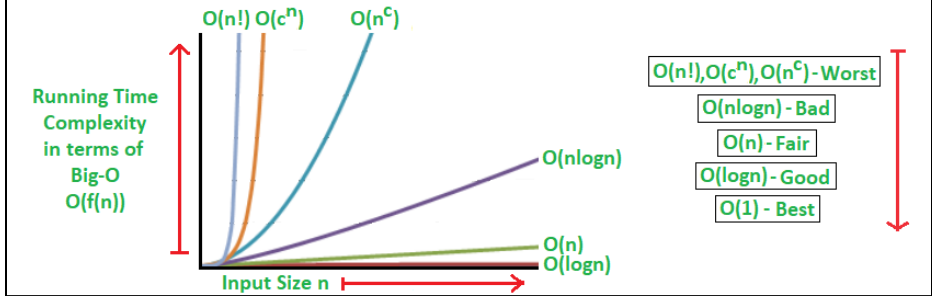


## Primitive Data Type

<u>Type</u>	<u>Data Type</u>
Logical	boolean(true/false)
Textual	char(16-bit Unicode character)
Integers	byte(1 byte), short(2 bytes), int(4 bytes) and float(4 bytes)
Real numbers	double(8 bytes) and float(4 bytes)

## Introduction to Object-oriented programming(OOP)

<u>Keywords</u>	<u>Explanation</u>
Classes	= a model for creating objects. <ul style="list-style-type: none"><li>• objects (also called “instances”) can be created after the class is defined.</li><li>• a program can only contain one copy of each class, but can have many objects.</li></ul>
Objects	= entities other than the primitive data type e.g. String, Scanner, File, PrintWriter, Exceptions, Arrays, etc.
The Above have the following components: <ul style="list-style-type: none"><li>• State Data - variables inc. instance variables(attributes) and local variables.</li><li>• Behavior - methods.</li></ul>	
Attributes (instance variables)	= variables that belong to a class. <ul style="list-style-type: none"><li>• created when a class/object is created.</li><li>• declared outside of any methods.</li><li>• can be accessed by any method of the object(or even other objects if it is public).</li></ul>
Local variables	= variables that declared inside methods. <ul style="list-style-type: none"><li>• created each time when the method is called.</li><li>• destroyed when exited from an object.</li><li>• can be used inside the method only.</li></ul>
Methods	= operations that an object can perform: <ul style="list-style-type: none"><li>• Update its own attributes.</li><li>• Perform some computations.</li><li>• Perform I/O operations.</li><li>• Call another object's method.</li></ul> Can be declared as public or private. <ul style="list-style-type: none"><li>• public - the variable, or method is visible to objects of any class, anywhere.</li><li>• private - the variable or method can be accessed only by objects in the same class as the current object.</li></ul>

	<ul style="list-style-type: none"> <li>• default - the variable, or method can be accessed by any class in the same package.</li> </ul>
[Extra] Big-O Notation	<ul style="list-style-type: none"> <li>• Constant-time function/method = <math>O(1)</math></li> <li>• Linear-time function/method = <math>O(n)</math></li> <li>• Quadratic-time function/method = <math>O(n^2)</math></li> <li>• Etc.</li> </ul> 

### Arrays (Vectors/Tuples in mathematics)

Aspects	Contents																									
Definition	= a linear collection of items stored at contiguous memory locations.																									
Characteristics	<ul style="list-style-type: none"><li>• Items are of the same type.</li><li>• Position is calculated by adding an <u>offset</u> to the base value (i.e. first address).<ul style="list-style-type: none"><li>◦ Memory is statistically allocated when declared (i.e. no expanding or shrinking).</li></ul></li></ul>																									
Complexity	<table><tr><td colspan="5">Time complexity</td></tr><tr><td></td><td>Access</td><td>Search</td><td>Insertion</td><td>Deletion</td></tr><tr><td>Best</td><td>O(1)</td><td>O(1)</td><td>O(1)</td><td>O(1)</td></tr><tr><td>Average</td><td>O(1)</td><td>O(N)</td><td>O(N)</td><td>O(N)</td></tr><tr><td>Worst</td><td>O(1)</td><td>O(N)</td><td>O(N)</td><td>O(N)</td></tr></table> <p>Memory complexity = O(1) [constant once declared]</p>	Time complexity						Access	Search	Insertion	Deletion	Best	O(1)	O(1)	O(1)	O(1)	Average	O(1)	O(N)	O(N)	O(N)	Worst	O(1)	O(N)	O(N)	O(N)
Time complexity																										
	Access	Search	Insertion	Deletion																						
Best	O(1)	O(1)	O(1)	O(1)																						
Average	O(1)	O(N)	O(N)	O(N)																						
Worst	O(1)	O(N)	O(N)	O(N)																						
Advantages	<ol style="list-style-type: none"><li>1. Allow random access to elements, faster accessing time.</li><li>2. Have better cache locality (i.e. the tendency to access the same set of memory locations repetitively over a short period of time.).</li><li>3. Represent multiple data items of the same type using a single name.</li></ol>																									
Limitations	<ol style="list-style-type: none"><li>1. Unchangeable size to prevent the risk of overwriting other data.</li><li>2. Predetermined memory space according to array size.</li><li>3. Move half of the elements for each insertion or deletion on average.</li></ol>																									

Examples	<ul style="list-style-type: none"> <li>• Used to implement data structures like a stack, queue, etc.</li> <li>• Used for matrices and other mathematical implementations.</li> <li>• Used in lookup tables in computers.</li> <li>• Can be used for CPU scheduling.</li> </ul>
----------	--

### Linked List

Aspects	Contents																				
Definition	= a linear collection of items stored and linked using pointers.																				
Characteristics	<ul style="list-style-type: none"><li>• Represented by a pointer to the first node(the head) of the linked list.</li><li>• Each node consists of two parts:<ul style="list-style-type: none"><li>◦ data (store integer, strings or any type of data).</li><li>◦ pointer/reference to the next node.</li></ul></li></ul>																				
Complexity	<div>Time complexity<table><tr><th></th><th>Access</th><th>Search</th><th>Insertion</th><th>Deletion</th></tr><tr><td>Best</td><td>O(1)</td><td>O(1)</td><td>O(1)</td><td>O(1)</td></tr><tr><td>Average</td><td>O(N)</td><td>O(N)</td><td>O(1)</td><td>O(1)</td></tr><tr><td>Worst</td><td>O(N)</td><td>O(N)</td><td>O(1)</td><td>O(1)</td></tr></table></div> <div>Memory complexity = O(n)</div>		Access	Search	Insertion	Deletion	Best	O(1)	O(1)	O(1)	O(1)	Average	O(N)	O(N)	O(1)	O(1)	Worst	O(N)	O(N)	O(1)	O(1)
	Access	Search	Insertion	Deletion																	
Best	O(1)	O(1)	O(1)	O(1)																	
Average	O(N)	O(N)	O(1)	O(1)																	
Worst	O(N)	O(N)	O(1)	O(1)																	
Advantages	<ol style="list-style-type: none"><li>1. Dynamic size and runtime memory allocation.</li><li>2. Useful when the size is uncertain and large variation in array sizes.</li><li>3. Ease of insertion and deletion.</li></ol>																				
Limitations	<ol style="list-style-type: none"><li>1. No random access and thus binary search.</li><li>2. Extra memory space for a pointer.</li><li>3. Not cache friendly as no locality of reference.</li></ol>																				
Variations	<ul style="list-style-type: none"><li>• Single Linked List = one can move or traverse the linked list in only one direction.</li><li>• Doubly Linked List = one can move or traverse the linked list in both directions.</li><li>• Circular Linked List = the last node contains the link of the first/head node, vice versa (in its previous pointer).</li></ul>																				
Examples																					

### Stacks (Through Arrays or LinkedList)

Aspects	Contents
Definition	= linear order data structures that store objects in a default LIFO (Last In First Out) structure.

Characteristics	<p>Main operations/methods:</p> <ul style="list-style-type: none"><li>● Push() = put a new value on the top of a stack.</li><li>● Pop() = take a value from the top of a stack.</li><li>● Peep() = return at the top element without removing it.</li><li>● IsEmpty() = check if the stack is empty and returns true/false.</li><li>● Size() = return the size of the stack.</li></ul> <p>Reverse Polish Notation (RPN) = all operators are expressed after the second operands.</p>																								
Complexity	<p>Time complexity</p> <table><tr><td></td><td>Access</td><td>Search</td><td>Insertion</td><td>Deletion</td></tr><tr><td>Best</td><td>O(1)</td><td>O(1)</td><td>O(1)</td><td>O(1)</td></tr><tr><td>Average</td><td>O(N)</td><td>O(N)</td><td>O(1)</td><td>O(1)</td></tr><tr><td>Worst</td><td>O(N)</td><td>O(N)</td><td>O(1)</td><td>O(1)</td></tr></table> <p>Space complexity = O(n)</p>						Access	Search	Insertion	Deletion	Best	O(1)	O(1)	O(1)	O(1)	Average	O(N)	O(N)	O(1)	O(1)	Worst	O(N)	O(N)	O(1)	O(1)
	Access	Search	Insertion	Deletion																					
Best	O(1)	O(1)	O(1)	O(1)																					
Average	O(N)	O(N)	O(1)	O(1)																					
Worst	O(N)	O(N)	O(1)	O(1)																					
Advantages	N/A																								
Limitations	N/A																								
Examples																									

## Queues

<u>Aspects</u>	<u>Contents</u>
Definition	= linear data structures that store objects in a FIFO (First In First Out) structure.
Characteristics	Components <ul style="list-style-type: none"> <li>• Front/Head = the first entry at the “bottom” of the queue, that would be removed first.</li> <li>• Rear/Tail = the last entry at the “top” of the queue, that is most recently added.</li> </ul> Main operations/method <ul style="list-style-type: none"> <li>• Enqueue() = insert a new element at the rear of a queue.</li> <li>• Dequeue() = delete and returns the element at the front of a queue.</li> <li>• Front() = get the front item from the queue.</li> <li>• Rear() = get the last item from the queue.</li> <li>• IsEmpty() = check if the queue is empty and returns true/false.</li> <li>• IsFull() = check if the queue is full and returns true/false.</li> </ul> Types in Java = ArrayBlockingQueue, PriorityQueue, LinkedList.

Complexity	Time complexity				
		Access	Search	Insertion	Deletion
	Best	O(1)	O(1)	O(1)	O(1)
	Average	O(N)	O(N)	O(1)	O(1)
	Worst	O(N)	O(N)	O(1)	O(1)
	Space complexity = O(n)				
Advantages	<ol style="list-style-type: none"> <li>1. Can Handle multiple data.</li> <li>2. Can be accessed at both ends.</li> <li>3. Fast and flexible.</li> </ol>				
Limitations	N/A				
Variations	<ul style="list-style-type: none"> <li>• Input Restricted Queue = input can be taken from only one end, but deletion can be done from any of the ends.</li> <li>• Output Restricted Queue = input can be taken from both ends, but deletion can be done from only one end.</li> <li>• Circular Queue = a special type where the last position is connected back to the first position, but performed in FIFO order.</li> <li>• Double-Ended Queue (Deque) = both the insertion and deletion operations can be performed from both ends.</li> <li>• Priority Queue = a special queue where the elements are accessed based on the priority assigned to them.</li> </ul>				
Examples	<ul style="list-style-type: none"> <li>• Has a single resource and multiple consumers.</li> <li>• Synchronize between slow and fast devices.</li> </ul>				

## Hash Tables

Aspects	Contents
Definition	= an improved direct access table with hash function and key mapping.
Characteristics	<p>Hash functions</p> <ul style="list-style-type: none"> <li>• Hash function = a function that transfers a given key into a specific slot index. <ul style="list-style-type: none"> <li>○ No control on the order and position for the stored entities.</li> <li>○ Hashcode = hashcode*MULTIPLIER + (int)[String].charAt([index]) <ul style="list-style-type: none"> <li>■ The initial value of the hashcode = 0.</li> <li>■ MULTIPLIER is usually a power of 2 for faster computation.</li> <li>■ return hashcode % ([tableSize]-1)</li> </ul> </li> <li>○ Generate a hash code between 0 and tableSize-1.</li> </ul> </li> <li>• Good hash function requirements: <ul style="list-style-type: none"> <li>○ Efficiently computable.</li> </ul> </li> </ul>

	<ul style="list-style-type: none"><li>○ Should uniformly distribute the keys.</li><li>○ Should minimize collisions.</li><li>○ Should have a low load factor (number of items in table divided by size of the table).</li></ul> <p>Basic operations:</p> <ul style="list-style-type: none"><li>● HashTable() = used for creating a new hash table.</li><li>● Delete() = used for deleting a particular key-value pair from the hash table.</li><li>● Get() = used for searching a key inside the hash table and returning the associated value.</li><li>● Put() = used for inserting a new key-value pair inside the hash table.</li><li>● DeleteHashTable() = used for deleting the hash table.</li></ul> <p>Collision handling</p> <ul style="list-style-type: none"><li>● Collision handling = situation where a newly inserted key maps to an already occupied slot.<ul style="list-style-type: none"><li>○ Chaining = make each cell of a hash table point to a linked list of records that have the same hash function value.<ul style="list-style-type: none"><li>■ buckets = the memory location represented by hash codes in a hash table.</li><li>■ by using the overloading method.</li></ul></li><li>○ Open Addressing = store all elements in the hash table, each entry then contains either a record or NIL.</li></ul></li><li>● Load Factor <math>\lambda = N_{\text{entries}} / N_{\text{buckets}}</math><ul style="list-style-type: none"><li>○ Too few buckets (large Load Factor) would lead to more collisions.</li></ul></li><li>● Enumeration = a complete, ordered listing of all the items in a collection.<ul style="list-style-type: none"><li>○ use hasMoreElements() and nextElement() functions <u>to loop through all keys</u>.</li></ul></li><li>● Rehashing = Increase the <math>N_{\text{buckets}}</math> and re-enter all keys when the table is too compacted.<ul style="list-style-type: none"><li>○ is done automatically in Java.</li></ul></li></ul>																				
Complexity	<p>Time complexity</p> <table><tr><td></td><td>Access</td><td>Search</td><td>Insertion</td><td>Deletion</td></tr><tr><td>Best</td><td>O(1)</td><td>O(1)</td><td>O(1)</td><td>O(1)</td></tr><tr><td>Average</td><td>O(1)</td><td>O(1)</td><td>O(1)</td><td>O(1)</td></tr><tr><td>Worst</td><td>O(n)</td><td>O(n)</td><td>O(n)</td><td>O(n)</td></tr></table> <p>Memory complexity = O(n)</p>		Access	Search	Insertion	Deletion	Best	O(1)	O(1)	O(1)	O(1)	Average	O(1)	O(1)	O(1)	O(1)	Worst	O(n)	O(n)	O(n)	O(n)
	Access	Search	Insertion	Deletion																	
Best	O(1)	O(1)	O(1)	O(1)																	
Average	O(1)	O(1)	O(1)	O(1)																	
Worst	O(n)	O(n)	O(n)	O(n)																	
Advantages																					
Limitations																					

Examples	
----------	--

## Binary Tree

Aspects	Contents																				
Definition	= a hierarchical data structure that stores objects in parent(root)/children relationship.																				
Characteristics	<p>Components</p> <ul style="list-style-type: none"><li>● Root = the topmost only node of the tree that has no parent node.</li><li>● Edge = acts as a link between the parent node and the child node.</li><li>● Leaf = the last node that has no child, which can be multiple.</li><li>● Subtree = the tree considering that particular node as the root node.</li><li>● Depth = the distance from the root node to that particular node.</li><li>● Height = the distance from that node to the deepest node of that subtree.<ul style="list-style-type: none"><li>○ Height of tree = the maximum height of any node, same as the height of root node.</li></ul></li></ul> <p>Properties</p> <ul style="list-style-type: none"><li>● Maximum number of nodes at level 'l' is <math>2^l</math>.</li><li>● Maximum number of nodes of height 'h' is <math>2^h - 1</math>.</li><li>● Minimum possible height or the minimum number of levels is <math>\log^2(n+1)</math>, with n nodes.</li></ul>																				
Complexity	<p>Time complexity</p> <table><tr><td></td><td>Access</td><td>Search</td><td>Insertion</td><td>Deletion</td></tr><tr><td>Best</td><td><math>O(\log n)</math></td><td><math>O(\log n)</math></td><td><math>O(\log n)</math></td><td><math>O(\log n)</math></td></tr><tr><td>Average</td><td><math>O(\log n)</math></td><td><math>O(\log n)</math></td><td><math>O(\log n)</math></td><td><math>O(\log n)</math></td></tr><tr><td>Worst</td><td><math>O(n)</math></td><td><math>O(n)</math></td><td><math>O(n)</math></td><td><math>O(n)</math></td></tr></table> <p>Memory complexity = <math>O(n)</math></p>		Access	Search	Insertion	Deletion	Best	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	Average	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	Worst	$O(n)$	$O(n)$	$O(n)$	$O(n)$
	Access	Search	Insertion	Deletion																	
Best	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$																	
Average	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$																	
Worst	$O(n)$	$O(n)$	$O(n)$	$O(n)$																	
Advantages	<ol style="list-style-type: none"><li>1. Allows dynamic size and can allocate memory in runtime.</li><li>2. provides moderate access/search (faster than Linked List and slower than Arrays).</li><li>3. provides moderate insertion/deletion (quicker than Arrays and slower than Unordered Linked Lists).</li></ol>																				
Limitations	N/A																				
Examples	<ul style="list-style-type: none"><li>● Manipulate hierarchical data.</li><li>● Make information easy to search (see tree traversal).</li><li>● Manipulate sorted lists of data.</li><li>● As a workflow for compositing digital images for visual effects.</li><li>● Router algorithms.</li></ul>																				

- |  |   |
|--|---|
|  | <ul style="list-style-type: none"><li>• Form of multi-stage decision-making (see business chess).</li></ul> |
|--|---|