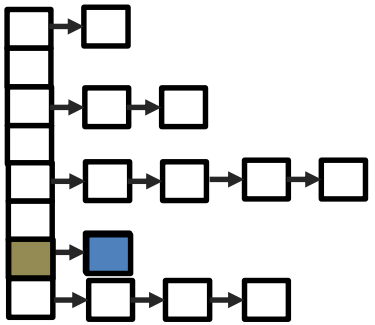


- How much memory is required? How long does it take?

  Big-oh notation

  Amortization

  Complexity analysis



Design & Analysis of Algorithms

- How to evaluate algorithms (correctness, complexity)
  - Notations and abstractions for describing algorithms
- Advanced data structures and their analysis
- Fundamental techniques to solve the vast array of unfamiliar problems that arise in a rapidly changing field
  - Up to date grasp of fundamental problems and solutions
  - Approaches to solve
- To think algorithmically like a ‘real’ computer scientist

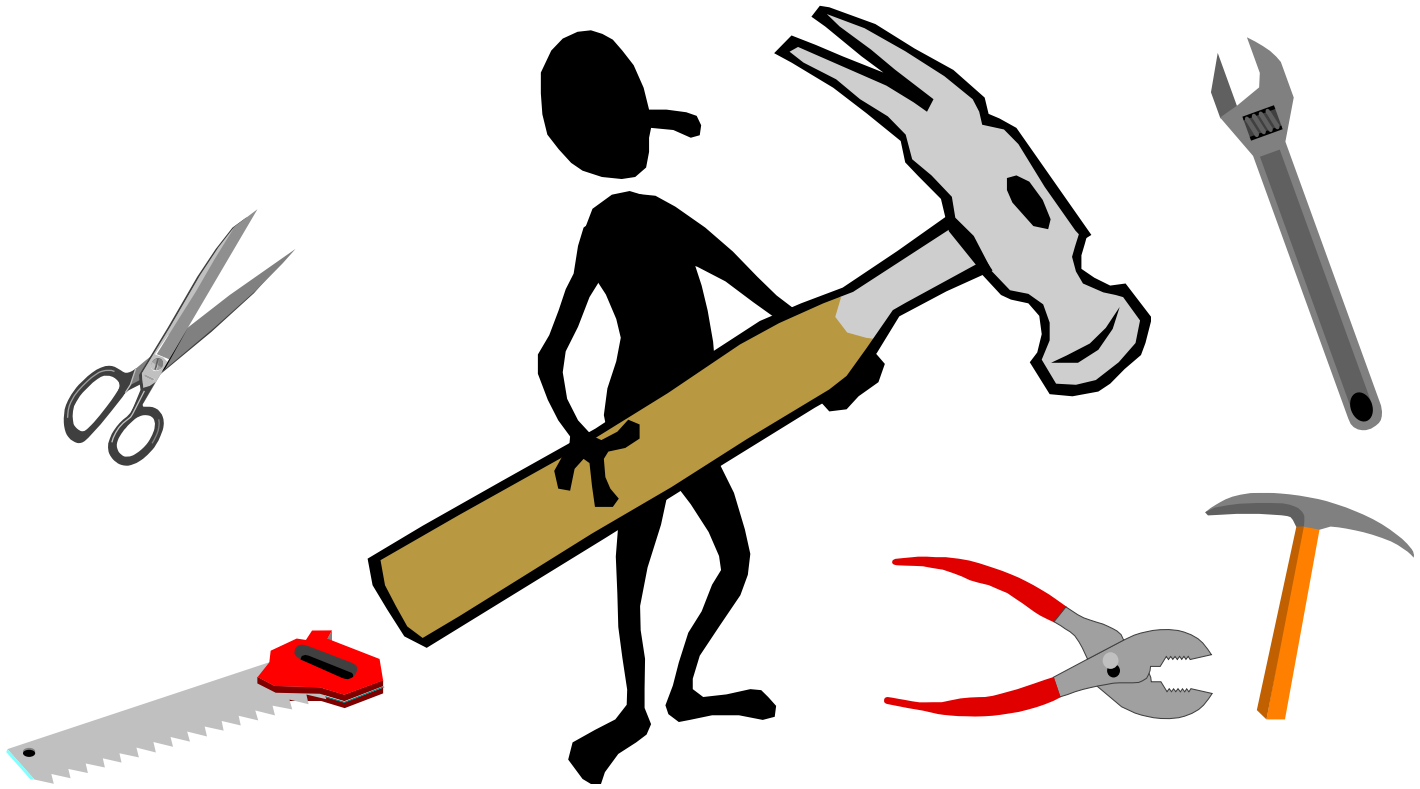
Course Content

- A list of algorithms
  - Learn the code
  - Trace them until you are convinced that they work
  - Implement them.

```
class InsertionSortAlgorithm extends SortAlgorithm
{
    void sort(int a[]) throws Exception {
        for (int i = 1; i < a.length; i++) {
            int j = i;
            int B = a[i];
            while ((j > 0) && (a[j-1] > B)) {
                a[j] = a[j-1];
                j--;  }
            a[j] = B;
            }
}
```

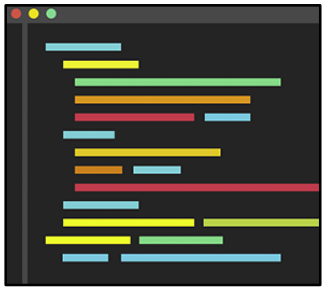
Course Content

- A survey of algorithmic design techniques
- Abstract thinking
- How to develop new algorithms for any problem that may arise



Start with some math

Time complexity as a function



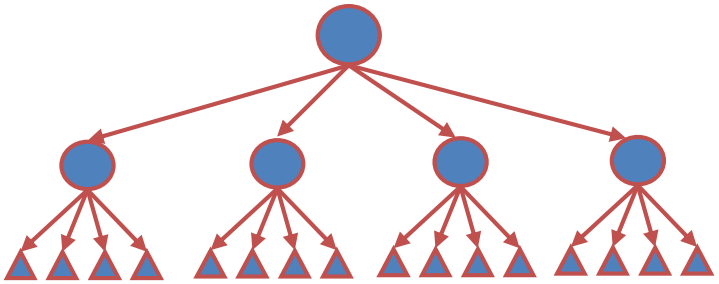
$t(n) = \Theta(n^2)$

Counting primitive operations

- Sequences and summations
- Linear functions
- Logarithmic and exponential functions

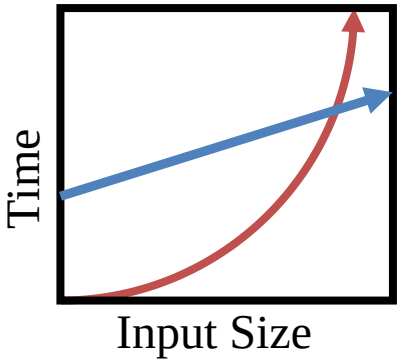
$$a + ar + ar^2 + ar^3 + \cdots + ar^{n-1} = \sum_{k=0}^{n-1} ar^k = a\left(\frac{1-r^n}{1-r}\right)$$

Recurrence Relations

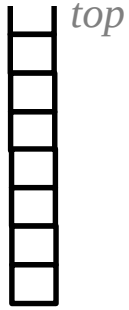


$T(n) = a\,T(n/b) + f(n)$

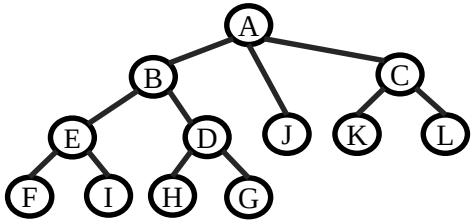
Classifying functions



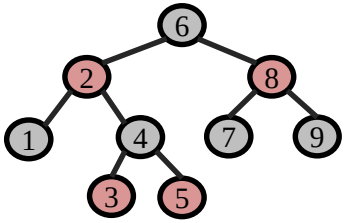
Data Structures



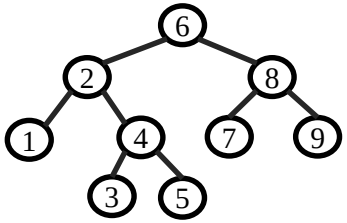
stack



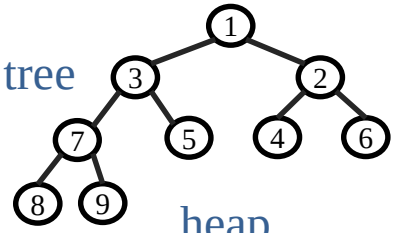
tree



red black tree

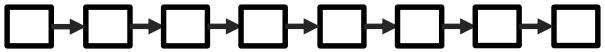


binary search tree

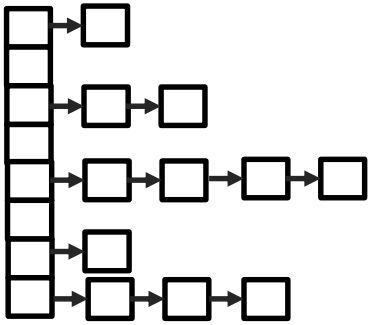


heap
&
priority queues

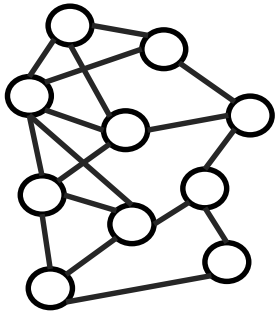
queue



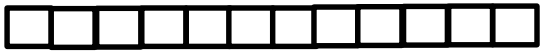
linked list



hash table
&
dictionaries

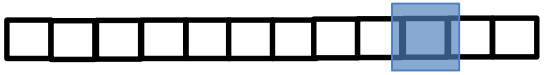


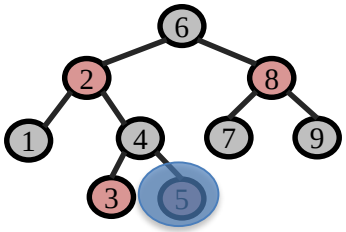
graph

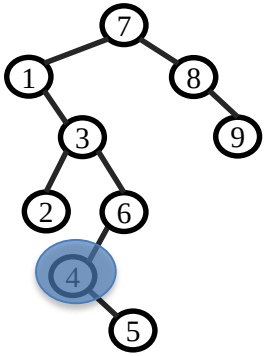


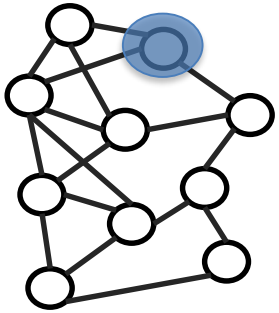
vector

Searching & Sorting









insertion sort

selection sort

heap sort

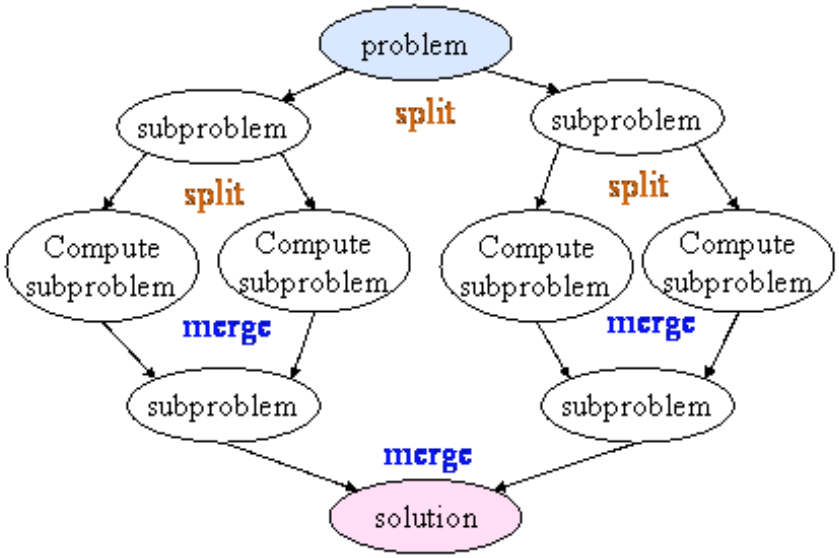
merge sort

quick sort

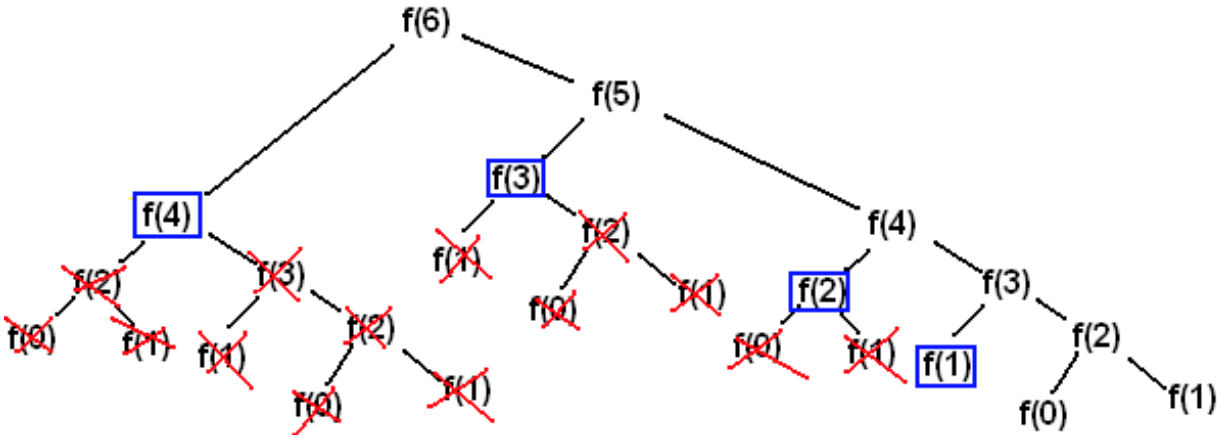
Fundamental Techniques

Greedy Algorithms

Divide and Conquer

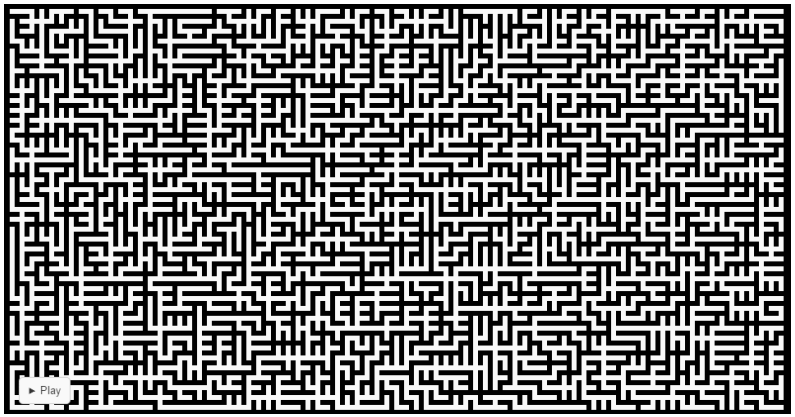


Dynamic Programming

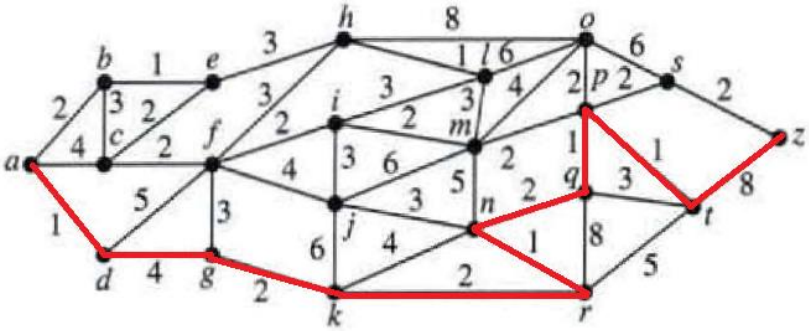


Graphs & Graph Algorithms

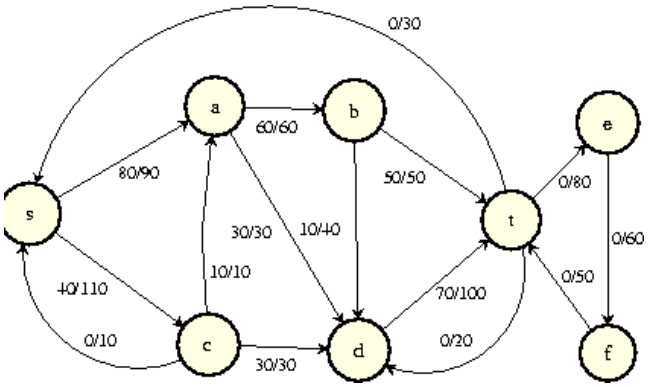
Graph search



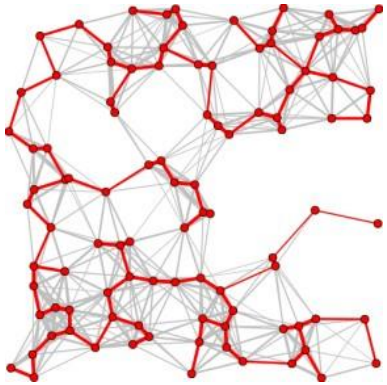
Shortest path



Network flow



Minimum Spanning Tree



Useful Learning Techniques

- You are expected to read ahead (before the lecture)
  - This will facilitate more productive discussion during class
- Guess at potential algorithms for solving a problem
  - Look for input instances where your algorithm is wrong
- Practice explaining
  - You'll be tested on your ability to explain material
- Ask questions
  - Why is it done this way and not that way?

Design an Algorithm

Given two integer arrays **A** and **B, i**s there an integer *i* which is in both arrays?

Algorithm 1

```
For Each a ∈ A
    For Each b ∈ B
        If a = b Then
            Return "Yes"
Return "No"
```

Algorithm 2

Sort A and B.

Set i := 0 and j := 0.

While i < |A| and j < |B|

    If A[i] = B[j] **Then**

        Return “Yes”

    Else If A[i] < B[j] **Then**

        Set i := i + 1.

    Else If A[i] > B[j] **Then**

        Set j := j + 1.

Return “No”

Question

Which algorithm is better and why?