CS 201
DATA STRUCTURES

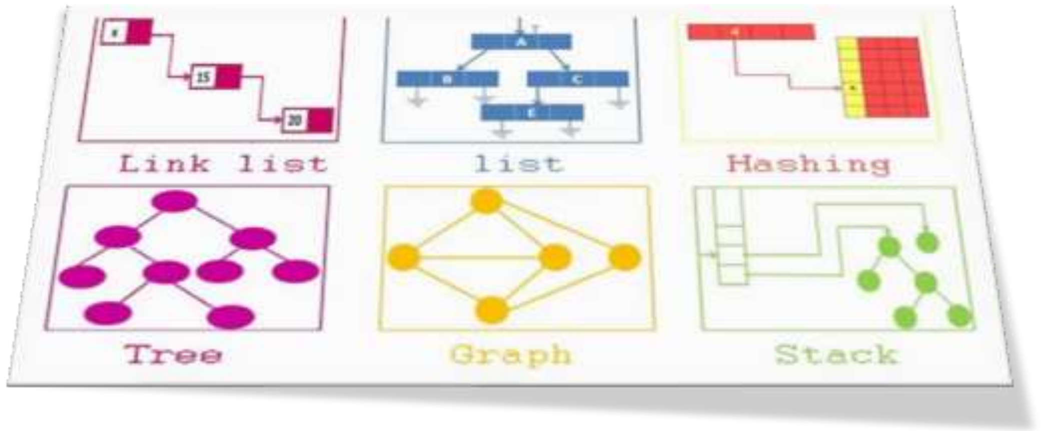
COURSE INSTRUCTOR

Course Instructor: Zareen Alamgir
Email: zareen.alamgir@nu.edu.pk
Office Hours: TBA

- **TA: refer to google classroom for TA email**

Course Objectives

Introduce students to basic data structures and related algorithms



Develop the skills to analyze time and space requirements for a data structure and its associated algorithms

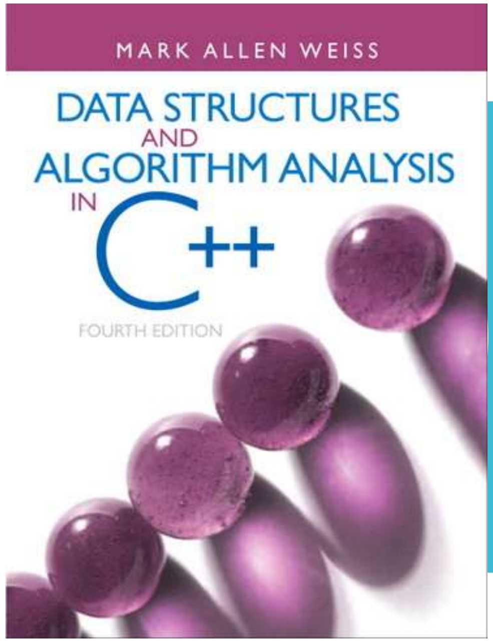
Prepare the students to pick the right data structure for a given problem

PRE-REQUISITE

- Computer Programming
  - It is assumed that students have a very good command on programming in C++.

https://www.goodcore.co.uk/blog/coding-vs-programming/

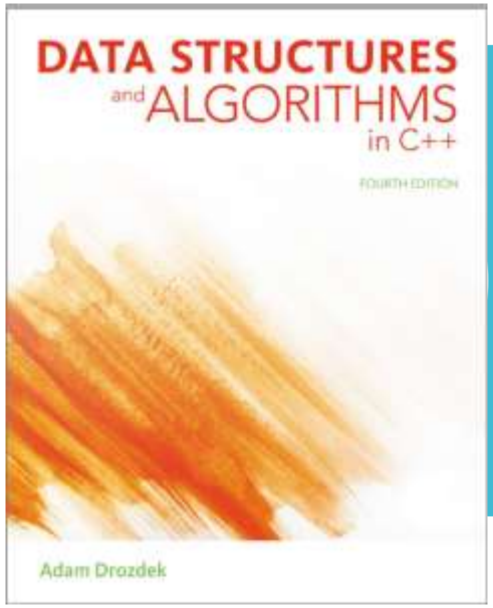
Textbook



Data structures and algorithm analysis

Mark Allen Weiss

4th Edition



Data structures and algorithms in C++

Adam Drozdek

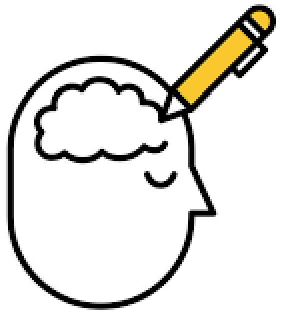
4th Edition

Grading Scheme

• Midterms	30%
• Quizzes	10%
• HomeWorks	0%
• Assignments	20%
• Final:	40%

Absolute grading scheme

Important



- ***There will be no makeup quiz.***

- ***Quiz can be announced or Surprise quiz. S***o always come prepared in the class.

- Submit assignments on time. No late assignments will be accepted.

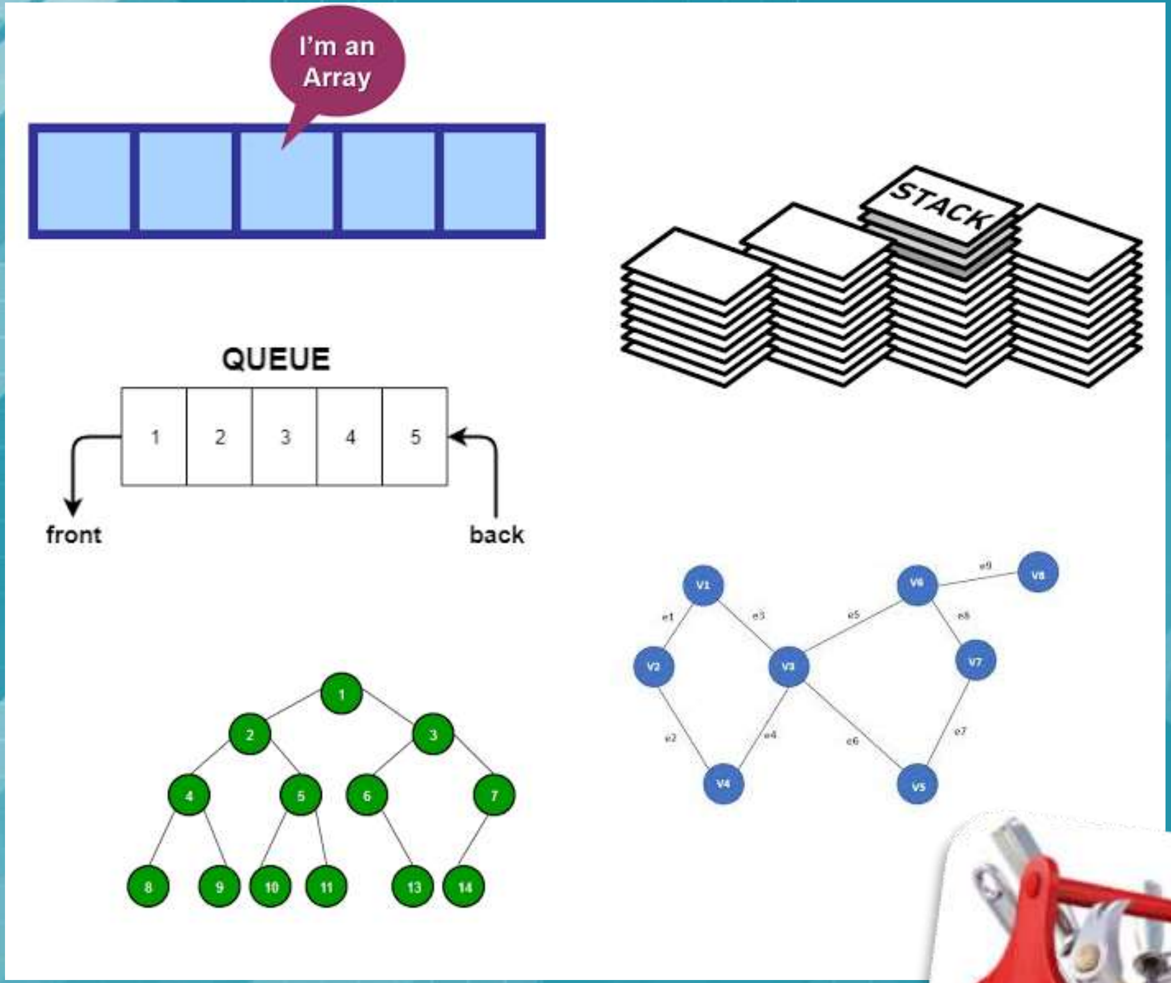
- There is a strict policy against plagiarism and cheating. **The penalty can be an F grade.**

- Be on time in class. All late comers will be marked absent.

TENTATIVE COURSE OUTLINE & LECTURE PLAN

NO. OF LECTURES	TOPICS
1	Introduction
2	Time Complexity Analysis and Asymptotic Bounds
4	Linked Lists Review of pointers Singly linked lists, doubly linked lists, circular lists and corresponding iterators
3	Stacks and Queues
MIDTERM 1	
2	Recursion
3	Trees Binary trees and their traversals Binary search trees (Insertion, Deletion and Search)
3	Height Balanced Binary Search Trees (AVL Trees)
2	Heaps and heap sort
MIDTERM 2	
1	Data compression and Huffman coding
2	Hashing Hash tables and hash functions Collision resolution
2	Universal hashing
3	Graphs, Breadth first search and Depth first search

Introduction to DS



WHAT IS DATA STRUCTURES ABOUT?

A data structure is a way of organizing data so that it can be used efficiently

Data structures are a key to designing efficient algorithms as a good representation of data can enable us to process the data more efficiently

Vs.

WHY DATA STRUCTURES ?

- Design an **efficient** search engine, as good as Google’s
- Build an **efficient** security system based on face recognition
- Understand the human genome and trace your ancestry

Data structures are a key for designing efficient algorithms

What is an efficient algorithm ?

How do we know that algorithm is efficient ?

What is an efficient algorithm ?

First let's determine **…**

What does efficient algorithm mean?

Time & Space efficient

If an algorithm takes 1 year to run and need 60TB of RAM

Then it's probably useless for us

How can we determine the efficiency of an algorithm?

We measure the cost of an algorithm in terms of **time and space.**

Time is usually considered more imp than space.

For now we will only concentrate on time …

How to compute Time Complexity?

- **Idea 1: Code the two algorithms and note the time they take.**
- **Issues:**
  - The machine should be same,
  - The language should be same.
  - We need to code and then we can calculate the time complexity.
- **Is it a reasonable approach ?**
  - **Think …**
- **Conclusion: ?**

How to compute Time Complexity?

- **Idea 2: Compute the time complexity without programing (executing) the algorithm.**

- **How ?**
  - We need ***logical units that express a relationship between the <u>size n of input</u> and the <u>time t</u> required to process the given input.***

Algorithm Analysis

Estimate the performance of an algorithm through

The **number of operations** required to process an **input of certain size**

ALGORITHM

```
i=0
sum=0
while(i<N ){
    sum++
    i++
}
```

Size of Input Matters

The differences between the algorithms may be immaterial for processing a small number of data items, but these differences grow with the amount of data.

Our aim is to see

How a program performs for reasonably <u>large input</u>

Number of operations matters

Suppose a machine that performs a million floating-point operations per second (10^6 FLOPS),

Then how long an algorithm will run for an input of size n=50?

1) If the algorithm requires **n^2** such operations:
   0.0025 second
2) If the algorithm requires **2^n** such operations:
   over 35 years!

Algorithm Analysis

- Estimate the performance of an algorithm through
  - the **number of operations (t)** required to process an **input of size n**
- We require a function that express relation between **n & t.** We call it **Time complexity function T(n)**
  - To calculate T(n) we need to compute the total number of **program steps**

Program step can be an executable statement or meaningful program segment (comparison and assignment statements)

The execution Time of Algorithms

- Each operation in an algorithm (or a program) has a cost.

  ➔ Each operation takes a certain amount of time.

- `count = count + 1;`

➔ in this case time taken is **constant** (not depended on input)

- To keep things simple let's assume each statement takes 1 unit of time (logical)

A sequence of operations:

```
count = count + 1;        1 unit of time
sum = sum + count;        1 unit of time
```

➔ Total Cost = 2

Simple If-Statement

Example: Simple If-Statement

	Cost	# of Times
`if (n < 0)`	1	1
`absval = -n`	1	1
`else`		
`absval = n;`	1	1

Total Cost <= 3

We are usually concern with the frequency of each statement

To make things **simple and rough** estimate of the run time of algorithm we consider all cost are same as 1 unit of time

Analysis of Loop

Algorithm	Cost (time complexity)

`i=0;`
`sum = 0;`
`while(i<N ){`
`        sum ++;`
`        i++`
`}`	

The condition always executes one more time than the loop itself

Analysis of Loop

Algorithm	Cost (time complexity)
`i=0;`	**1**
`sum = 0;`	**1**
`while(i<N ){`	**N+1**
`sum ++;`	**N**
`i++`	**N**
`}`	

The condition always executes one more time than the loop itself

```
while(i<N ){
        sum ++;
        i++
}
```

T(N) = 1 + 1 + N+1 + N + N

T(N) = 3N+3

Calculating T(n) Program step

- A **program step** is the syntactically / semantically meaningful segments of a program
- A step DOESNOT correspond *to a definite time unit but a logical one*

A step count is telling us how run time of a program changes with change in data size

- ***Calculate the total number of steps/executable statements in a program***
- ***Find the frequency of each statement***
- ***Sum them up***

Algorithm	T(N)
i=0;	**1**
sum = 0;	**1**
while(i<N){	**N+1**
sum ++;	**N**
i++	**N**
}	

Purpose of algorithm analysis

- To estimate how long a program will run.
- To estimate the largest input that can reasonably be given to the program.
- To compare the efficiency of different algorithms.
- To help focus on the parts of code that are executed the largest number of times.
- To choose an algorithm for an application.

TO DO

Revise Computer programming concepts

- Chapter 1 of Alan Weiss's book
- Chapter 1 of Adam Drozdek's book