# Algorithms and Data Structures

*Notes on Algorithms, Data Structures and other concepts*

ERIC LI

February 9, 2019

# Contents

# 1 Disclaimer

The author(s) of this document assume(s) no responsibility or liability for any errors or omissions in the content of this document. The information contained in this site is provided on an "as is" basis with no guarantees of completeness, accuracy, usefulness or timeliness or of the results obtained from the use of this information.

Information provided in this document has been taken from various sources and is by no means a comprehensive record of the source's views nor of the concepts represented.

# 2 Trees

## 2.1 Structure

### 2.1.1 Terminology

Node

# 3 Linked Lists

Linked lists are helpful in addressing some of the limitations that arrays face, such as inherent dynamic memory allocation and ease of modification. Instead of memory being assigned for use (as in the programming language C) linked lists are built on the premise of allocating memory as needed, and inserting or deleting as required by the user or data set.

## 3.1 Structure

A linked list is a linear data structure constructed by building blocks called nodes. Like some other data structures, the node contains two items: data, which can be simple, compound, pointer, etc. and an address reference (or link to the next node). The address reference will access the next node's memory address in the list, and thus creates a linear structure

There are two types of nodes:

- Head Node
  - Head insertion is much quicker due to the lack of list traversing done
- Tail Node
  - Has an address reference of NULL
  - Entire list must be traversed before tail insertion can occur

### 3.2 Basic Implementation

1. Define a node

```
1    typedef struct int_node{
2        int stored_int; // data
3        struct int_node *ref; // address
4    } int_node;
```

2. Dynamic creation of nodes

## 4 Sources

https://www.cs.cmu.edu/ adamchik/15-121/lectures/Linked