## Data structures

* Programmatic way of organizing & storing data so that it can be used efficiently

## Hashing

* Transformation of string characters into a shorter fixed length key or value that represents the original string
* Used to index and retrieve items from database because it is faster to find them using shorter key

## HashCode

* A number generated from any object
* This number allows objects to be stored/retrieved quickly in Hashtables

## Abstract Data Types:

* Entities that are definitions of data & operation but no implementations
* List – Collection of objects
* Set, SortedSet, NavigableSet
* Map

## Concrete Data Types :

* Implementations of abstract data types
* ArrayList , LinkedList , Doubly Linkedlist
* HashSet, LinkedHashSet,TreeSet, – All implements Set Interface
* HashMap , LinkedHashMap (preserves insertion order), TreeMap

### Big O Notation

* Used to describe the time complexity of operations in a collection
* Constant time - O(1)

### Array

* Data is store in one contiguous block in Memory
* Static, cannot increase size
* Accessing an element can be done in constant time

## List

* Accessing an element (Read/Write) in any index is constant - O(1)
* Add/Insert/Delete/ – Time taken is proportional to the size (Linear) - O(n)
* List is ordered the way it is stored
* Accepts duplicate elements
* Not a good use of Memory
* Two types (ArrayList & LinkedList)

## Array List

* Data is store in one contiguous block in Memory

## Linked List

* Represent data in nodes. A node has the data and address of next node

Struct Node{

*Int data; // 4 bytes in typical architecture*

*Node\* next; // 4 bytes in typical architecture*

}

* The first node is called head node
* The nodes are non-contiguous and connected
* Access to the address of head node yields the complete data in the list
* Address block in the last node is null/zero (This means there are no more elements)
* Only way to access the elements in a Linkedlist is by accessing / traversing each node
* Unlike array, we cannot access an element in constant time (Time complexity is propagational to size of the element) - O(n)
* Unlike array, inserting/deleting is simple, we just need to create a node and link it properly – O(n)

## Arrays vs Linked List:

|  |  |  |  |
| --- | --- | --- | --- |
| **Criteria** | **Array** | **Linked List** | **Binary Search Tree** |
| Search - Cost of accessing an element | O(1) Constant time since it’s a contiguous block of memory – Use array if you use the elements all the time. | O(n) – Need to traverse the list | O(log n) |
| Memory requirements | We need to know the size before creating. Size is fixed, so memory is unused in some cases | No unused memory. No reserved space like array, however we need extra memory for storing the address of next element. Easy to get memory to store large data as there is no need to get contiguous block of memory |  |
| Cost of inserting an element at the beginning (Time complexity) | O(n) – Proportional to the size of the array as we need to shift elements | O(1) – Constant , just need to create a new node and link it | O(log n) |
| Remove - Cost of inserting an element at the end | O(1) – If array has some space  O(n) – If array is full as we must create new array and copy contents to new array | O(n) – Need to Traverse till the end and create node adjusting the link | O(log n) |
| Cost of inserting an element at the ith position (in between) | O(n) - Need to shift elements | O(n) – Need to traverse till the ith position |  |
| Cost of deleting an element | O(n) | O(n) |  |
|  |  |  |  |

## Queue

* Implements first in first out

## Priority Queue

* Used to sort collection

## Legacy Classes in Java

1. HashTable
   * Similar to HashMap, but it is synchronized
2. Vector – Similar to ArrayList, but it is synchronized
3. Stack –
   * Stack is an area in the memory that holds all local variables & parameters used in a program
   * Extends Vector
   * Implements last in first out
4. Properties –
   * Extends HashTable
   * Both key & value must be String
   * Used to represent persistent set of properties (System properties)
   * Provides an ability to assign a default value when there is no value found for a key
5. Dictionary

## Set

* Will not accept duplicates
* Will not preserve insertion order (Elements cannot be found by Index)

## SortedSet

* Will not accept duplicates
* Elements are sorted based on the sorting order provided

|  |  |
| --- | --- |
| HashSet | HashMap |
| Only one Null Key is allowed | One null key and multiple null values accepted |
| Synchronized, slow | Not, so not thread safe, faster |
| No null accepted in key or value | Accepts one Null Key and many null values |
| Use enumerator | Uses Iterator for iteration |
| Do not maintain order | Do not maintain order |
|  |  |

|  |  |  |
| --- | --- | --- |
| HashSet | LinkedHashSet | TreeSet |
| Will no preserve insertion order | Maintains insertion Order | Ordered based on natural order |
| Better Performance | Performance in between HashSet & TreeSet |  |
| Uses HashMap | Uses LinkedHashMap | Use TreeMap |
| Insert, Remove & Retrieval – O(1) | Insert, Remove & Retrieval – O(1) | OLog(n) |

## HashMap: HashMap is implemented as a **hash table**, and there is no ordering on keys or values.

TreeMap: TreeMap is implemented based on **red-black tree structure**, and it is ordered by the key

LinkedHashMap: LinkedHashMap preserves the insertion order. Hashtable is synchronized, in contrast to HashMap.

## Differences of HashTable HashMap & TreeMap

|  |  |  |
| --- | --- | --- |
| HashMap | LinkedHashMap | TreeMap |
| Not, so not thread safe, faster | Maintains insertion Order | Ordered by Key/ Ordered based on natural order |
| Accepts one Null Key and many null values |  |  |
| Uses Iterator for iteration |  |  |
| Do not maintain order |  |  |
|  |  |  |

## Differences of Enumerator & Iterator

|  |  |
| --- | --- |
| Enumerator | Iterator |
| Used only to traverse elements in a collection | In addition to traverse, used to remove elements |
|  | Advanced version of Enemurator |

## Similarities of HashTable & HashMap

* Order will not be maintained
* Both implements Map interface
* Same time for put / get methods
* Both works based on the principle of hashing

## Heaps

* An array which can be visualized as a tree (Tree based data structure)

## Red-black Tree

* It is a self balancing binary search tree
* Each node of the binary tree has an extra bit which is represented as color (red or black) of the node
* These color bits are used to ensure the tree is balanced during insertions and deletions

## Binary Tree

How to Balance a Binary Tree

What is height balanced Tree