CSC 1203 & SWE 1202
Data Structures and Algorithms

Introduction

David Sabiiti Bamutura
Lecturer
+2560704496748 | dbamutura@must.ac.ug
&
Joachim Musimenta
+256789231327 | 2021bse084@std.must.ac.ug
Tutorial Assistant

What are data structures?(1)

Data structures and algorithms are fundamental building blocks in almost all software products.

Knowledge of data abstraction, data structures, and algorithms is important in the construction, use, and maintenance of adaptable, reusable, and efficient program components.

(the syllabus, not your course outline)

What are data structures?(2)

A data structure is a data organisation, management, and storage format that enables efficient access and modification.

More precisely, a data structure is a collection of data values, the relationships among them, and the functions or operations that can be applied to the data.
(wikipedia)

What are data structures?(3)

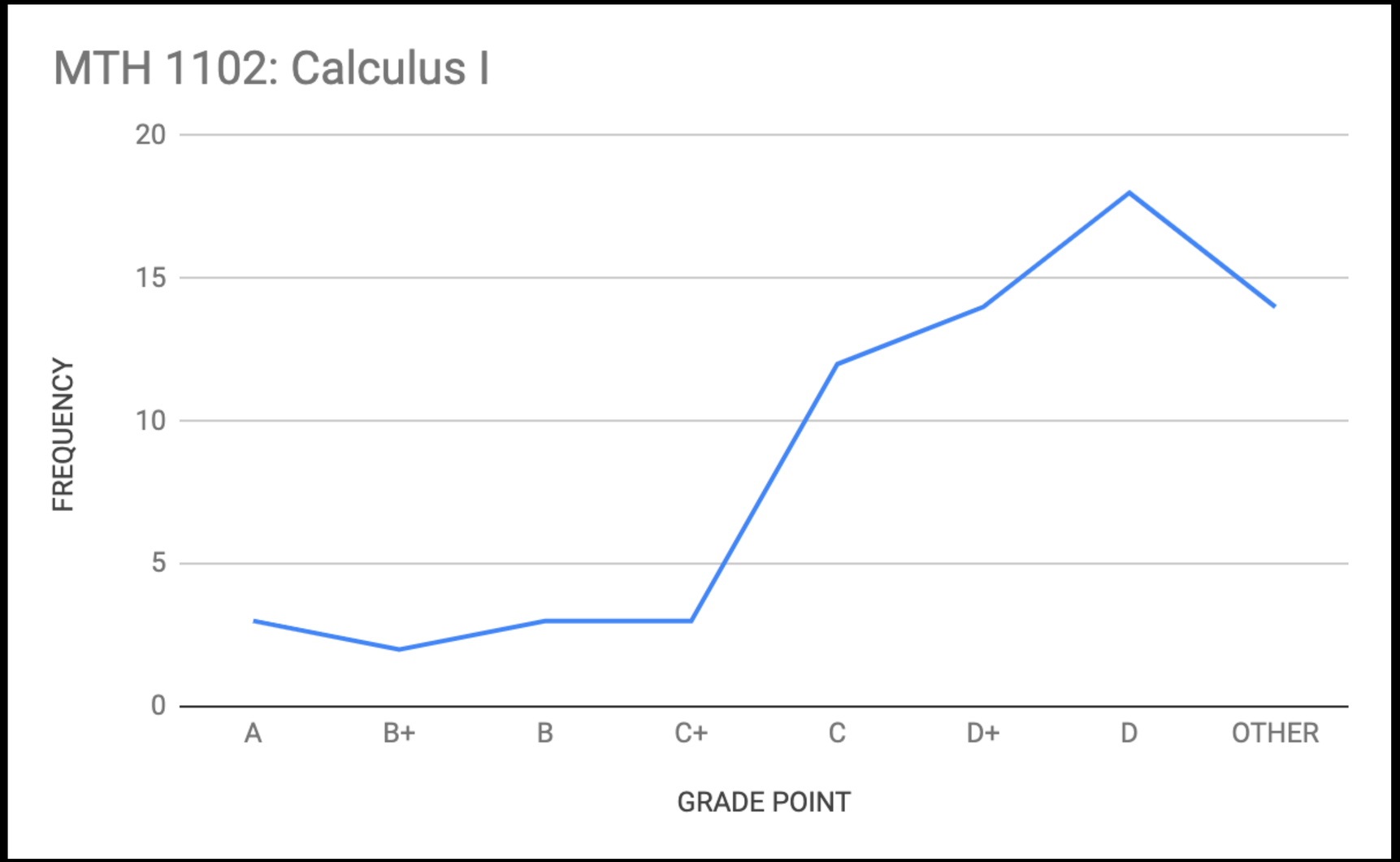
Data structures are the essential building blocks that we use to organise all of our digital information.
Choosing the right data structure allows us to use the algorithms we want and keeps our code running smoothly.

Understanding data structures and how to use them well can play a vital role in many situations.
(codecademy)

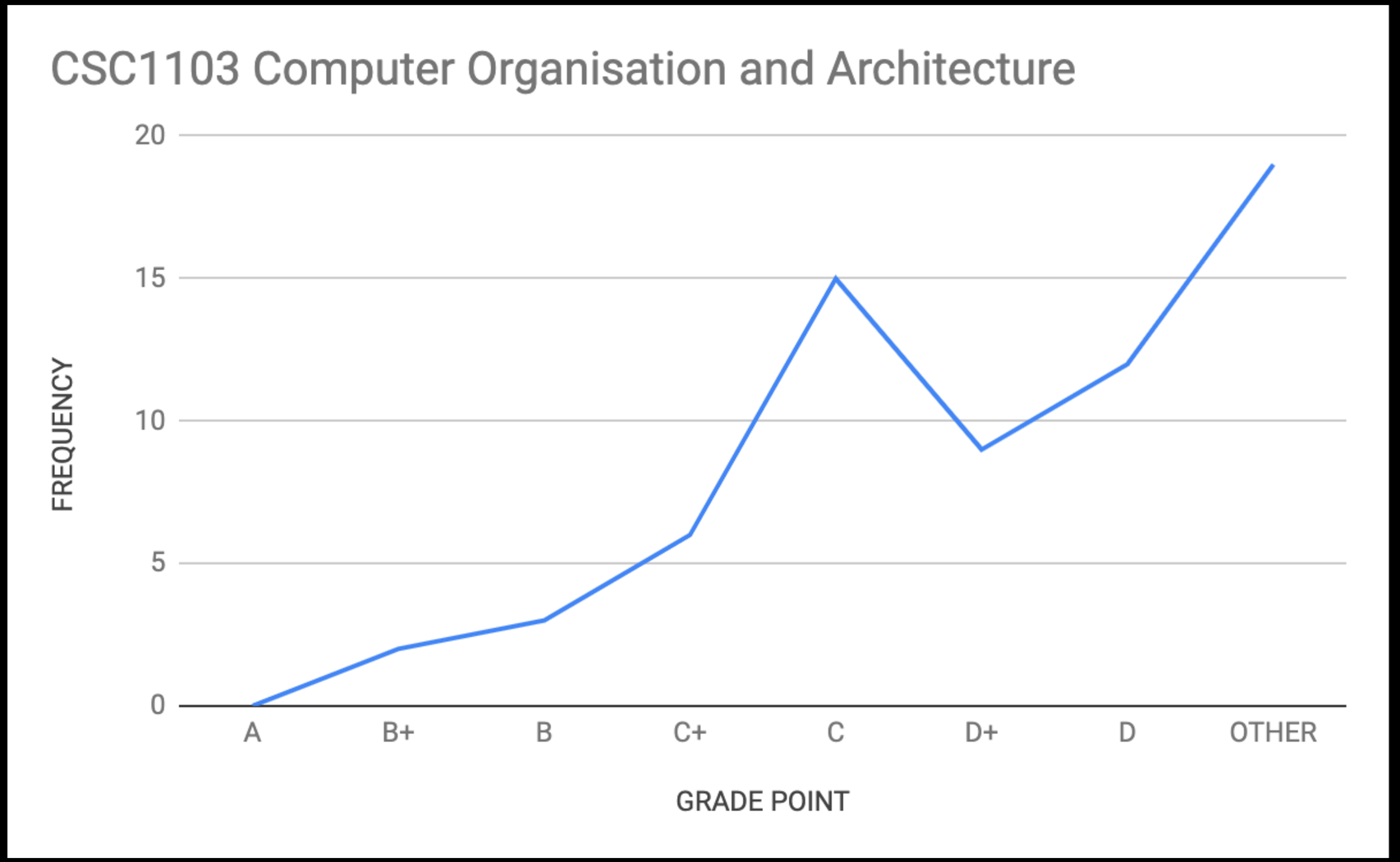
What is this course?

- Introductory course about solving (Computational) Problems
- Taken by all computing students (CS, SWE)
- Requires strong knowledge of & skills in
  - Structured or Object Oriented Programming e.g. CSC 1102, CSC 1202 (you are doing now!)
  - Discrete Mathematics e.g SWE 1101
- Some Calculus and Computer Organisation & Architecture (BSE doing it this semester, BCS last semester)
- 1 Lecturer, 1 Tutorial assistant
- Grading
  - Course Work(/40): 2 written tests, 3 Problem sets, 2 compulsory laboratories,
  - Exam(/60): Written (60/100) & Practical (40/100)

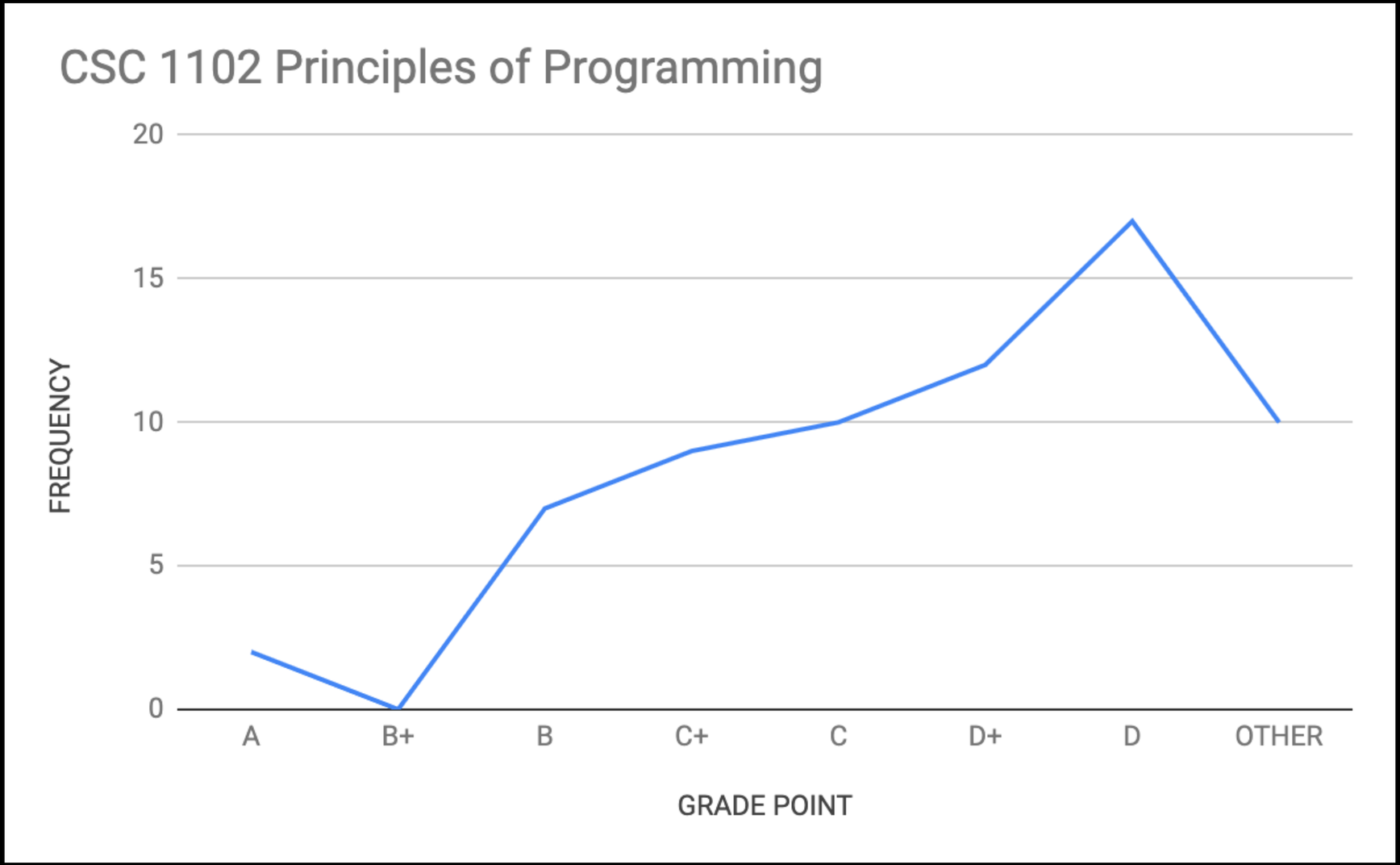
Which grade did you get from MTH 1102 Calculus (BCS)?



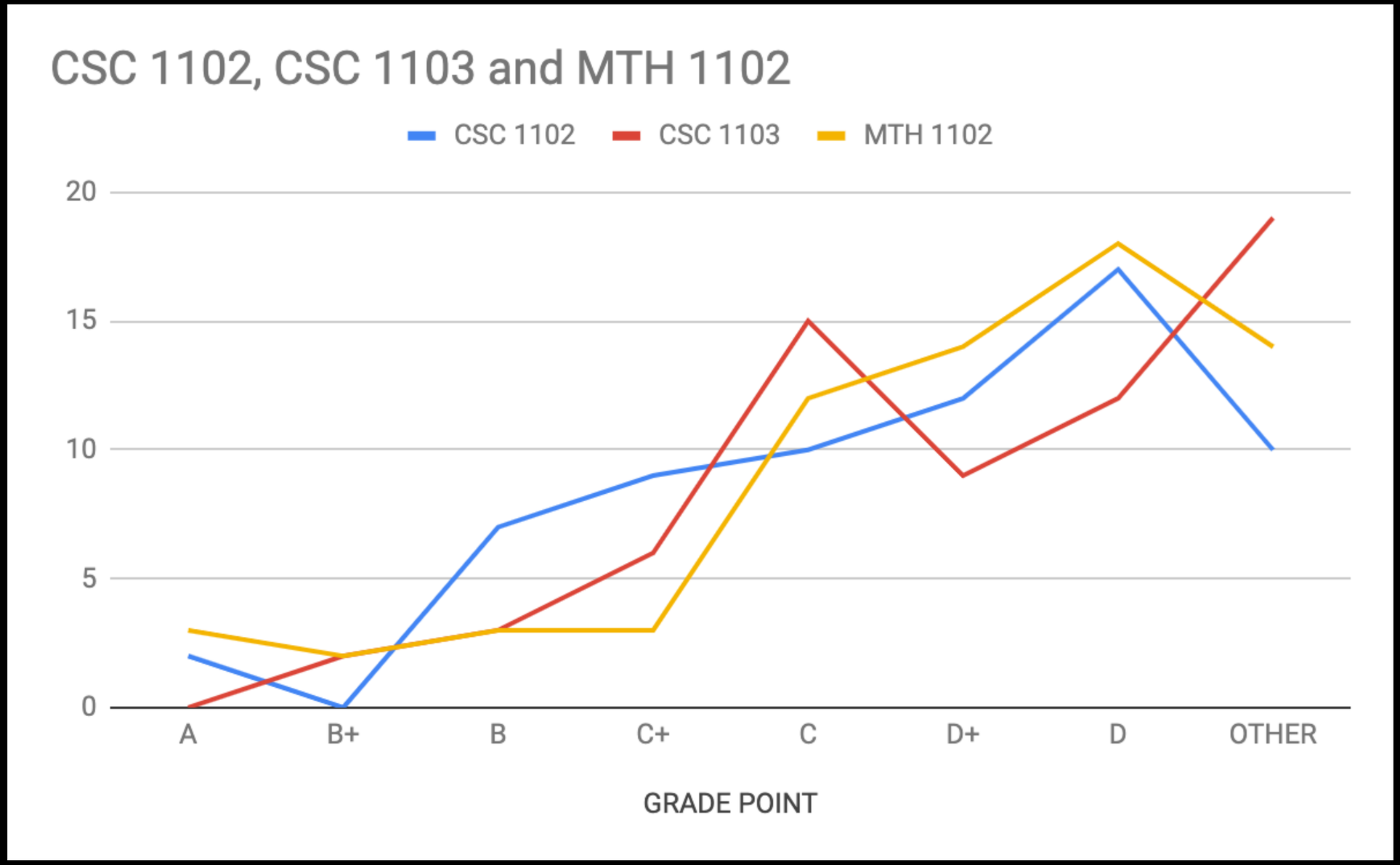
Which grade did you get from Computer Architecture & Organisation (BCS)?



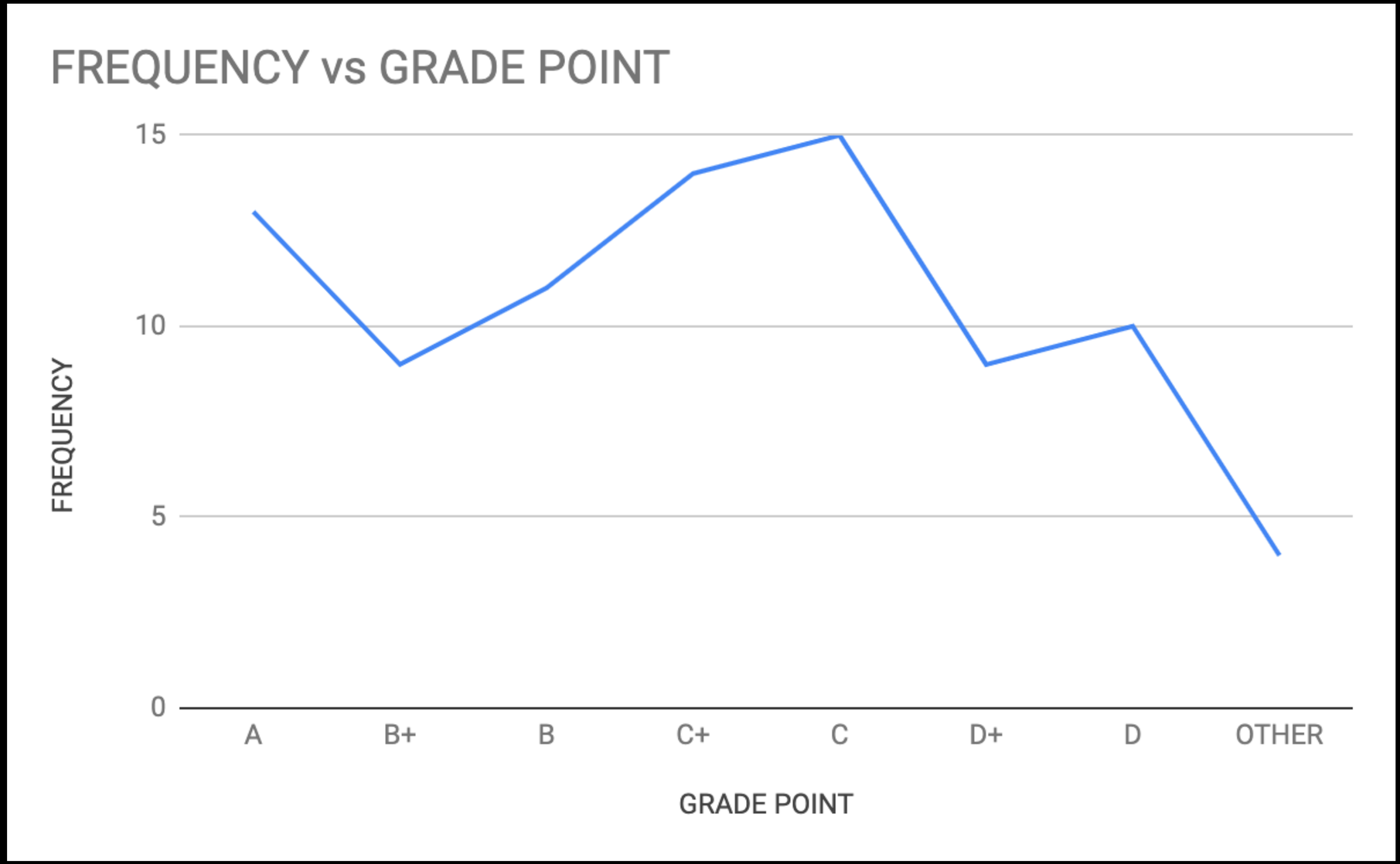
Which grade did you get from CSC 1102 Structured Programming (BSC)?



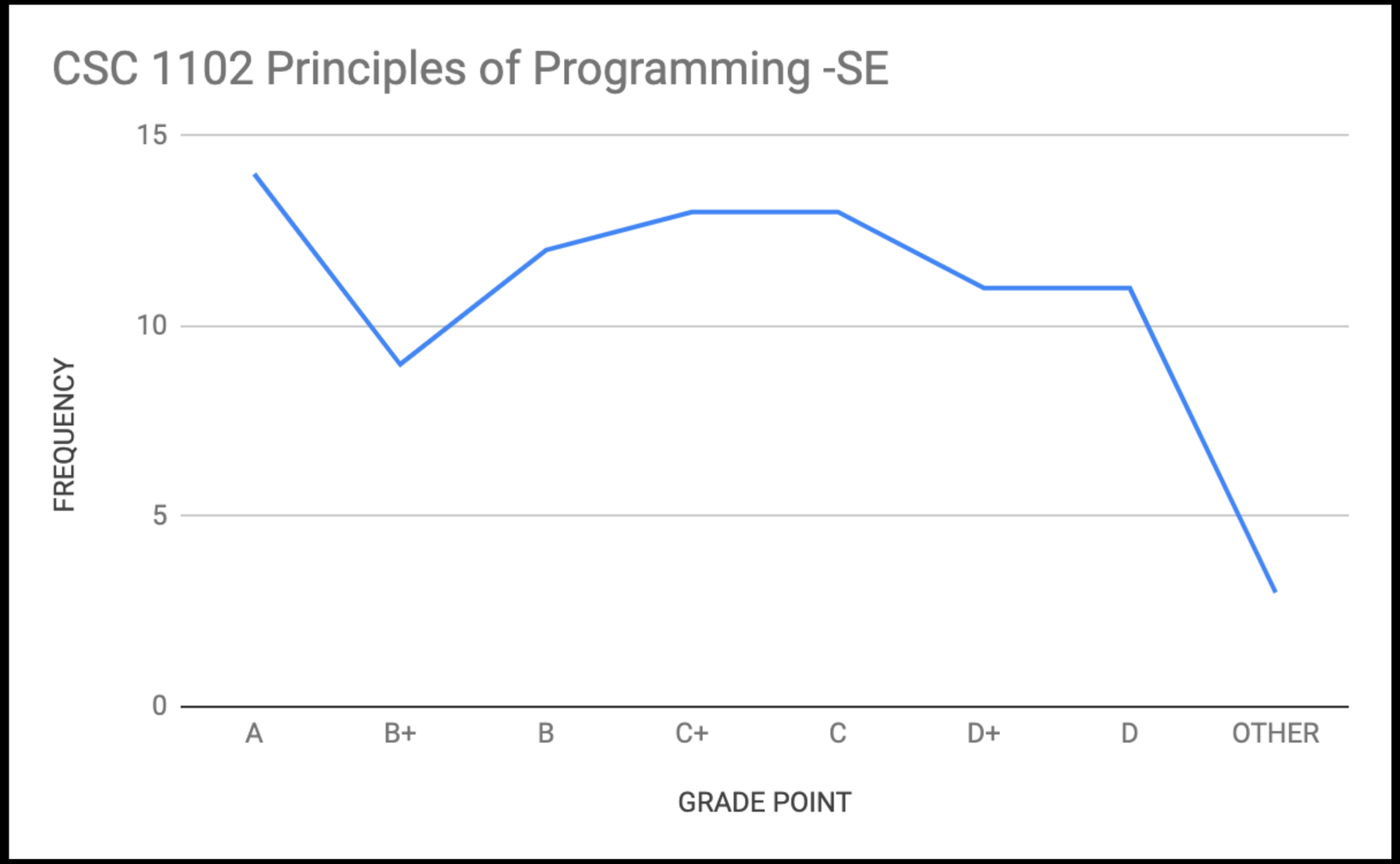
Combining MTH 1102, CSC 1103 and CSC 1102



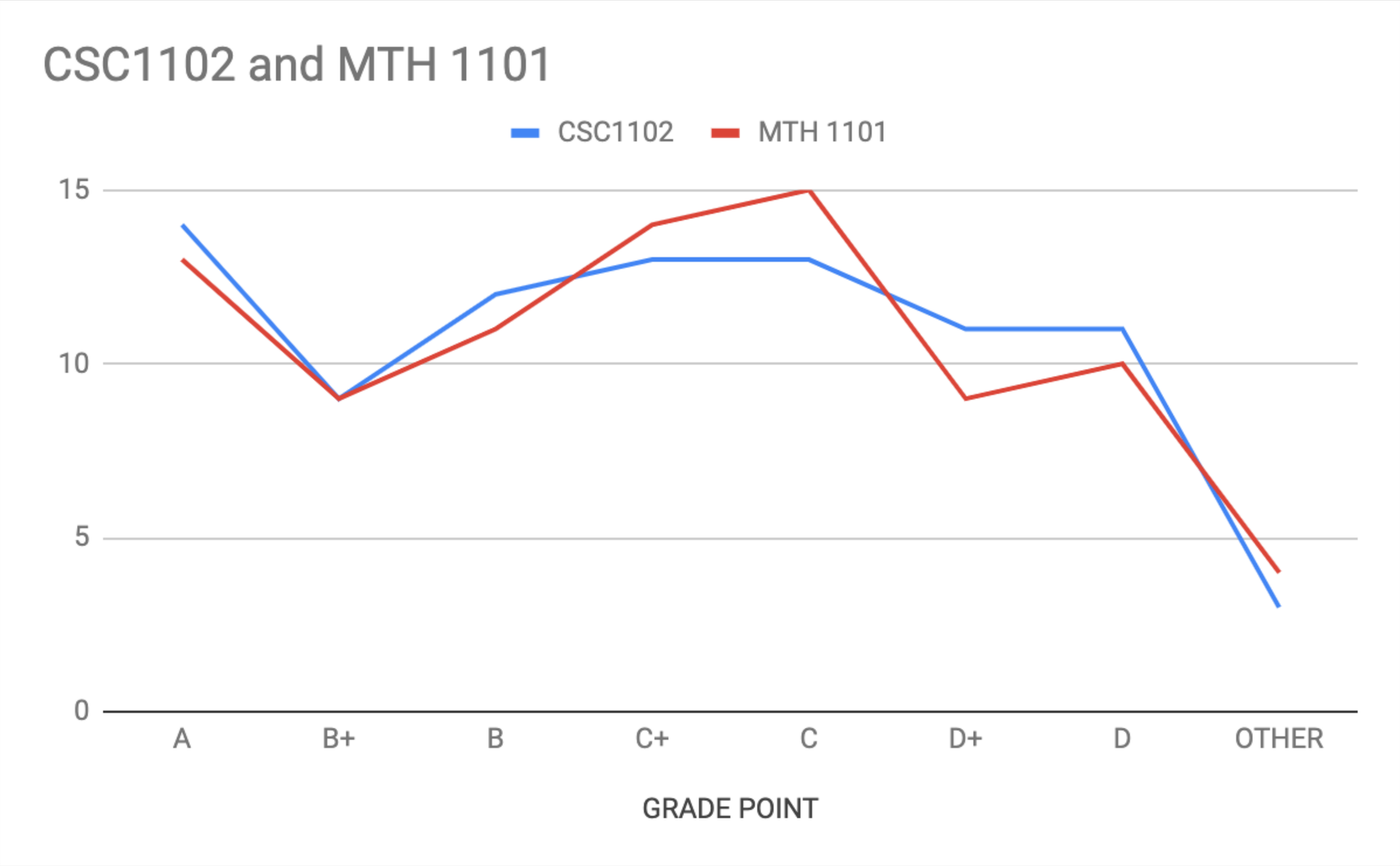
Which grade did you get from MTH 1101 Discrete Mathematics (BSE)?



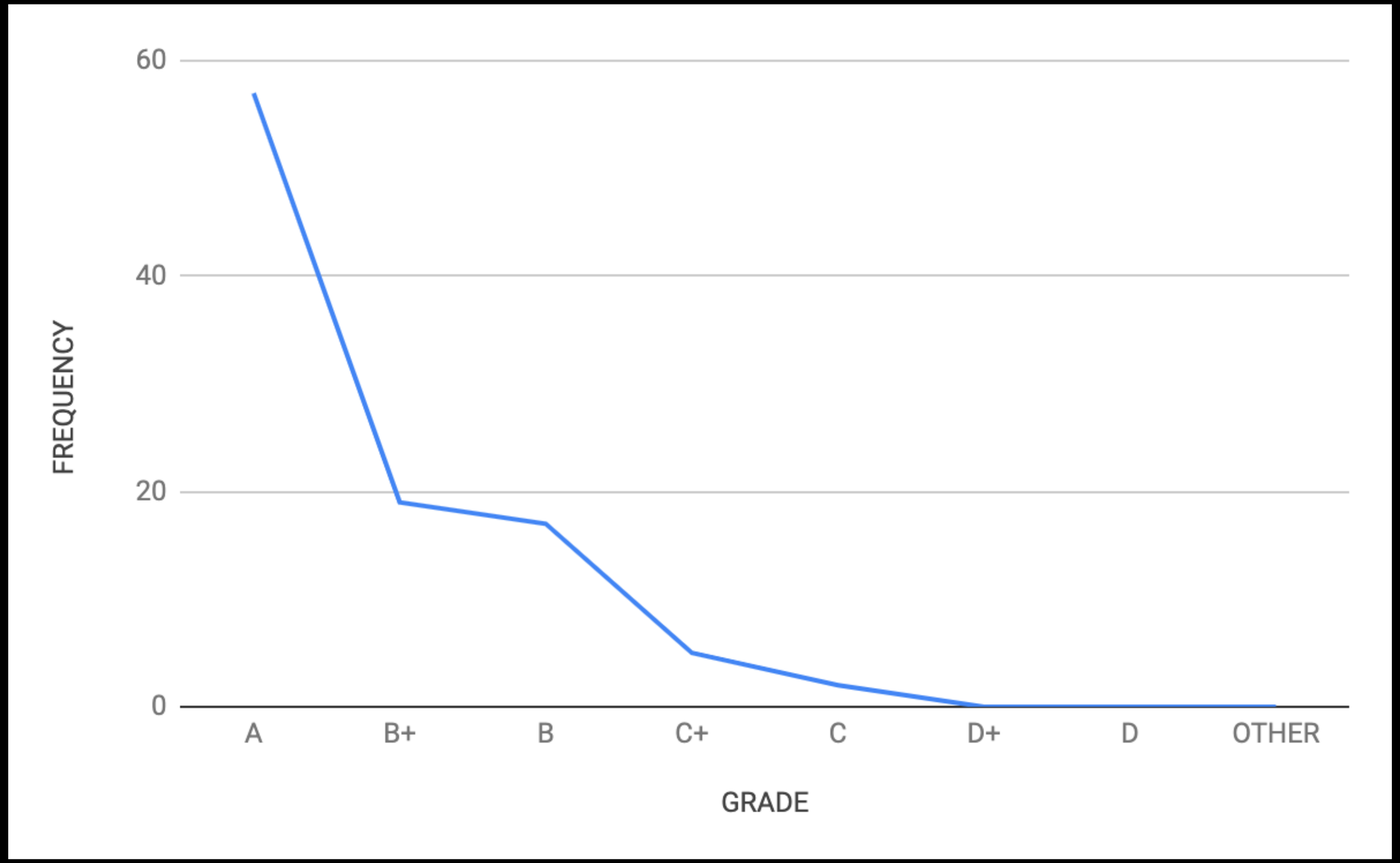
Which grade did you get from CSC 1102 Structured Programming (BSE)?



Combining MTH 1101 and CSC 1102



Which grade are you aiming for in SWE 1202 / CSC 1203?



Which grade do you think you will get ?

Lectures, Seminars and Tutorials

- Lectures
  - Thursdays
    - 1400 - 1645 (Kihumuro Library Room 2)
    - 1700 – 16:45 (Kihumuro Library Room 2)
- Tutorials
  - Thursdays 1400 - 1800 (Kihumuro Library Room 2)
- Seminar and Lab supervision
  - Thursdays 1400 - 1800 (Kihumuro Library Room 2)

Learning Management Tools

- MUST Google Classroom
  - register yourself on the platformusing your @std.must.ac.ug
  - enrol yourself using the class code: 42dck7f
- For questions and Discussions
  - Use Google Classroom in a browser or apps on Desktop, laptop, tablet or smartphone
  - WhatsApp : DSA(23-24)
  - Zoom/google meet is also an option

Problem sets & Laborations

- Problem sets
  - Submission: Both
    - handwritten solutions and
    - scanned pdf versions (<= 2MB) should be submitted via Google Classroom
- Both Problem sets & Labs should
  - be done in groups of 5 with at least 2 ladies
- form groups quickly!

Problem set deadlines

- Problem set 0:
  - Out: 4/03/2024
  - Due: 16/04/2024
- Problem set 1:
  - Out: 18/03/2024
  - Due: 13/04/2024
- Problem set 2:
  - Out: 15/04/2024
  - Due: 27/04/2024
- Problem set 3:
  - Out: 29/05/2024
  - Due: 11/05/2024

Lab deadlines

- Lab 1: start 7/03/2024, first deadline 13/03/2024, final deadline 27/03/2024
- Lab 2: start 21/03/2024, first deadline 3/04/2024, final deadline 17/04/2024
- Lab 3: start 11/04/2024, first deadline 24/04/2024, final deadline 07/05/2024

Laboration Rules

- All group members must understand submitted solution
  - Be able to answer questions about code that someone else in the group implemented
- Discussions with other groups is allowed but should stop at high level
  - You are not allowed to give away solutions
- Deadlines must be respected
- Your submissions will be graded quickly. If the submission is not accepted
  - You get feedback to help you improve before final deadline
  - incomplete submission should prove serious attempt at solving the lab
- Deadlines may change forwards due to unforeseen circumstances

Written Tests & Final Examination

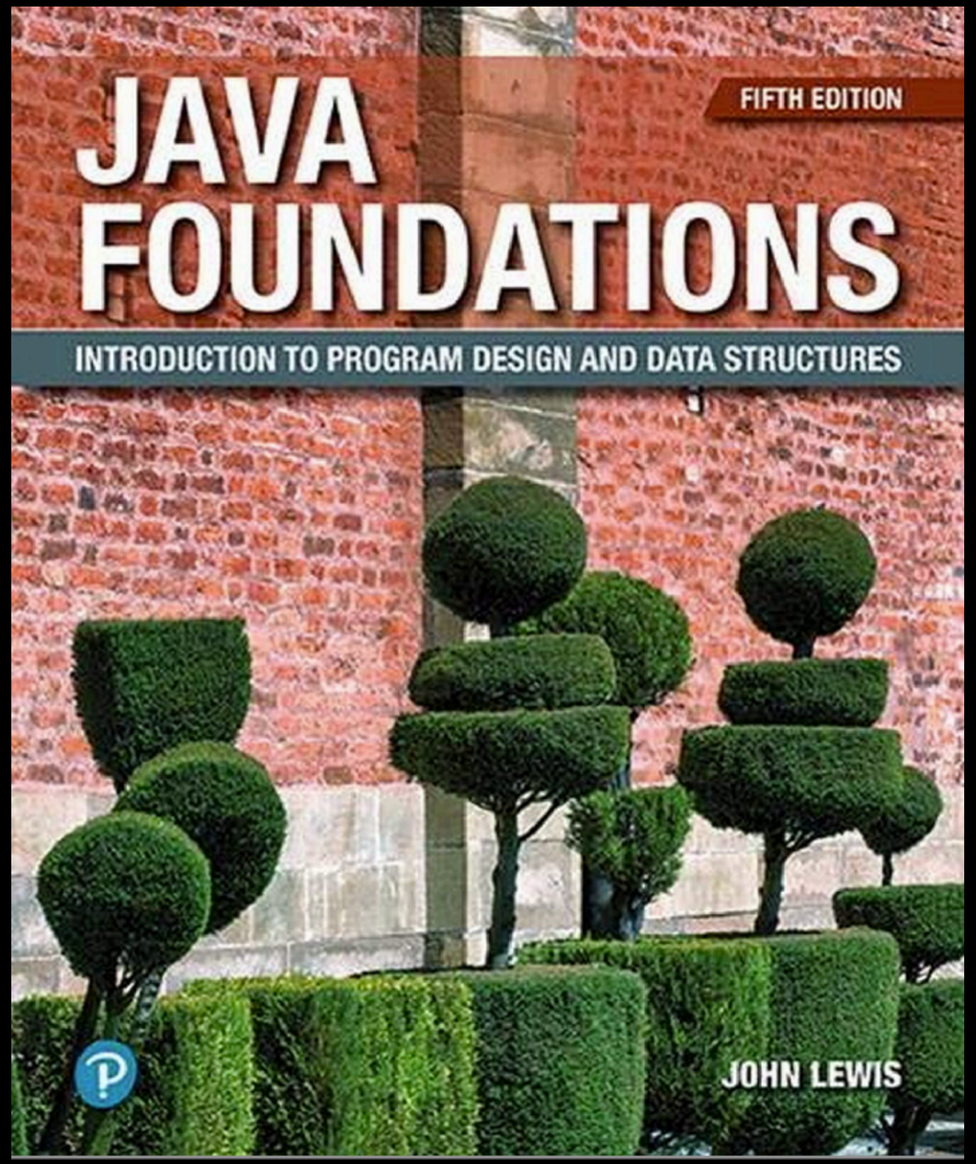
- All Tests shall be done on a Thursday during lecture time
  - Test 1: Between 27/03/2024 & 12/04/2024
  - Test 2: Between 30/04/2024 & 10/05/2024
  - Scripts returned within 1-3 weeks.
- Final Exam
  - As per the university time table

Course Texts

Main Course Book I (JL)

List of Chapters

- Recap of Programming
  - Arrays (7)
  - Inheritance (8)
  - Polymorphism (9)
  - Iterators (16)
  - Comparables & Comparators
  - Recursion (17)
- Part I
  - Analysis of Algorithms (11)
  - Introduction to collections (12)
  - Stacks (12 &13)
  - Queues (14)
  - Lists (15)
  - Searching and Sorting (18)
  - Trees (19)
  - Binary Search Trees (20)
  - Heaps and Priority Queues (21)
  - AVL and Red Black Trees (another book)
  - Sets and Maps (22)
  - Multi-way Search trees (23) i.e. 2-3 and 2-4 trees
  - Graphs (24)



Course Book II

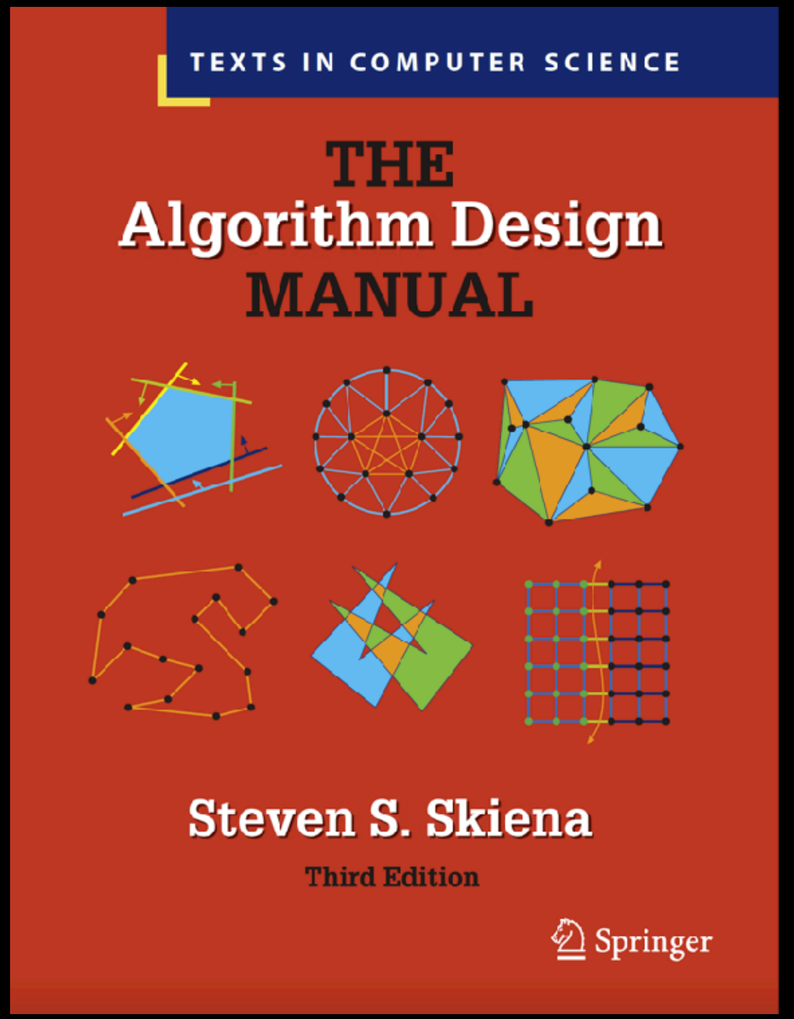
Algorithms(Sedgewick & Wayne)

- Referred to as Algorithms(S&W)
- ~~Better TOC detailed on LMS~~
- Some material is based on
  - the book
  - a course on coursera
- Use it for the Labs too.

Course Books III

Algorithms Design Manual (Skiena)

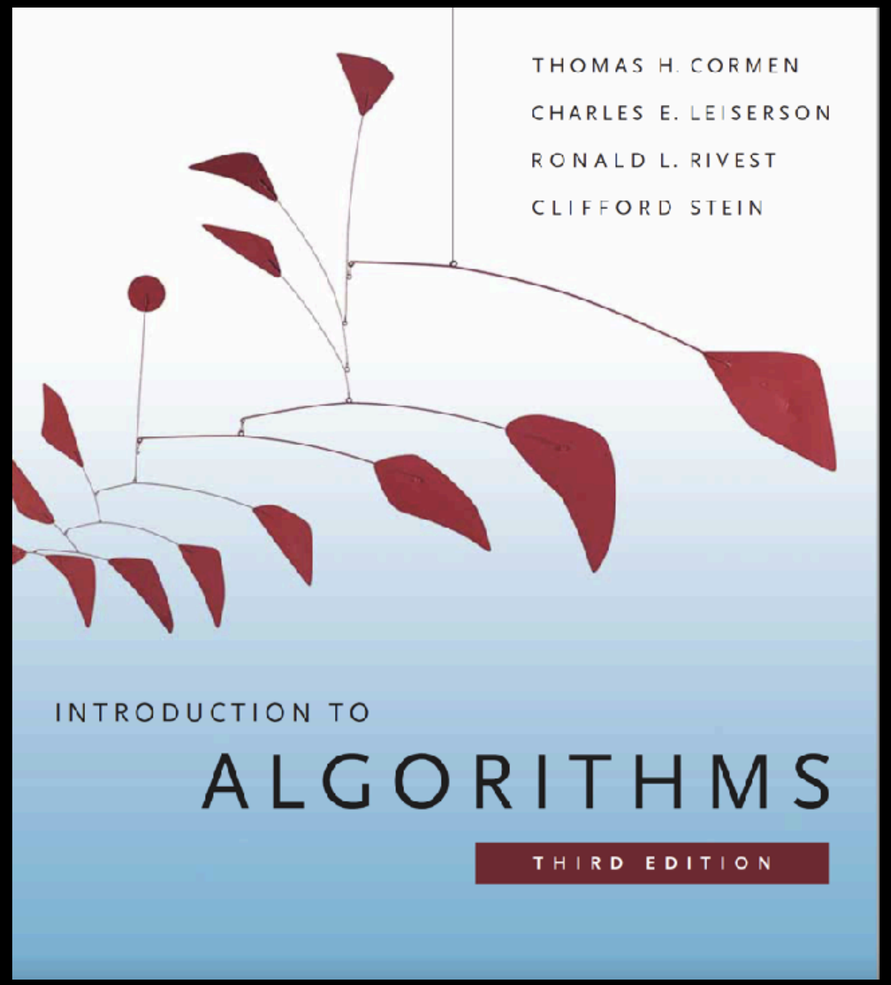
- Referred to as ADM(S)
- TOC available
- Some material is based on
  - the book
  - a course on skiena's website
- Use it for the Labs too.



Course Book IV

Introduction to Algorithms (Cormen et al)

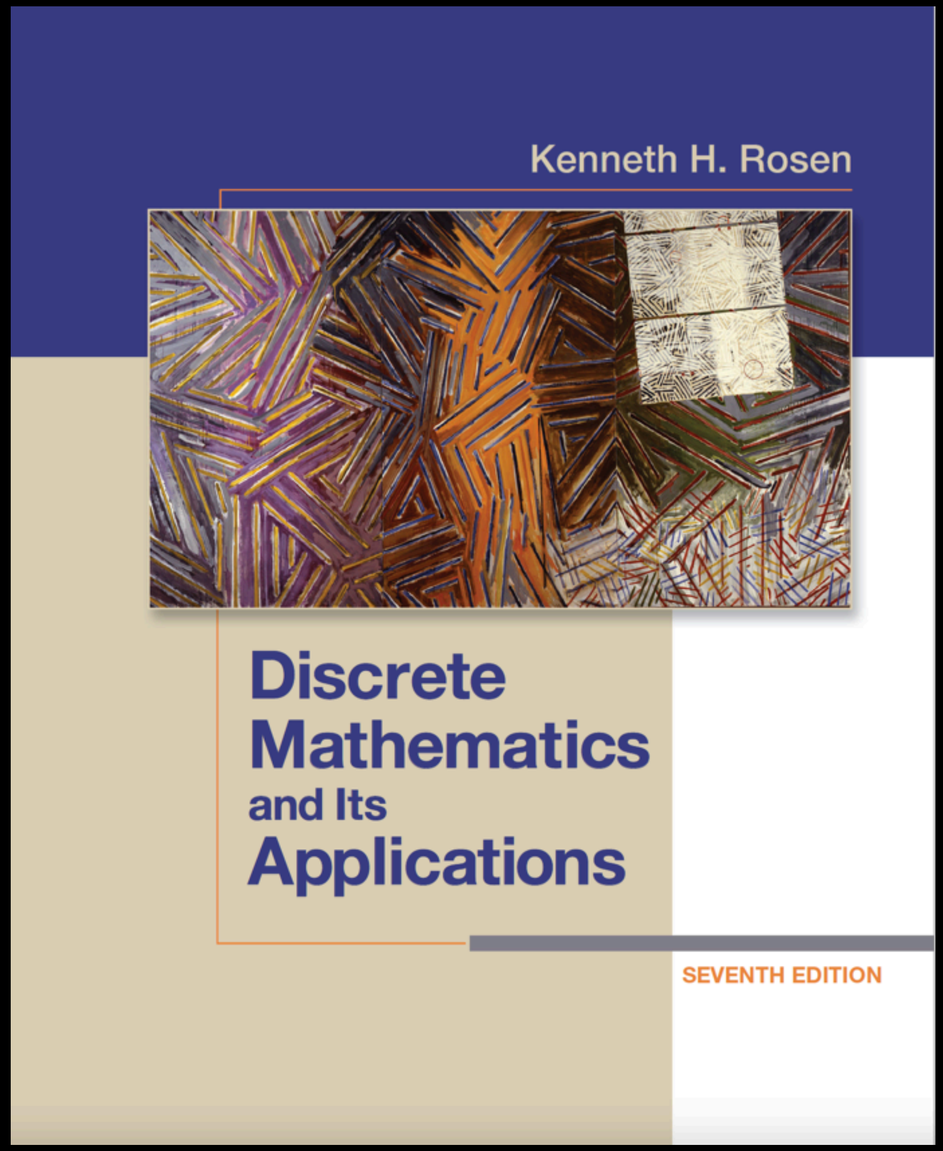
- Referred to as IA(CLRS)
- TOC available
- All the Mathematical Background (Appendix VIII)
  - pls 1143-1217
- Some course material is based on the book
- Use it for the Labs too.
- Course facilitators will refer you to read up some material in these books especially when the main course book falls short



Course Book IV

Discrete Mathematics and Its Applications (Rosen)

- Referred to as DM(RKH)
- TOC available
- Detailed mathematics text
- Some course material is based on the book
- Course facilitators will refer you to read up some material in these books especially when the main course book falls short

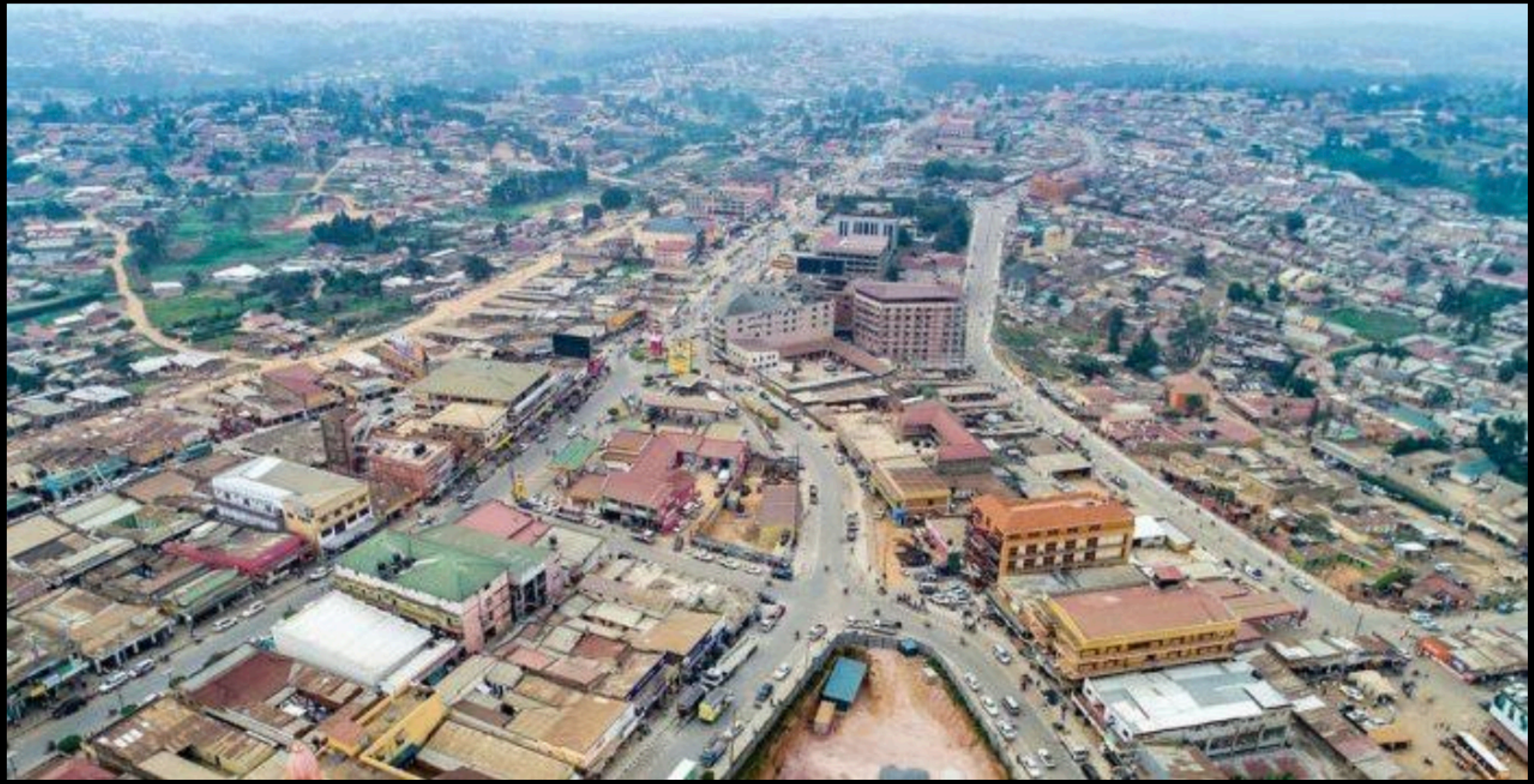


Questions?

Group Formation Activity

Introduction - Informal

Introduction - Informal Modelling and Abstraction




Pinnacle Hotel Mbarara, Kyamugorani, Mb
Mbarara University Of Science And Techn
Depart at
Options
8:35 AM
Wed, Jul 6
Send directions to your phone
via Mbarara Northern Bypass Rd and Mbarara - Bushenyi Rd/Mbarara - Kasese Rd
typically 16 min
Arrive around 8:51 AM
9.5 km
Details
via Mbarara Northern Bypass Rd and Kampala Road
typically 16 min
Arrive around 8:51 AM
9.6 km
via Mbarara Northern Bypass Rd
typically 16 min
Arrive around 8:51 AM
7.8 km
Explore Mbarara University Of Science And Technology
Collapse side panel
16 min
9.5 km
16 min
7.8 km
16 min
9.6 km
Pebu Bar Motel And Lounge
BHK pork joint
Joshua House
Hotel Triangle Mbarara
Hilltop Cottage
Kakika
Pinnacle Hotel Mbarara
Mbarara Northern Bypass Rd
Kibira bar
Wagga Resort Mbarara
Kosiya Hotel & Services Apartment
Nkokonjeru Tombs
Kampala Road
Buffalo Resort
InkSpot stationery and computer services
Mbarara town
Nyakizi Junc
Ntare Rd
KAKYEKA
KAKYEKA A
Mbarara
BOMA A
Baguma
Rolex Booth
Palace Hotel Mbarara
BOMA
BYAFULA ZONE
KAMUKUZI
KAMUKUZI HILL
Mbarara-Kasese Rd
Kabale-Mbarara Rd
KISWAHILI ZONE
BOMA B
Lake View Resort
Nile Breweries Mbarara. Ruharo
Nyamitanga
Mbarara University Of Science And Technology
KASHANYARAZI
KAKOBA DIVISION

Route finding and Planning

Real World

Computing World

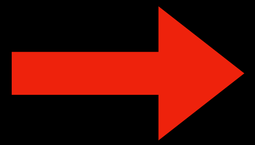
Graph

$G = (V, E)$

Modelling & Abstraction

Real World

- Objects
- Entities
- Activities / Actions /Events



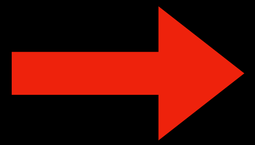
Computing World

- Data Structures
- Abstract Data Types, Classes
- Operations, Functions, Methods

Modelling

Real World

- Objects
- Entities
- Activities / Actions /Events



Computing World

- Data Structures
- Abstract Data Types, Classes
- Operations, Functions, Methods

- Problems: Are descriptions of objects with an *objective*
- Instances: are problems on a specific input
- Algorithms: are methods or procedures that solve instances of problems

Route finding and Planning

Real World → **Computing World**

Graph

$G = (V, E)$

Data Structures are adjacency matrices or adjacency lists (city connectedness)

Algorithms include: BFS, DFS, A*, UCS, Dijkstra's shortest path

Formal Definition of Algorithm

Algorithm

- An Algorithm ($\mathscr{A}$) is a finite sequence of **unambiguous** instructions for solving a problem with the following properties:
  - Input ($\mathscr{I}$): must have input values from a specified set
  - Output ($\mathscr{O}$): Given any ***instance*** ($I \subset \mathscr{I}$), $\mathscr{A}$ outputs a set of values that are the solution to the problem
  - Correctness: $\mathscr{A}$ should always produce correct solutions for any given input instance
    - Incorrect algorithms can be useful (such as hashing algorithms) as long as the arror rate can be controlled
  - Finiteness: $\mathscr{A}$ should eventually terminate
  - Effectiveness: $\mathscr{A}$ should execute using a "***reasonable***" amount of resources e.g time, memory, energy
  - Generality: $\mathscr{A}$ should be able to work on all instances of $\mathscr{I}$
- Note: $\mathscr{A}$ is actually the Input - Output relationship and can be defined as a function

Generalisations

Broad Categories

- Randomised algorithms
- Monte-carlo Algorithms
- Aproximation Algorithms
- Parallel algorithms
- Distributed Algorithms
- And many more!

General Techniques

- Brute force
- Divide & Conquer
- Decrease & Conquer
- Transform & Conquer
- Dynamic Programming
- Greedy Techniques

Specifying Algorithms

Pseudo Code

Algorithms can be specified using some form of pseudo-code

Good Pseudo-Code:

- Balances Clarity and detail
- Abstracts the algorithm
- Makes use of good mathematical notation
- Is easy to read

Bad pseudo-code:

- Gives too many details
- Is implementation or language

Good Psedo-Code Example I

INTERSECTION

```
    INPUT|    : Two sets of integers, A and B
    OUTPUT    : A set of integers C such that C = A ∩ B
1   C = ∅
2   FOR i = 1, ..., |A| DO
3   |   IF a_i ∈ B THEN
4   |   |   C = C ∪ {a_i}
5   |   END
6   END
7   output C
```

Good Psedo-Code Example II

A method for binary search:

```
boolean binarySearch(int[] arr, int item):
    int lo = 0
    int hi = arr.length - 1
    while lo ≤ hi:
        int mid = (lo + hi)/2
        if item < arr[mid]:
            hi = mid - 1
        else if item > arr[mid]:
            lo = mid + 1
        else:
            return true
    return false
```

Read Pseudo code cheat sheet on course website under learning path

Sorting Problem

- Because Sorting is a fundamental operation in computing,
  - There exist many algorithms for solving it
  - The algorithm chosen for a given application usually depends on
    - # of items to be sorted
    - the extent to which the input is already sorted
    - constraints on the input (special restrictions on the input instances)
    - the architecture of computer and kind of storage devices i.e memory, disks or tapes

Sorting Problem - Specification I

See CLRS pg 16-23 of the book but search in e-book pgs: 37-44. Reading chapter 1 pgs 5-15 is also worthwhile

- Input (I): A sequence of n objects $\langle a_1, a_2, \ldots, a_n \rangle$ e.g numbers such as
  - $\langle 31,41, 59,26, 41,58 \rangle$
- Output (O) : A permutation (reordering) $\langle a_1', a_2', \ldots, a_n' \rangle$ of I such that $a_1' \leq a_2' \leq \ldots \leq a_n'$ which is:
  - $\langle 26,31, 41,41, 58,59 \rangle$

Sorting Problem - Specification II

- IA (CLRS)

```
INSERTION-SORT(A)
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into the sorted sequence A[1 .. j − 1].
4      i = j − 1
5      while i > 0 and A[i] > key
6          A[i + 1] = A[i]
7          i = i − 1
8      A[i + 1] = key
```

Sorting Problem - Specification II

- IA (CLRS)


```
INSERTION-SORT(A)
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into the sorted sequence A[1 .. j − 1].
4      i = j − 1
5      while i > 0 and A[i] > key
6          A[i + 1] = A[i]
7          i = i − 1
8      A[i + 1] = key
```

Picking the card from the deck on a table

Sorting Problem - Specification II

- IA (CLRS)

```
INSERTION-SORT(A)
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into the sorted sequence A[1 . . j − 1].
4      i = j − 1
5      while i > 0 and A[i] > key
6          A[i + 1] = A[i]
7          i = i − 1
8      A[i + 1] = key
```

loop invariant

At the start of each iteration of the for loop of lines 1-8, the subarray A[1 . . *j*-1] consists of the elements originally in A[1 . . *j*-1] , but in sorted order.

Sorting Problem - Specification II

- This Course

```
List<X> Insertion-Sort ( List<X>  list):
  for j in range(1, len(list)):
    key = list[j]
    // insert list [ j ] into the sorted sequence list[ 0 .. j - 1 ]
    i =  j-1
    while i >  0 and list[ i ] > key:
      list[ i + 1] = list[ i ]
      i = i -1
      list[ i + 1]  = key
```

Loop invariant

Sorting Problem - Specification II

- IA (CLRS)

```
INSERTION-SORT(A)
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into the sorted sequence A[1 .. j − 1].
4      i = j − 1
5      while i > 0 and A[i] > key
6          A[i + 1] = A[i]
7          i = i − 1
8      A[i + 1] = key
```

Index *i* tracks the endpoint of the hand.

Sorting Problem - Specification II

- IA (CLRS)

```
INSERTION-SORT(A)
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into the sorted sequence A[1 .. j − 1].
4      i = j − 1
5      while i > 0 and A[i] > key
6          A[i + 1] = A[i]
7          i = i − 1
8      A[i + 1] = key
```

***i* tracks the endpoint of the hard.**

Loop Invariants

- Loop invariants help us understand why an algorithm is correct
- We must show three things about a loop invariant
  - **Initialisation** : Is it true prior to the first iteration of the loop?
  - **Maintenance:** If it is true before an iteration of the loop, does it remain true for the next iteration?
  - **Termination**: When the loop terminates, the invariant gives us a useful property that can help us prove the correctness of the algorithm

Loop Invariant - Insertion Sort

- When the first 2 properties hold, the loop invariant is true prior to every iteration of the loop
    - Notice the similarity to mathematical induction
        - To prove a property holds, you prove a base case and an inductive step
- Termination property differs from induction because we must stop as opposed to continuing ad infinitum
    - we must use both the inductive step and the condition that causes the loop to terminate.

Motivating Example II

A tale of two programs

Data Structure studied: Dynamic Arrays

Next Lecture

Motivating Example III

Applying Divide & Conquer on either Insertion sort or selection sort
Data Structure studied: Arrays

Computational Complexity

Introduction (JL), and more complex work