CS214 – Data Structures
Lecture 01: A Course Overview

Instructor: Cherry Ahmed

c.ahmed@fci-cu.edu.eg

Slides by

Ahmed Kamal, PhD

Mohamed El-Ramly, PhD

Basheer youssef PhD

Lecture 1 Outline

1. Course Objectives
2. Course Administration
3. Introduction to Data Structures
4. Revision

Course Rationale

- So far, you have acquired proficiency in programming.
- This course starts to transform you from a programmer to a computer scientist.
- CS scientist must make efficient use of computational resources.
- This course will teach you about different ways of organizing the data and performing operations on them to facilitate efficient use of resources.

Course Rationale

- It will also teach you to analyze the efficiency of your program in a mathematical manner.
- This is critical to your becoming a good software developer later.

Course Objectives

- Understand the basic ADT and their characteristics
- Learn how to decide on the suitable data structure for an application.
- Implement basic data structures in C++
- Use existing data structures effectively in applications.
- Learn algorithm complexity and efficiency

Course Objectives

- This IS NOT a course in object-oriented programming!

Course Objectives

- It is efficient to:
  - minimize the run time of code.
  - minimize the memory / storage needs of code.
  - recognize that there may be a trade-off between speed and memory requirements.
  - writing a program of higher quality.
  - re-use code, instead of re-writing code.

Why do we study Data Structures?

- Obviously, the best organization of data in the main memory optimizes the memory usage and improves the performance of the specified operations on data
- Professional programmers are the ones who know how to select the appropriate data structures to organize their data in the main memory!

2. Course Administration

Basic Course Information

- **Course Code: CS214**
- **Course Name: Data Structures**
- **Course Credit: 3 credits**
- **Instructor:**

Dr. Cherry Ahmed Amir
c.ahmed@fci-cu.edu.eg

Course on Google Classroom

- Class Name:

  2023-SCS214-Data Structures

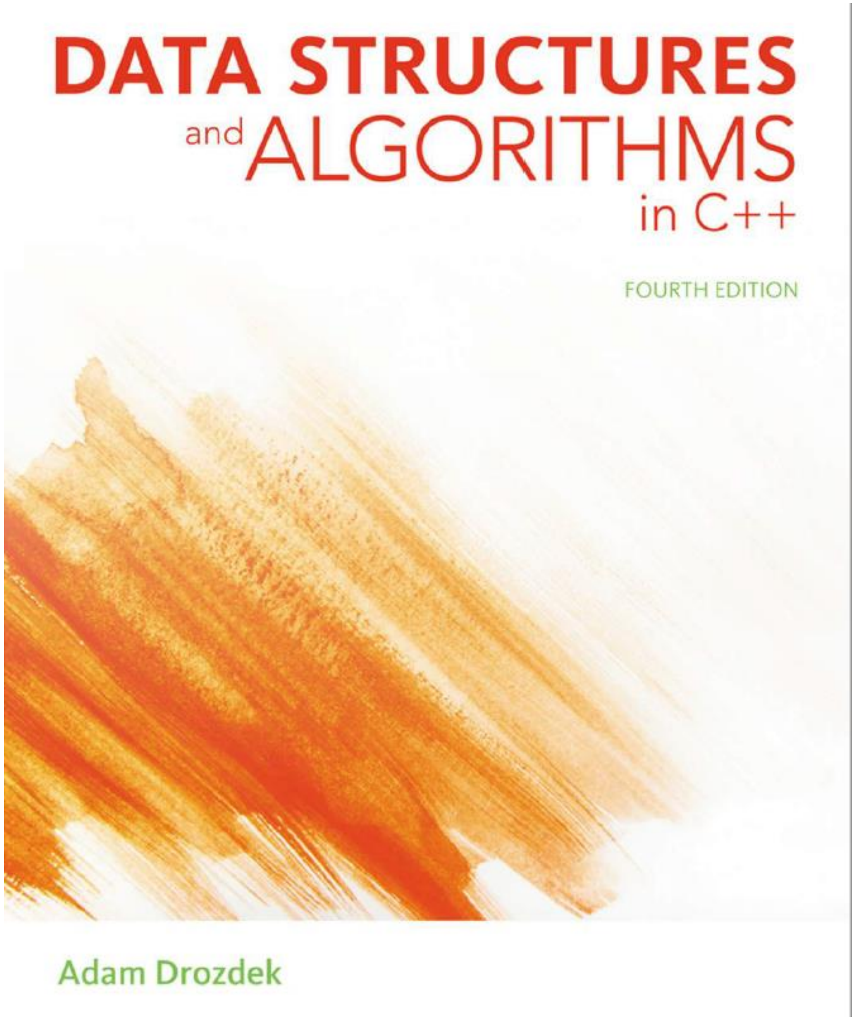
- Link to join:

https://classroom.google.com/c/NTkyNzMyNDIwNDE3?cjc=7n2quth

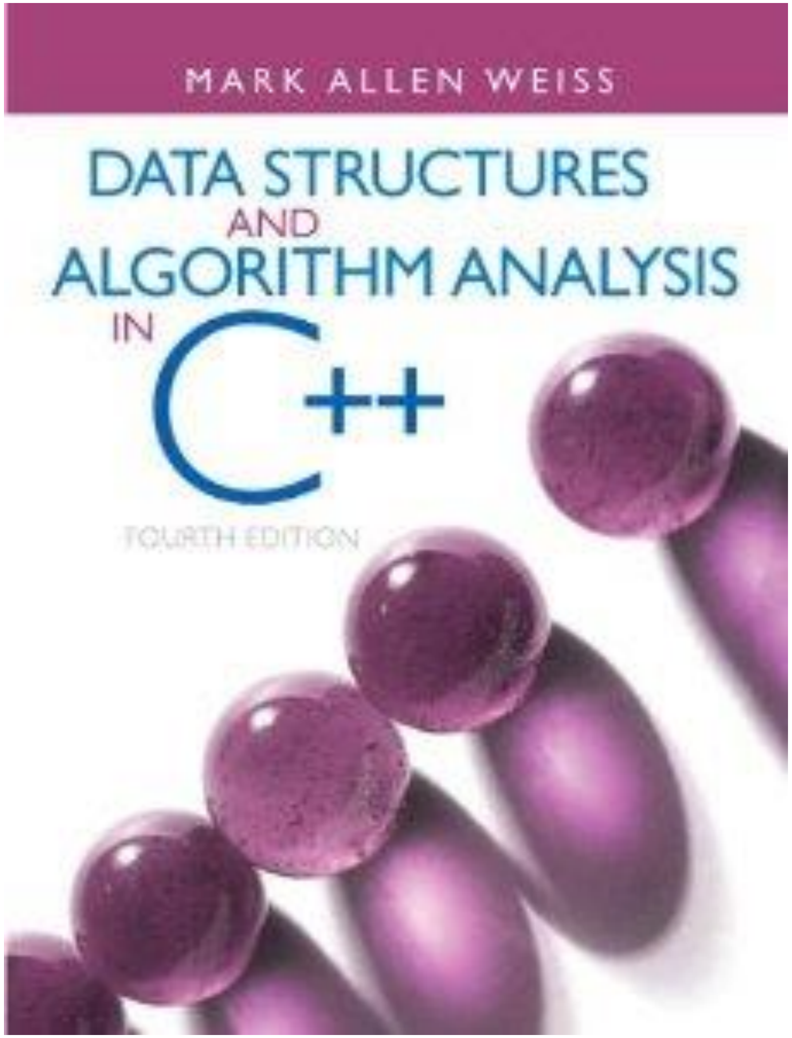
Student Assessment (might be updated)

- Assessment aims to inform students of their progress and evaluate their effort.
- Midterm 15%
- Coursework
  - Assignments 12%
  - Lab Participation 8%
    - No mark for attendance. Mark is for doing lab work.
  - Lab Quiz 5%
- Final Exam 60%

Textbook


DATA STRUCTURES
and ALGORITHMS
in C++
FOURTH EDITION
Adam Drozdek

Textbook



http://goo.gl/THfPJk

Course Syllabus

1.	Introduction	Week 1
2.	Complexity Analysis	Week 2
3.	Searching and Sorting Algorithms	Week 3, 4
4.	Linked Lists	Week 5
5.	Stacks, Queues	Week 6
6.	Binary Trees-BST	Week 7
7.	Balanced Trees-Heaps	Week 8
8.	Graphs and Graph Algorithms	Week 9,10
9.	Hash Tables	Week 11

Course Rules

- **No late delivery.**
- **Do not try cheating.**

4. ADT, Algorithms and Data Structures

D/S vs. Algorithms vs. ADT

ADT is a data type (or class) whose behavior is defined by a set of values and a set of operations. The definition of ADT only mentions what operations are to be performed but not how these operations will be implemented.

Algorithms define the set of operations that can be performed on data to produce information. Their efficiency depends on the selected data structures

Data Structures is mainly concerned with finding the best representation or organization of data In the memory that leads to efficient processing

Abstraction and Encapsulation

- **ADT** enforces encapsulation:
  - We say that the ADT encapsulates, i.e. hides, the implementation details of the algorithms and the way that data values are stored in the main memory (i.e. its data structures)
- **ADT** enforces abstraction
  - We say that the ADT abstracts, i.e. let the users of the ADT look at the data in terms of the specified operations

Example #1: Java / C++ Array D/T

- We say that Java / C++ array data type encapsulates and abstracts the representation and the manipulation of the array data values from users by providing them with a set of operations that help them declare and manipulate arrays without even caring how they are made!

Example #2: Floating Point D/T

- We know in Java / C++ how to declare a ***float*** variable and how to process its value using pre-defined operators such as `+, -, *, /, <. <=, != ,>,` etc.
- The main questions are
  - How can we represent a floating point value in the main memory?
  - How do these operators work?

  - http://www3.ntu.edu.sg/home/ehchua/programming/java/datarepresentation.html#zz-4

Example #2: Floating Point D/T

- We say that Java floating point data type encapsulates and abstracts the representation and the manipulation of the floating point data values from users by providing them with a set of operations to declare and manipulate floating point values without even caring about how these values are represented and computed!!

Why is ADT Important?

- It helps us specify precisely a collection of data through a standard set of operations (interface)
- It helps us update the representation when we discover better one (without changing the interface)
- It helps us change the implementation when we discover efficient algorithms (without changing the interface)
- Reusability
- Modularity

What is an algorithm?

- An **algorithm** is any well-defined computational procedure that takes a set of values as input and produces some values as output
  - The decimal equivalent of a binary number?

What is an algorithm?

- Algorithms can be non computer based
  - Cake recipe, instructions to get to from here to there, find a number in a phonebook
    - Or how back up your contact list on a cell phone
- Algorithms are much

older than computers

Algorithms

- People use different ways to solve their problems, so we may design different algorithms to solve a particular problem, e.g. there are so many ways to sort an array
- An ***algorithm*** is a sequence of precise instructions that specify how to solve a problem in a finite number of steps (time)

Properties of Algorithms

- Several algorithms may solve the same problem, but in differrent time
  - Much of Computer Science research is to develop efficient algorithms to solve difficult problems
  - Then implementing them in a programming language
  - Believe it or not, it can take seconds or decades
- The question is how to select the best algorithm to solve a particular problem?

Algorithm Complexity

- To solve a computational problem, an algorithm needs some resources such as
  - Processor or CPU time
  - Memory Space
  - Communication bandwidth
  - etc.
- The term ***algorithm complexity*** specifies the amount of resources needed by an algorithm to solve a problem of a certain size

Algorithm Complexity

- In this course, we will focus on two types of complexities:
  - **Time Complexity** – the amount of CPU time needed by an algorithm to solve a problem of size n
  - **Space Complexity** – the amount of memory needed to store the data of a problem of size n
- We will roughly estimate the amount of time and space taken by an algorithm by means of counting the total number of steps an algorithm performs and the number of bytes needed to solve a problem of size n

Example #1

```
Algorithm swap (x, y)
    temp ← x;
    x ← y;
    y ← temp;
```

- The algorithm **`swap`** performs only three operations to exchange the values of two variables ***x*** and ***y***, so its time complexity is constant (3 steps) and its memory requirement is only a memory word to store the ***temp*** value provided that ***x*** and **y** are already stored by the calling routine

Example #2: How many times will the printElem be called?

```
Algorithm:
 Print (array[][n], n)
   for i←1 to n do
      for j←1 to n do
         printElem (array [i][j])
```

Example #3

```
Algorithm:
   search (array[ ], n, key)
    for i←1 to n do
       if (key = array [i]) return true;
   end for
   return false;
```

- Here the ***for*** loop may run ***once*** if the key is found at the beginning of the array or at most ***n*** times when the key is not found in the array, so ….

Best, Worst, and Average Time Complexity

- we need to distinguish between three types of time complexity analyses:
  - ***Best case*** – how to estimate the minimum number of steps required to solve a problem of size n
  - ***Worst case*** – how to estimate the maximum number of steps required to solve a problem of size n
  - ***Average case*** – how to estimate the average number of steps required to solve a problem of size n (which greatly depends on the way data is distributed/stored array!)

Why Algorithms are Important?

- If you have a hard problem that needs a lot of time to solve, e.g. printing all subsets of a set of n values)
  - Any deterministic algorithm will perform at least 2^n printing steps to print all the subsets of the set
  - If we assume that our computer prints 100 subsets every second, so for a set of 100 elements, it will take more than $4x10^{18}$ centuries to complete the printing task
  - Efficient hardware ??????

Why Algorithms are Important?

- If we improve the performance of our computer 10000 times, still it needs more than 10^{10} centuries!!!!
- But if we discover an algorithm to print it in n^2 steps (which in this case impossible, but assume that for the sake of this example), we can print all the subsets in 2 minutes!!!!
- So efficient algorithm is far more important than efficient hardware!

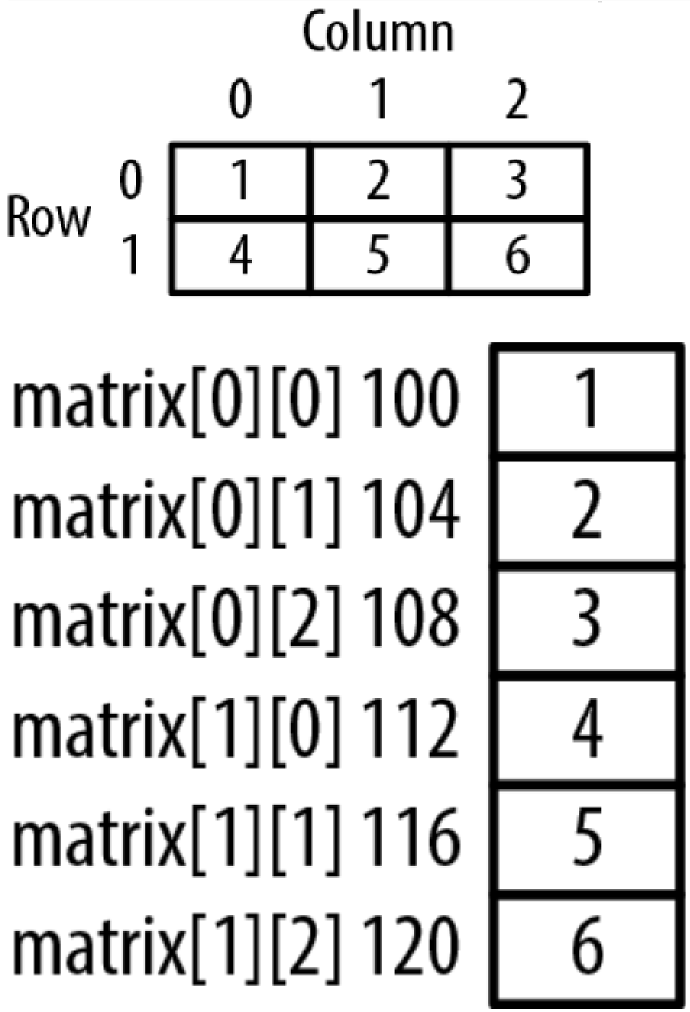
Data Structures

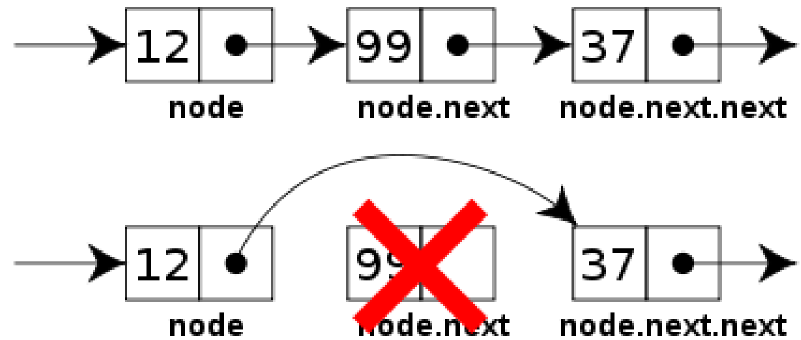
- We will use two different representations to organize data in memory:
  - Serial/Sequential organization where we store data values adjacent to each others in the main memory
  - Linked organizations where we store data values sporadically in the main memory and link them by pointers
- The sequential representation is fast but not good enough to insert values into it or delete values from it
- The linked representation is superb when we deal with dynamic structures that expand and shrink but very slow when we access it randomly

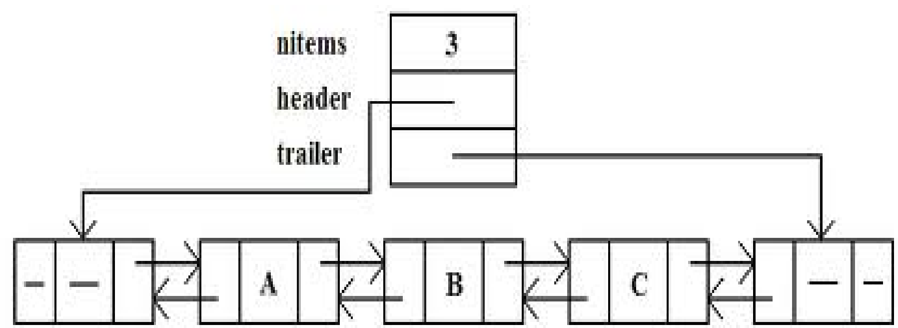
Array in C++ is an example of sequential data structure. What did you usually do to add element in the middle of a filled array?

Sequential vs. Linked Representation

```
int matrix[2][3] = {{1,2,3},{4,5,6}};
```







Samples of Data Structures

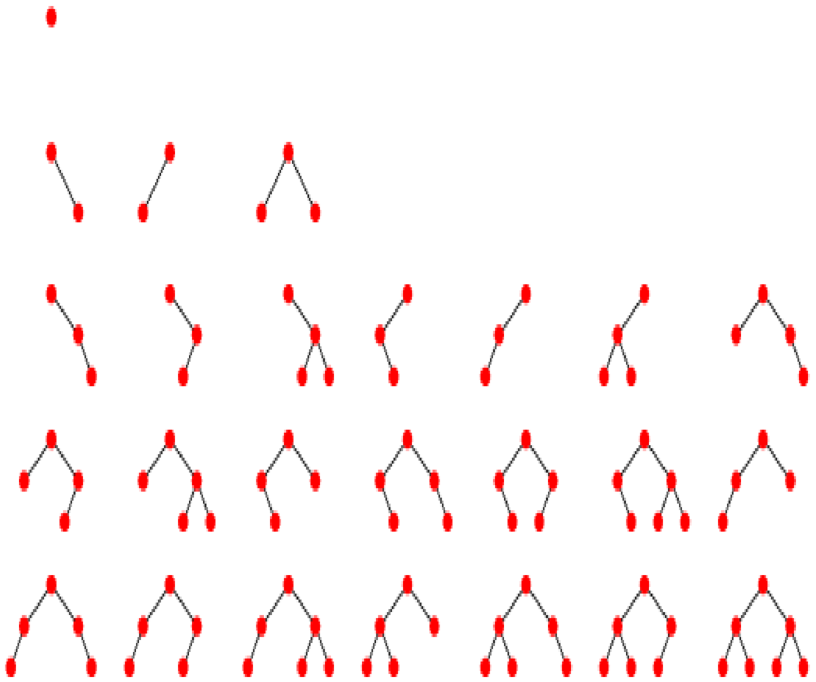
- Unrestricted linear Structures
  - String
- Restricted linear Structures
  - Stacks
  - Queues
- Non-linear Structures
  - Trees
  - Binary Search Trees
  - Priority Queues
  - Graphs

Restricted Linear Sequences

- Two important and useful data structures are:
  - Stacks – which applies First-In-Last-Out (FILO) strategy when we insert to and delete items from that sequence
  - Queues – which applies First-In-First-Out (FIFO) strategy when we insert to and delete items from that sequence
- Stacks are used extensively in computer science (e.g. compilers, program execution, etc.)
- Queues are used extensively in simulations (simulating waiting lines) and in computer networks (sequence of packets, etc.)

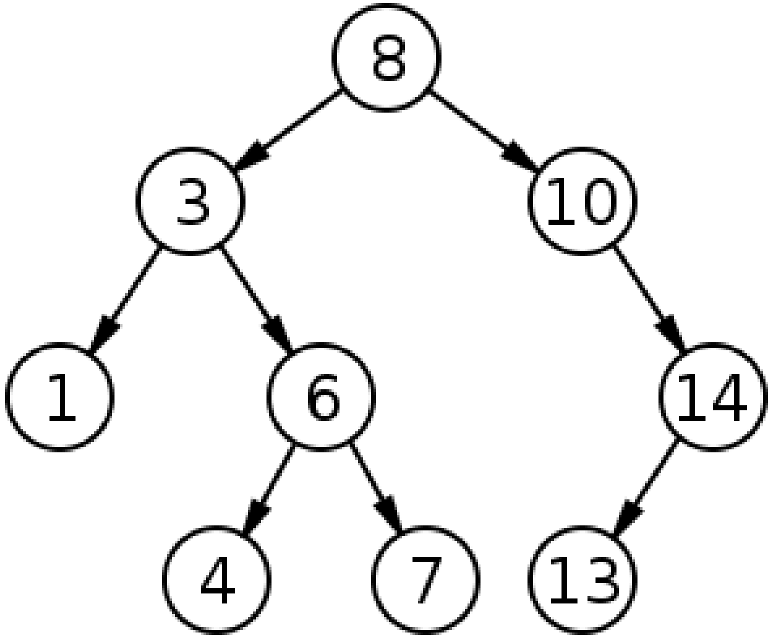
Non-linear Structures

- BINARY TREE is a tree like structure that satisfies the following conditions:
  - Each node has at most two children
  - Each node has one parent, except the root
  - Each Sibling may form a binary tree
  - No cycles are allowed



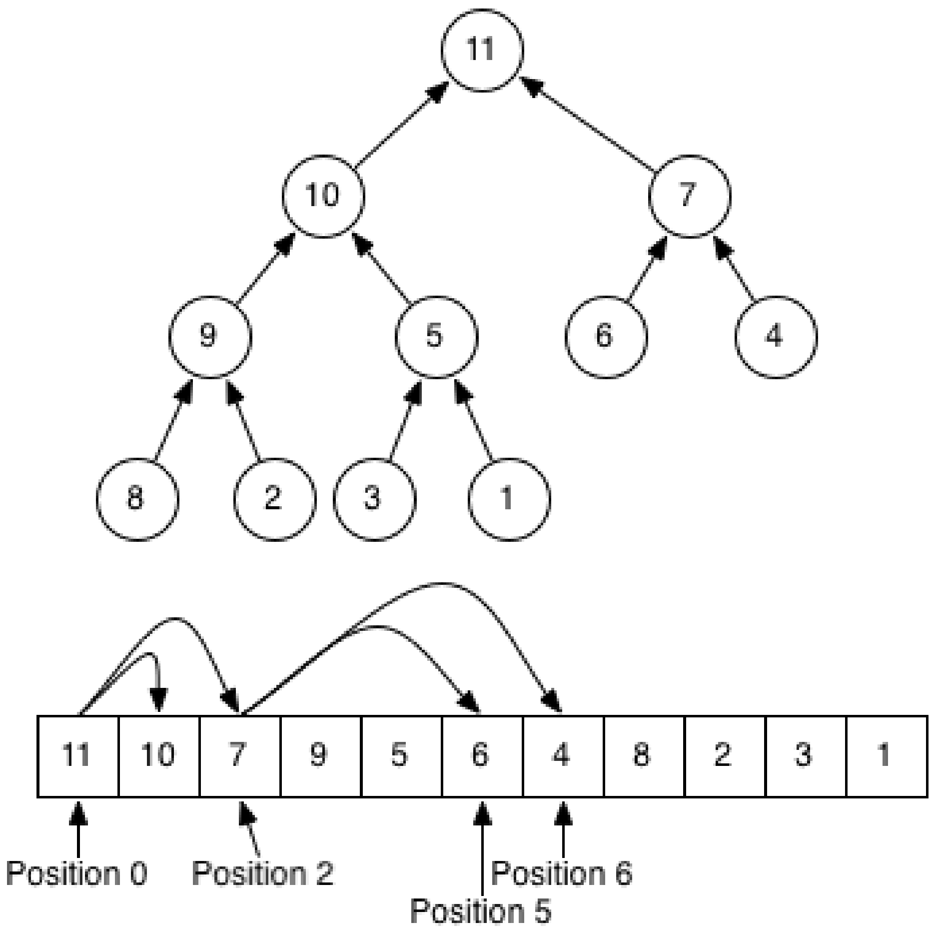
Non-linear Structures

- BST – Binary Search Trees are a kind of binary trees that is used to arrange keys in a certain order to speedup the process of searching for these keys



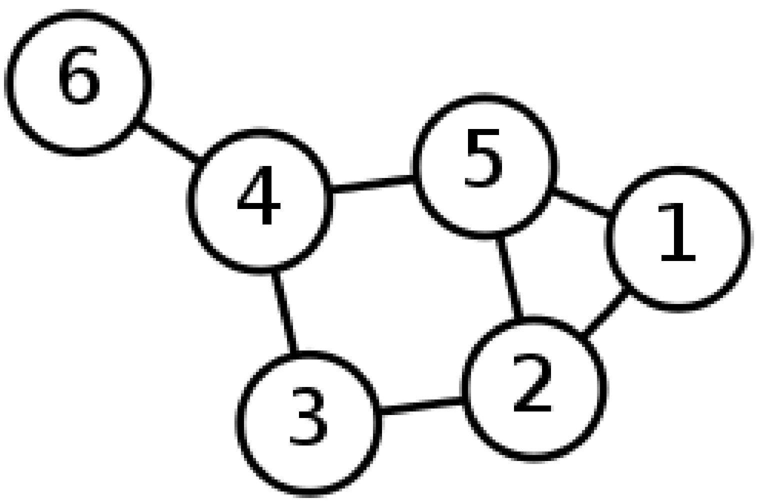
Non-linear Structures

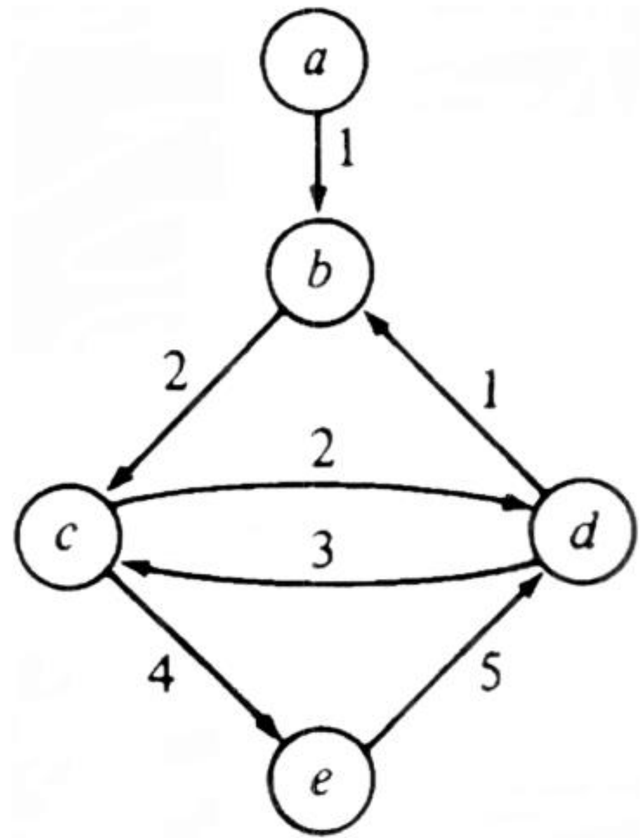
- Heap is another tree structure that is used to find the maximum or minimum key values in a single operation



Non-linear Structures

- Finally graph data structure is a collection of nodes that are linked by arcs as shown in the figure





What to do till next time?

- Review C++
- Form a team for A1