**Nabeel Hussain**

**CMSC 204**

**Study Guide - Big O Notation**

Give a brief description of the following concepts:

* field, node. Key

A **field** is a piece of data, such as a person name, age, address, or phone number. It is indivisible, meaning that it would lose its meaning if the data in the field were to be divided up further.

A **node** is a group that contains related fields. For example, if a program wanted to know the physical characteristics of a person, then a single person would be the node, and its related fields would be the person’s name, height, weight, eye color, hair color, etc…

A **key field** is the designated field(s) in a node that is used to fetch or identify the node. For example, a person’s name would make the most sense in terms of identifying them, rather than how tall they are or how old they are.

* procedural abstraction, data abstraction

**Procedural abstraction** is the term that’s used to describe the use of methods or procedures, without necessarily knowing how they are implemented or constructed. We just need to know what the method does in order to use it, that way we can integrate their functionality into our programs.

**Data abstraction** is the term that’s used to describe the use of data structures, without knowing their implementation details. As long as we generally know what the operation methods do and their signatures, then we will be able to know when and how to use them if necessary.

* encapsulation

**Encapsulation** is the term used to describe the process of restricting access to certain data in your code, to prevent it from being accessed and changed from anywhere. This is very important when working with large programs, because if there is an error that occurs in your code, and you need to access a certain variable or piece of data, which was not encapsulated, then it would be extremely time consuming to find and resolve the issue.

* Fundamental Operations performed on Data Structures

There are four main fundamental operations performed on data structures, which are the **Insert, Delete, Fetch, and Update** operations. The most fundamental of them all is the *insert* function, because it’s used to add nodes to the data structure, and without this operation all data sets would be empty. The *fetch* and *delete* operations are at the next level of the operation hierarchy, in which they return nodes from a data set or delete them, respectively. The *update* operation is another level above the *fetch* and *delete* operations, and it is used to change the content or information of fields in an existing node.

* Operation access modes

The fundamental operations performed on Data Structures can be performed in either the node number or key field access mode. With the exception of the Fetch method, all of the methods return a Boolean value that is set to false if they cannot complete their operation. The fetch method returns a reference to the requested node or a null reference if the node requested is not in the data set.

* speed complexity

This is the term used to describe the speed of an algorithm, or the time it takes to execute. It is expressed as a mathematical function T(n), where n is usually the number of pieces of data the algorithm processes.

* space complexity

This is the term used to describe the storage requirements of an algorithm when analyzing the algorithm for its speed and efficiency.

* Big-O: relative and absolute speed

Big-O analysis is a technique used to approximate the value of a function at a large value of the independent variable. As a result, we can simply just evaluate the functions dominant term and neglect all the other terms

The term **relative speed** is used to determine whether an algorithm is faster or slower than other algorithms.

The term **absolute speed** is used to determine the actual execution time, in seconds, of an algorithm. This is much more difficult to calculate, because it is dependent on many other parameters that must be taken into account.

Big-O analysis can simplify the analysis performed to determine both of these speeds, however for the absolute speed, this type of analysis can invalidate its use in calculations for time-critical applications.

* density of a data structure

**The density of a data structure is used to describe how efficiently a data structure uses memory. The one that requires the least amount of overhead (extra memory to store a nodes information), is the most desirable and more efficient.**

**D = information bytes / total bytes**

**= information bytes / (information bytes + overhead bytes)**

**Density is always between 0 and 1, depending on the amount of overhead memory, and the larger the overhead, the closer the density is to 0. As a result, the more efficient data structures have a density closer to 1.**