The Heap ADT

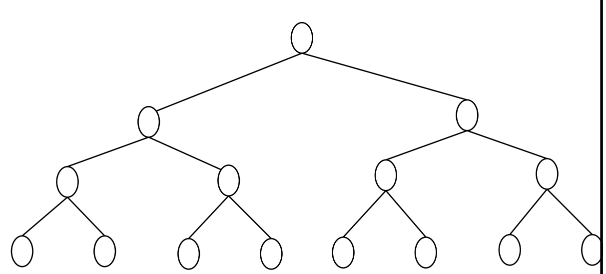
•In this section of notes you will learn about a new abstract data type, the heap, as well how heaps can be used.

A (Binary) Heap Is A Complete Binary Tree

•A complete binary tree of height *h* is full down to height *h – 1*.

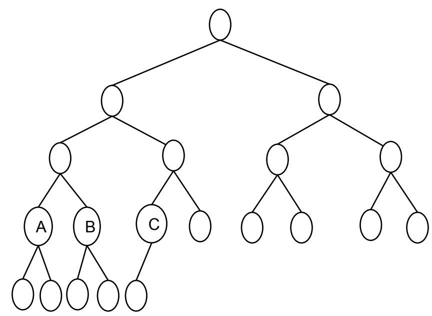
•Example:

- Height = 5
- Full from height = 1 to height = 4



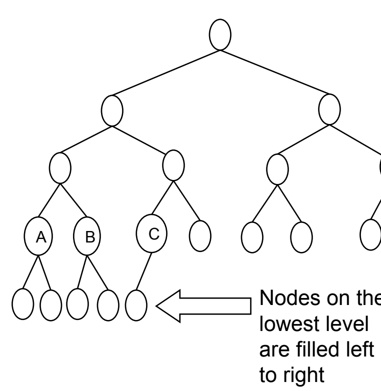
A (Binary) Heap Is A Complete Binary Tree

- A complete binary tree of height *h* is full down to height *h – 1*.
- Example:
  - Height = 5
  - Full from height = 1 to height = 4
  - Complete tree: When a node at height 4 has children all nodes at the same height and to it’s left have two children each
  - Complete tree: When a node at height 4 has one child it’s a left child



A (Binary) Heap Is A Complete Binary Tree

- A complete binary tree of height *h* is full down to height *h – 1*.
- Example:
  - Height = 5
  - Full from height = 1 to height = 4
  - Complete tree: When a node at height 4 has children all nodes at the same height and to it’s left have two children each
  - Complete tree: When a node at height 4 has one child it’s a left child



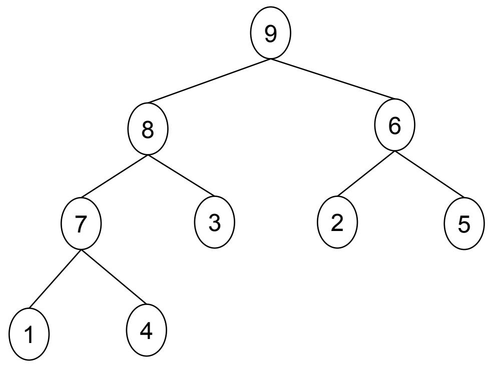
Complete Tree: General Specification

- A complete binary of height *h* is a binary tree that is full down to height *h* – 1 with height h filled in from left to right
  1. All nodes at *h* – 2 and above have two children each
  2. When a node at *h* – 1 has children, all nodes at the same height which are to the left of this node will each have two children.
  3. When a node at *h* – 1 has one child, it's a left child.

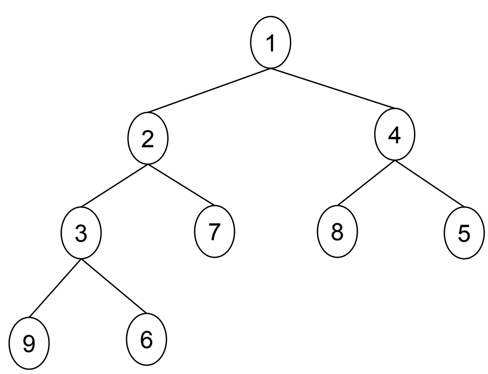
Heaps

1. A complete binary tree
2. Max heap (most common type of heap): The data in a parent node is greater than or equal to the it's descendent objects.
3. Min heap: The data in a parent node is lesser than or equal to the it's descendent objects.

Maxheap

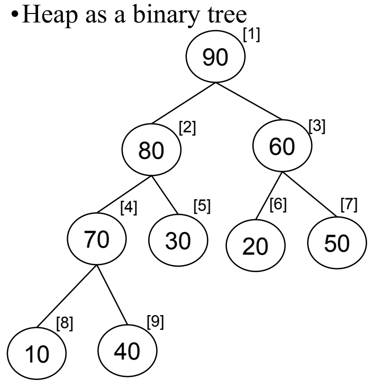


Minheap



Array Representation Of A Heap

- Heap as a binary tree



- Array representation of a heap:

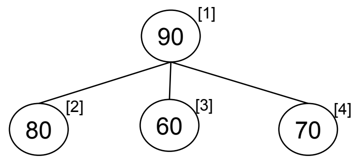
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]
	90	80	60	70	30	20	50	10	40			

Array Representation Of A Heap

- Recall: For a given node at index “I”:
  - The left child of that node will be at index = (I * 2)
  - The right child will be at index = (I * 2) + 1

Ternary Heaps (0, 1, 2, 3 Children)

- A heap as a ternary tree



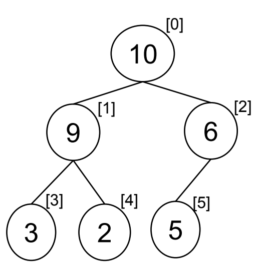
- Array representation of a trinary heap:

[0]	[1]	[2]	[3]	[4]
	90	80	60	70

- For a given index "I", the index of it's children:
  - First: (3 * I) – 1
  - Second: (3 * I)
  - Third: (3 * I) + 1

Alternative Indexing: Consider The Cases

Tree implementation



Array implementation

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
10	9	6	3	2	5			

Alternative Indexing: General Formula

- For a given node at index “I”:
  - The left child of that node will be at index = (2I + 1)
  - The right child will be at index = (2I + 2)

Methods Of Creating A Heap

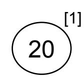
1. Ensure that the heap retains the property of a max/min heap as the heap is built.
2. Build the heap and then transform the heap into a max/min heap (“heapify” the heap).

Method 1: Add The First Element

•Array representation

[0]	[1]	[2]	[3]	[4]	[5]	[6]
	20					

•The corresponding tree

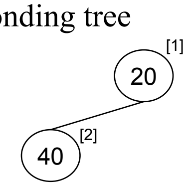


Method 1: Add The Second Element

•Array representation

[0]	[1]	[2]	[3]	[4]	[5]	[6]
	20	40				

•The corresponding tree

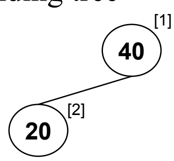


Method 1: Swap The First And Second Elements

•Array representation

[0]	[1]	[2]	[3]	[4]	[5]	[6]
	40	**20**				

•The corresponding tree

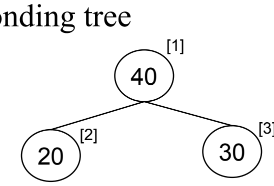


Method 1: Add The Third Element

•Array representation

[0]	[1]	[2]	[3]	[4]	[5]	[6]
	40	20	30			

•The corresponding tree

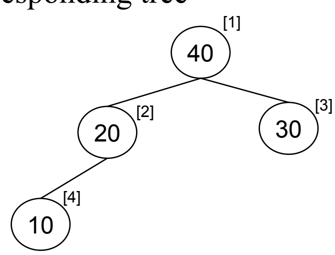


Method 1: Add The Fourth Element

•Array representation

[0]	[1]	[2]	[3]	[4]	[5]	[6]
	40	20	30	10		

•The corresponding tree

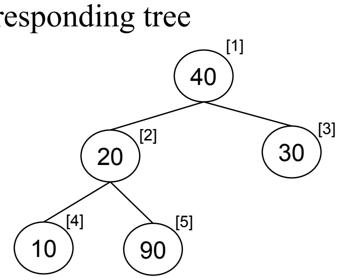


Method 1: Add The Fifth Element

•Array representation

[0]	[1]	[2]	[3]	[4]	[5]	[6]
	40	20	30	10	90	

•The corresponding tree

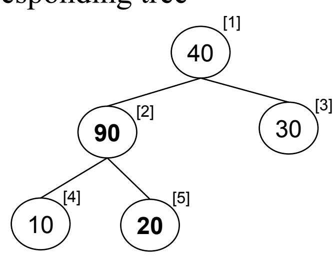


Method 1: Swap the Second And Fifth Elements

•Array representation

[0]	[1]	[2]	[3]	[4]	[5]	[6]
	40	**90**	30	10	**20**	

•The corresponding tree

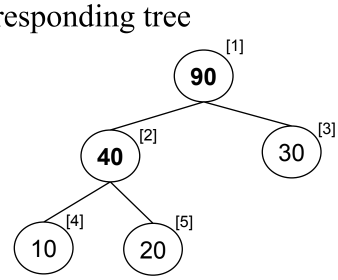


Method 1: Swap the First And Second Elements

•Array representation

[0]	[1]	[2]	[3]	[4]	[5]	[6]
	90	**40**	30	10	20	

•The corresponding tree

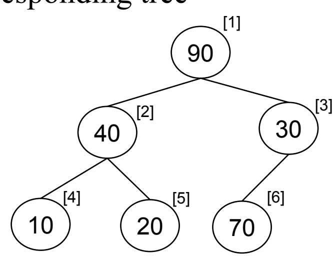


Method 1: Add The Sixth Element

- Array representation

[0]	[1]	[2]	[3]	[4]	[5]	[6]
	90	40	30	10	20	70

- The corresponding tree

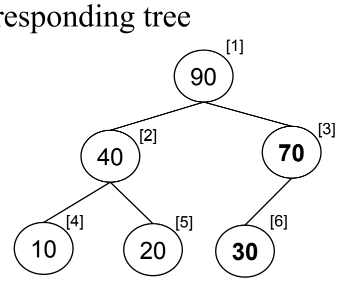


Method 1: Swap The Third And Sixth Elements

- Array representation

[0]	[1]	[2]	[3]	[4]	[5]	[6]
	90	40	**70**	10	20	**30**

- The corresponding tree

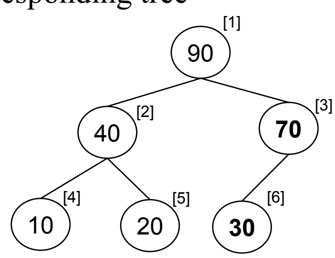


Method 1: The Final State Of The Heap

•Array representation

[0]	[1]	[2]	[3]	[4]	[5]	[6]
	90	40	70	10	20	30

•The corresponding tree

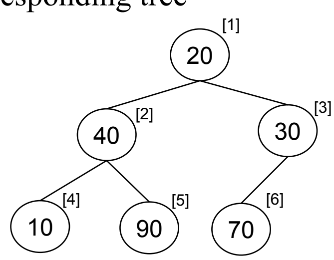


Method 2 For Building A Heap

•The information is read into an array

[0]	[1]	[2]	[3]	[4]	[5]	[6]
	20	40	30	10	90	70

•The corresponding tree

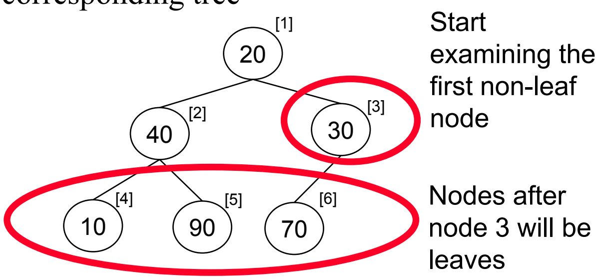


Method 2 For Building A Heap: Where To Start

•Start with node [⌊noNodes/2⌋], examine if the heap is a maxheap

[0]	[1]	[2]	[3]	[4]	[5]	[6]
	20	40	30	10	90	70

•The corresponding tree

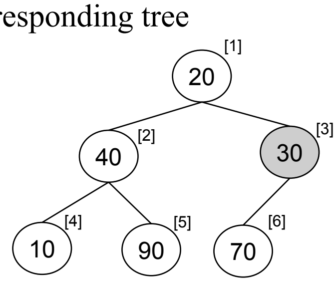


Method 2 For Building A Heap: Examine Element [3]

•The information is read into an array

[0]	[1]	[2]	[3]	[4]	[5]	[6]
	20	40	30	10	90	70

•The corresponding tree

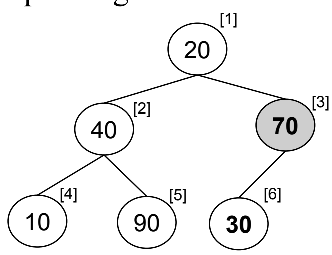


Method 2 For Building A Heap: Reheap Related To Element [3]

•The information is read into an array

[0]	[1]	[2]	[3]	[4]	[5]	[6]
	20	40	**70**	10	90	**30**

•The corresponding tree

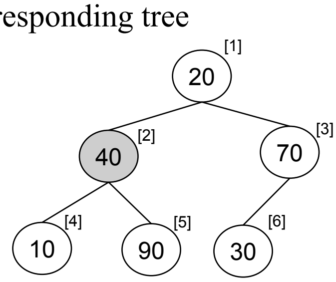


Method 2 For Building A Heap: Examine Element [2]

•The information is read into an array

[0]	[1]	[2]	[3]	[4]	[5]	[6]
	20	40	70	10	90	30

•The corresponding tree

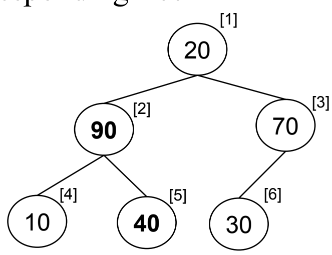


Method 2 For Building A Heap: Reheap Related To Element [2]

•The information is read into an array

[0]	[1]	[2]	[3]	[4]	[5]	[6]
	20	**90**	70	10	**40**	30

•The corresponding tree

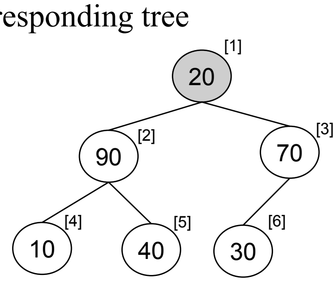


Method 2 For Building A Heap: Examine Element [1]

•The information is read into an array

[0]	[1]	[2]	[3]	[4]	[5]	[6]
	20	90	70	10	40	30

•The corresponding tree

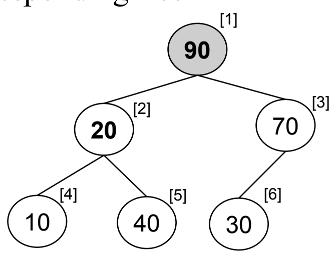


Method 2 For Building A Heap: First Reheap Related To Element [1]

•The information is read into an array

[0]	[1]	[2]	[3]	[4]	[5]	[6]
	90	**20**	70	10	40	30

•The corresponding tree

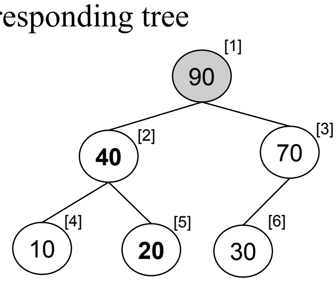


Method 2 For Building A Heap: Second Reheap Related To Element [1]

•The information is read into an array

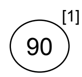
[0]	[1]	[2]	[3]	[4]	[5]	[6]
	90	40	70	10	20	30

•The corresponding tree



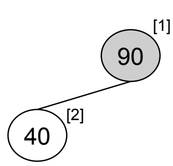
Determining The First Non-Leaf Element

- $\lfloor noNodes/2 \rfloor$ is first non-leaf node
- $\lfloor 1/2 \rfloor = 0$: No non-leaf nodes exist



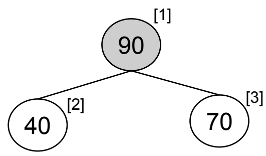
Determining The First Non-Leaf Element (2)

- $\lfloor noNodes/2 \rfloor$ is first non-leaf node
- $\lfloor 2/2 \rfloor = 1$



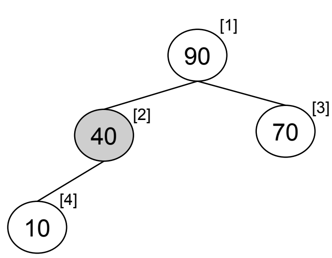
Determining The First Non-Leaf Element (3)

- $\lfloor noNodes/2 \rfloor$ is first non-leaf node
- $\lfloor 3/2 \rfloor = 1$



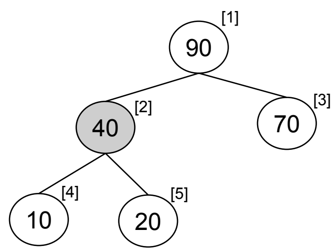
Determining The First Non-Leaf Element (4)

- $\lfloor noNodes/2 \rfloor$ is first non-leaf node
- $\lfloor 4/2 \rfloor = 2$



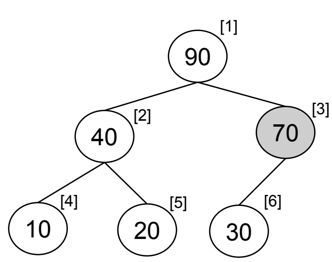
Determining The First Non-Leaf Element (5)

- $\lfloor noNodes/2 \rfloor$ is first non-leaf node
- $\lfloor 5/2 \rfloor = 2$



Determining The First Non-Leaf Element (6)

- $\lfloor noNodes/2 \rfloor$ is first non-leaf node
- $\lfloor 6/2 \rfloor = 3$



Priority Queue

- Review: Regular queues (FIFO)
  - Elements are de-queued according their order of arrival

Front: Exit queue **Back: Enter queue**

- Priority queue:
  - Elements are removed according to their priority level.

1 1 1 2 3

Priority Queue

- Review: Regular queues (FIFO)
  - Elements are de-queued according their order of arrival

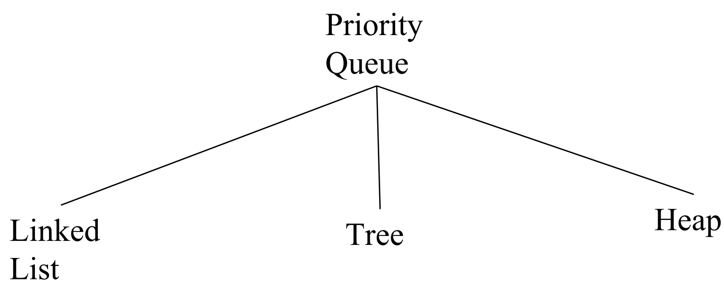
Front: Exit queue **Back: Enter queue**

- Priority queue:
  - Elements are removed according to their priority level.

1 1 1 2

3

Priority Queue: ADT



Linked List Implementations Of A Priority Queue

- Sorted[1]

4	3	2	1	1	1

  - Insertion: *O*(n)
  - Remove: *O*(1)

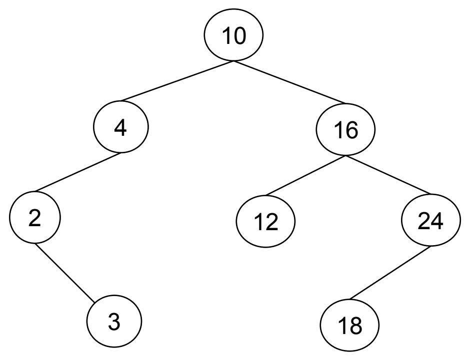
- Unsorted

2	1	3	1	4	1

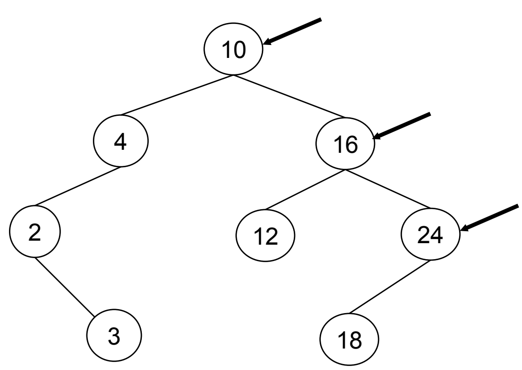
  - Insertion: *O*(1)
  - Remove: *O*(n)

1 Sorted in this case refers to maintaining the list in order but it is done by in-order insertions rather than applying a sorting algorithm.

Binary Search Tree Implementation Of A Priority Queue



Binary Search Tree Implementation Of A Priority Queue: Remove Largest Element



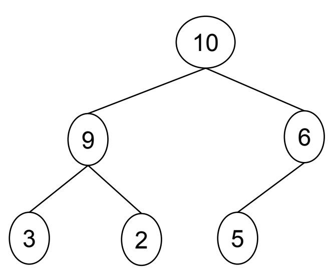
Efficiency Of The Binary Search Tree Implementation Of A Priority Queue

Operation	Average case	Worse Case[1]
Insertion	O ($\log_2$n)	O (n)
Deletion	O ($\log_2$n)	O (n)

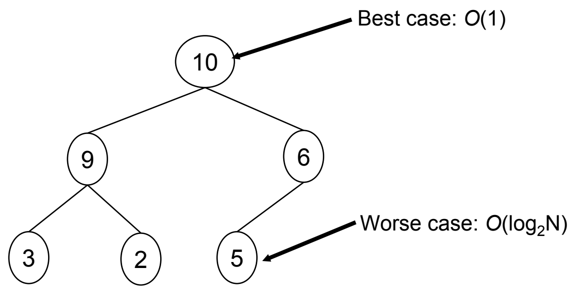
The worse case is always a possibility unless a self-balancing tree implementation (e.g., AVL tree) is employed

Heap Implementation Of A Priority Queue

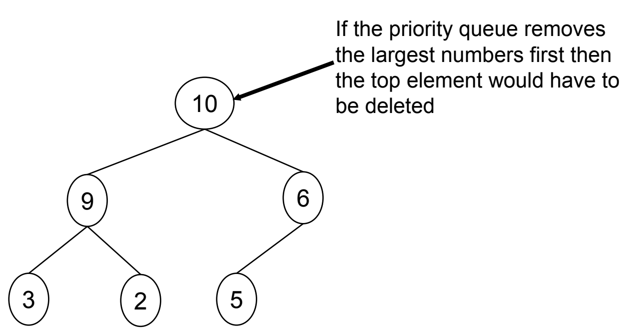
- Example: A Maxheap implementation



Heap Implementation: Insertions

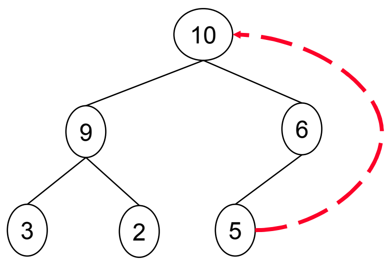


Heap Implementation: Deletions



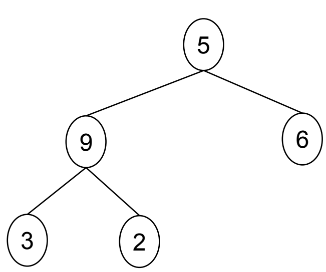
Heap Implementation: Deletions (2)

•Move the last element to the top of the heap



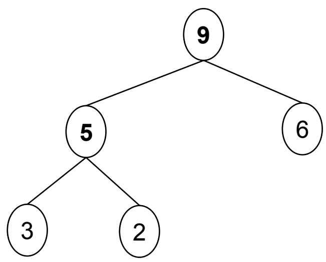
Heap Implementation: Deletions (3)

•Move the last element to the top of the heap



Heap Implementation: Deletions (4)

•The element at the top "trickles down" to it's proper location in the tree via a swap or series of swaps.



Efficiency Of Deletions From a Heap

•Trickling down the top element to it's proper place:

- When the element must be moved the height of the tree: $O(\log_2 N)$

You Should Now Know

- What is a heap / complete tree?
- The difference between the categories of heaps:
  - Min vs. max heaps.
  - Binary and ternary heaps.
- What types of data structures can be used to implement heaps?
- How to build a heap using two different approaches.
- The different ways in which a priority queue can be implemented and the efficiency of each approach?

Sources Of Lecture Material

- *"Data Abstraction and Problem Solving With Java: Walls and Mirrors"* updated edition by Frank M. Carrano and Janet J. Prichard
- *From "Data Structures and Abstractions with Java*" by Frank M. Carrano and Walter Savitch.
- "*Introduction to Algorithms*" by Thomas M. Cormen, Charles E. Leiserson and Ronald L. Rivest.
- Course notes by Claudio T. Silva http://www.cs.utah.edu/classes/cs3510-csilva/lectures/
- CPSC 331 course notes by Ken Loose http://pages.cpsc.ucalgary.ca/~marina/331/