**Mentoring 12202018**

**Data Structures**

A data structure is a particular way of storing and organizing data in a computer so that it can be used efficiently. Data structures provide a means to manage large amounts of data efficiently. efficient data structures are a key to designing efficient algorithms.

**Array:**

What is Arrays in Java?

An *array* is a container object that holds a fixed number of values of a single type. The length of an array is established when the array is created. After creation, its length is fixed.

An **array in Java is an object**. Now the question how is this possible? What is the reason behind that? In Java, we can create arrays by using new operator and we know that every object is created using new operator. Hence we can say that array is also an object. Now the question also arises, every time we create an object for a class then what is the class of array?

* In Java, there is a class for every array type, so there’s a class for int[] and similarly for float, double etc.
* The direct superclass of an array type is Object. Every array type implements the interfaces Cloneable and java.io.Serializable.

For every array type corresponding classes are available and these classes are the part of java language and not available to the programmer level. To know the class of any array:

**IntelliJ : knowArray> DemoArray1**

**Array type** **Corresponding class Name**

int[] [I

int[][] [[I

double[] [D

double[][] [[D

short[] [S

byte[] [B

boolean[] [Z

String [] [Ljava.lang.String;

**Declare and Initializing Array:**

An array of 1000 integers has been created. What is the largest integer that can be used as an index to the array? Ans: 999.

**IntelliJ : knowArray> DemoArray2**

**Arrays Length is fixed:**

**IntelliJ : knowArray> DemoArray3**

[Given](https://codelab4.turingscraft.com/lab) that an array [named](https://codelab4.turingscraft.com/lab) ‘a’ whose [elements](https://codelab4.turingscraft.com/lab) are of [type](https://codelab4.turingscraft.com/lab) [int](https://codelab4.turingscraft.com/lab) has been [declared](https://codelab4.turingscraft.com/lab), [assign](https://codelab4.turingscraft.com/lab) the [value](https://codelab4.turingscraft.com/lab) 20 to the **last** [element](https://codelab4.turingscraft.com/lab) in a.

Ans: a[a.length-1] = 20;

[Assume](https://codelab4.turingscraft.com/lab) that an [array](https://codelab4.turingscraft.com/lab) of [integers](https://codelab4.turingscraft.com/lab) [named](https://codelab4.turingscraft.com/lab) a has been [declared](https://codelab4.turingscraft.com/lab) and [initialized](https://codelab4.turingscraft.com/lab). Write a single [statement](https://codelab4.turingscraft.com/lab) that [assigns](https://codelab4.turingscraft.com/lab) a new [value](https://codelab4.turingscraft.com/lab) to the first [element](https://codelab4.turingscraft.com/lab) of the [array](https://codelab4.turingscraft.com/lab). The new [value](https://codelab4.turingscraft.com/lab) should be [equal](https://codelab4.turingscraft.com/lab) to twice the [value](https://codelab4.turingscraft.com/lab) [stored](https://codelab4.turingscraft.com/lab) in the last [element](https://codelab4.turingscraft.com/lab) of the [array](https://codelab4.turingscraft.com/lab).

Ans: a[0] = a[a.length-1] \* 2;

Some Useful use of arrays:

**IntelliJ : knowArray> DemoArray4**

Accessing the elements of a specific Array:

**IntelliJ : knowArray> DemoArray5**

Generating Array elements by Random Class:

**IntelliJ : knowArray> DemoArray6**

Basic of Random:

**IntelliJ : knowRandom> DemoRandom1**

**For more about Arrays:**

<https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html>

<https://www.geeksforgeeks.org/array-primitive-type-object-java/>

<https://docs.oracle.com/javase/8/docs/api/java/util/Objects.html>

**Iterator:**

‘Iterator’ is an interface from **java.util** package which is used to iterate or traverse or retrieve a Collection elements one by one. Iterator must be used whenever we want to enumerate elements in all Collection framework implemented interfaces like Set, List, Queue, Deque and also in all implemented classes of Map interface. Iterator is the **only** cursor available for entire collection framework.

Iterator object can be created by calling *iterator()* method present in Collection interface.

// Here "c" is any Collection object. itr is of

// type Iterator interface and refers to "c"

Iterator itr = c.iterator();

An Iterator traverses the elements in one direction only. It just goes.

**IntelliJ : knowIterator> DemoIterator1**

If we observe the two above examples, both examples are doing the same thing. In Example-1, we have created Iterator object externally and retrieved List object elements one by one. In Example-2, we have Not created Iterator object externally. We are using Enhanced for loop to retrieve the List object elements one by one.

Enhanced for loop uses Iterator object internally and do the same thing like External Iterator example. So both examples gives the same output

**ListIterator:**

‘ListIterator’ in Java is an Iterator which allows users to traverse Collection in both direction.

**IntelliJ : knowIterator> DemoIterator2**

Iterator and ListIterator are two interfaces in Java to traverse data structures. The differences between these two are:

I. ListIterator can be used to traverse only a List. But Iterator can be used to traverse List, Set, and Queue etc.

II. An Iterator traverses the elements in one direction only. It just goes. ListIterator can traverse the elements in two directions i.e. backward as well as forward directions.

III. Iterator cannot provide us index of an element in the Data Structure. ListIterator provides us methods like nextIndex() and previousIndex() to get the index of an element during traversal.

IV. Iterator does not allow us to add an element to collection while traversing it. It throws ConcurrentModificationException. ListIterator allows use to add an element at any point of time while traversing a list.

An existing element’s value cannot be replaced by using Iterator. ListIterator provides the method set(e) to replace the value of last element returned by next() or previous() methods.

**Know ForLoop:**

**Simple for loop-**

**for**(**int** i=0;i<5;i++){  
 System.***out***.println(i);  
}

**Two independent Simple for Loop-**

**for**(**int** i=0;i<5;i++){  
 System.***out***.println(i);  
}  
  
**for**(**int** j=5;j<10;j++){  
 System.***out***.println(j);  
}

**Differentiate Print and Println:**

**Loop inside Loop:**

**for**(**int** i=0;i<5;i++){  
 System.***out***.print(i);  
  
  
 **for**(**int** j=5;j<10;j++){  
 System.***out***.print(j);  
  
 }  
 System.***out***.println();  
}

**Changing numbers by asterisk in nested loop:**

**for**(**int** i=0;i<5;i++){  
 System.***out***.print(i);  
  
  
 **for**(**int** j=5;j<10;j++){  
 System.***out***.print(**"\*"**);  
  
 }  
 System.***out***.println();  
}

**Printing Triangle with the nested for loop:**

**for**(**int** i=1;i<8;i++){  
 *//System.out.print(i);* **for**(**int** j=0;j<i;j++){  
 System.***out***.print(**"\*"**);  
  
 }  
 System.***out***.println();  
}

**IntelliJ : knowForLoop> DemoForLoop1**

**IntelliJ : knowForLoop> DemoForLoop2**

**ArrayList:**

public class **ArrayList<E>**

extends [AbstractList](https://docs.oracle.com/javase/8/docs/api/java/util/AbstractList.html)<E>

implements [List](https://docs.oracle.com/javase/8/docs/api/java/util/List.html)<E>, [RandomAccess](https://docs.oracle.com/javase/8/docs/api/java/util/RandomAccess.html), [Cloneable](https://docs.oracle.com/javase/8/docs/api/java/lang/Cloneable.html), [Serializable](https://docs.oracle.com/javase/8/docs/api/java/io/Serializable.html)

*What is <E>?*

*These are called*[*Generics*](http://docs.oracle.com/javase/tutorial/java/generics/index.html)*.*

*In general, these enable types (classes and interfaces) to be parameters when defining classes, interfaces and methods.*

*Using generics give many benefits over using non-generic code*

*Generics in Java is to allow type (Integer, String, … etc and user defined types) to be a parameter to methods, classes and interfaces. For example, classes like HashSet, ArrayList, HashMap, etc use generics very well. We can use them for any type.*

// To create an instance of generic class

BaseType <Type> obj = new BaseType <Type>();

ArrayList is class and is present in java.util package. Arrays, ArrayList are linear data Structures. It provides dynamic arrays in Java. Though, it may be slower than standard arrays but can be helpful in programs where lots of manipulation in the array is needed.

* ArrayList is initialized by a size, however the size can increase if collection grows or shrunk if objects are removed from the collection.
* Java ArrayList allows us to randomly access the list.
* ArrayList can not be used for primitive types, like int, char, etc. We need a wrapper class for such cases (see [this](https://www.geeksforgeeks.org/array-vs-arraylist-in-java/) for details).

Creating ArrayList and adding elements to the ArrayList:

**IntelliJ : knowArrayList> DemoArrayList1**

Removing Elements from ArrayList:

**IntelliJ : knowArrayList> DemoArrayList2**

Iterating ArrayList in three different methods:

**IntelliJ : knowArrayList> DemoArrayList3**

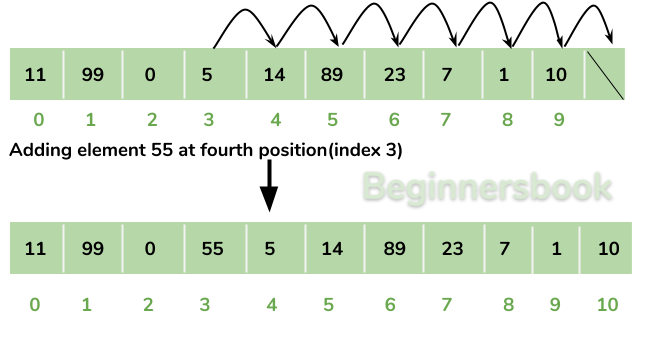
How to Find Length of ArrayList:

**IntelliJ : knowArrayList> DemoArrayList4**

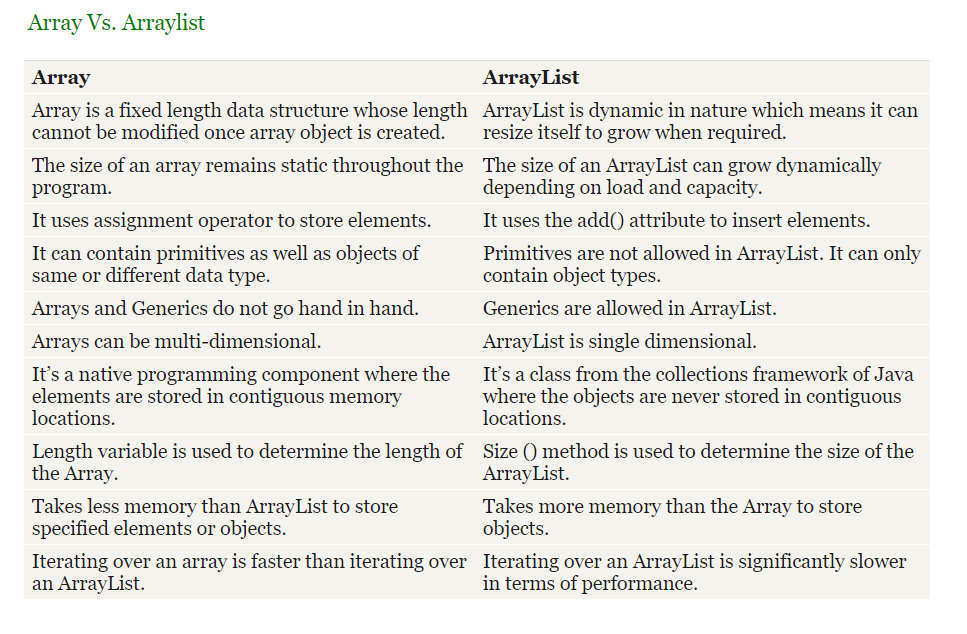
Why ArrayList is better than Array?

The limitation with array is that it has a fixed length so if it is full you cannot add any more elements to it, likewise if there are number of elements gets removed from it the memory consumption would be the same as it doesn’t shrink.

On the other ArrayList can dynamically grow and shrink after addition and removal of elements (See the images below). Apart from these benefits ArrayList class enables us to use predefined methods of it which makes our task easy. Let’s see the diagrams to understand the addition and removal of elements from ArrayList and then we will see the programs.

**Adding Element in ArrayList at specified position:**  


**Removing Element from ArrayList:**  

Know more about ArrayList: <https://beginnersbook.com/2013/12/java-arraylist/>

**LinkedList:**

public class **LinkedList<E>**

extends [AbstractSequentialList](https://docs.oracle.com/javase/7/docs/api/java/util/AbstractSequentialList.html)<E>

implements [List](https://docs.oracle.com/javase/7/docs/api/java/util/List.html)<E>, [Deque](https://docs.oracle.com/javase/7/docs/api/java/util/Deque.html)<E>, [Cloneable](https://docs.oracle.com/javase/7/docs/api/java/lang/Cloneable.html), [Serializable](https://docs.oracle.com/javase/7/docs/api/java/io/Serializable.html)

LinkedList<E> is a java class from the package java.util. Like arrays, Linked List is a linear data structure. Unlike arrays, linked list elements are not stored at contiguous location; the elements are linked using pointers.

Linked List are linear data structures where the elements are not stored in contiguous locations and every element is a separate object with a data part and address part. The elements are linked using pointers and addresses. Each element is known as a node.

### Types of Linked Lists:

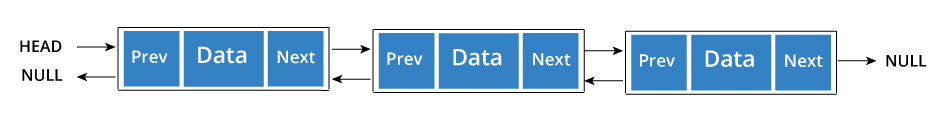
A singly linked list-

Each node has data and a pointer to the next node.



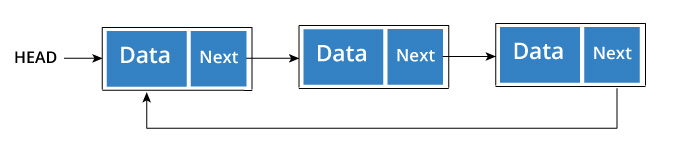
A doubly linked list-

We add a pointer to the previous node in a doubly linked list. Thus, we can go in either direction: forward or backward.



Circular Linked List

A circular linked list is a variation of linked list in which the last element is linked to the first element. This forms a circular loop.



A circular linked list can be either singly linked or doubly linked.

* for singly linked list, next pointer of last item points to the first item
* In doubly linked list, prev pointer of first item points to last item as well.

Creating, Adding, Removing elements from LinkedList:

**IntelliJ : knowLinkedList> DemoLinkedList1**

Iteration in LinkedList:

**IntelliJ : knowLinkedList> DemoLinkedList2**

# Difference between ArrayList and LinkedList in Java

[ArrayList](https://beginnersbook.com/2013/12/java-arraylist/) and [LinkedList](https://beginnersbook.com/2013/12/linkedlist-in-java-with-example/" \o "LinkedList" \t "_blank) both implements List interface and their methods and results are almost identical. However there are few differences between them which make one better over another depending on the requirement.

## ArrayList Vs LinkedList

## ArrayList takes consecutive memory addresses while LinkedList does not.

1) **Search**: ArrayList search operation is pretty fast compared to the LinkedList search operation. get(int index) in ArrayList gives the performance of O(1) while LinkedList performance is O(n).

***[O(1) means the time to access something is independent /constant of the number of items in the collection.***

***O(N) would mean the time to access an item is a proportional to the number (N) of items in the collection.]***

Reason: ArrayList maintains index based system for its elements as it uses array data structure implicitly which makes it faster for searching an element in the list. On the other side LinkedList implements **doubly linked list** which requires the traversal through all the elements for searching an element.

2) **Deletion**: LinkedList remove operation gives O(1) performance while ArrayList gives variable performance: O(n) in worst case (while removing first element) and O(1) in best case (While removing last element).

Conclusion: LinkedList element deletion is faster compared to ArrayList.

Reason: LinkedList’s each element maintains two pointers (addresses) which points to the both neighbor elements in the list. Hence removal only requires change in the pointer location in the two neighbor nodes (elements) of the node which is going to be removed. While In ArrayList all the elements need to be shifted to fill out the space created by removed element.

3) **Inserts Performance**: LinkedList add method gives O(1) performance while ArrayList gives O(n) in worst case. Reason is same as explained for remove.

4) **Memory Overhead**: ArrayList maintains indexes and element data while LinkedList maintains element data and two pointers for neighbor nodes hence the memory consumption is high in LinkedList comparatively.

There are few **similarities between** these classes which are as follows:

1. Both ArrayList and LinkedList are implementation of List interface.
2. They both maintain the elements insertion order which means while displaying ArrayList and LinkedList elements the result set would be having the same order in which the elements got inserted into the List.
3. Both these classes are non-synchronized and can be made synchronized explicitly by using [Collections.synchronizedList](https://docs.oracle.com/javase/6/docs/api/java/util/Collections.html" \l "synchronizedList(java.util.List)" \t "_blank) method.
4. The iterator and listIterator returned by these classes are fail-fast (if list is structurally modified at any time after the iterator is created, in any way except through the iterator’s own remove or add methods, the iterator will throw a [ConcurrentModificationException](https://docs.oracle.com/javase/6/docs/api/java/util/ConcurrentModificationException.html" \t "_blank)).

### When to use LinkedList and when to use ArrayList?

1) As explained above the insert and remove operations give good performance (O(1)) in LinkedList compared to ArrayList(O(n)). Hence if there is a requirement of frequent addition and deletion in application then LinkedList is a best choice.

2) Search (get method) operations are fast in Arraylist (O(1)) but not in LinkedList (O(n)) so If there are less add and remove operations and more search operations requirement, ArrayList would be your best bet.

For more: <https://beginnersbook.com/2013/12/linkedlist-in-java-with-example/>

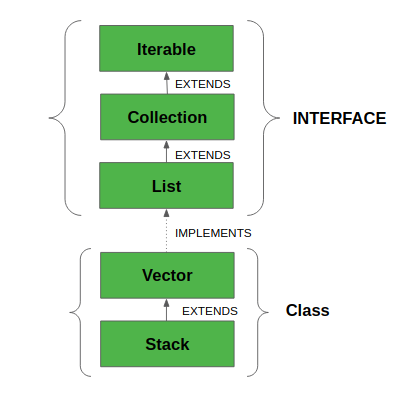
Stack:

public class **Stack<E>**

extends [Vector](https://docs.oracle.com/javase/7/docs/api/java/util/Vector.html)<E>

Stack is a class from Java.util. A Stack is a Last In First Out (LIFO) or FILO data structure. The class is based on the basic principle of last-in-first-out. In addition to the basic push and pop operations, the class provides three more functions of empty, search and peek.

The below diagram shows the hierarchy of Stack class:



**Methods in Stack class**

1. [**Object push(Object element)**](https://www.geeksforgeeks.org/stack-push-method-in-java/)**: Pushes an element on the top of the stack.**
2. [**Object pop()**](https://www.geeksforgeeks.org/stack-pop-method-in-java/)**: Removes and returns the top element of the stack. An ‘EmptyStackException’ exception is thrown if we call pop() when the invoking stack is empty.**
3. [**Object peek()**](https://www.geeksforgeeks.org/stack-peek-method-in-java/)**: Returns the element on the top of the stack, but does not remove it.**
4. [**boolean empty()**](https://www.geeksforgeeks.org/stack-empty-method-in-java/)**: It returns true if nothing is on the top of the stack. Else, returns false.**
5. [**int search(Object element)**](https://www.geeksforgeeks.org/stack-search-method-in-java/)**: It determines whether an object exists in the stack. If the element is found, it returns the position of the element from the top of the stack. Else, it returns -1.**

**IntelliJ : knowStack> DemoStack1**

**Viewing values of Stack:**

**IntelliJ : knowStack> DemoStack2**

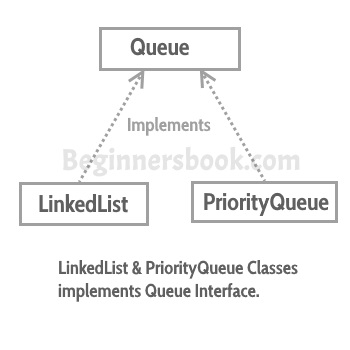
Queue:

public interface **Queue<E>**

extends [Collection](https://docs.oracle.com/javase/7/docs/api/java/util/Collection.html)<E>

A **Queue** is designed in such a way so that the elements added to it are placed at the end of Queue and removed from the beginning of Queue. The concept here is similar to the queue we see in our daily life, for example, when a new iPhone launches we stand in a queue outside the apple store, whoever is added to the queue has to stand at the end of it and persons are served on the basis of FIFO (First In First Out), The one who gets the iPhone is removed from the beginning of the queue.

A Queue in Java is just an interface. We need a concrete implementation of the Queue interface to work with, in programs.



Queue interface in Java collections has two implementation: LinkedList and PriorityQueue, these two classes implements Queue interface.  
**Queue is an interface** so we cannot instantiate it, rather we create instance of LinkedList or PriorityQueue and assign it to the Queue like this.

**IntelliJ : knowQueue> DemoQueue1**

## Some Important Methods of Queue interface

**E element()**: This method returns the head (the first element) of the Queue.

**E remove()**: This method removes the head(first element) of the Queue and returns its value.

**E poll()**: This method is almost same as remove() method. The only difference between poll() and remove() is that poll() method returns null if the Queue is empty.

**E peek()**: This method is almost same as element() method. The only difference between peek() and element() is that peek() method returns null if the Queue is empty.

**IntelliJ : knowQueue> DemoQueue2**

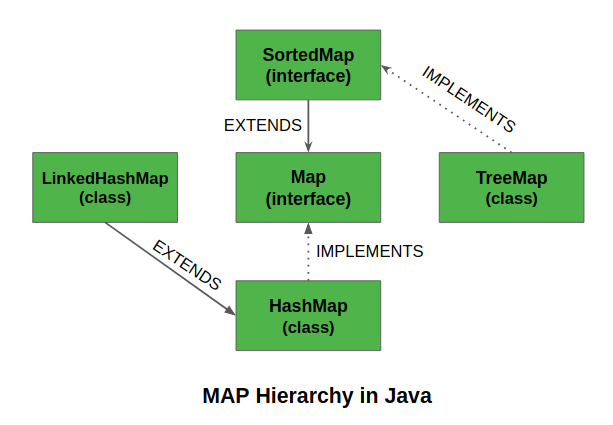
Map:

The java.util.Map interface represents a mapping between a key and a value. Therefore it behaves a bit different from the rest of the collection types.

Few characteristics of the Map Interface are:

1. A Map cannot contain duplicate keys and each key can map to at most one value. Some implementations allow null key and null value like the [HashMap](https://www.geeksforgeeks.org/java-util-hashmap-in-java/" \t "_blank)and [LinkedHashMap](https://www.geeksforgeeks.org/linkedhashmap-class-java-examples" \t "_blank), but some do not like the [TreeMap](https://www.geeksforgeeks.org/treemap-in-java/" \t "_blank).
2. The order of a map depends on specific implementations, e.g [TreeMap](https://www.geeksforgeeks.org/treemap-in-java/" \t "_blank)and [LinkedHashMap](https://www.geeksforgeeks.org/linkedhashmap-class-java-examples" \t "_blank)have predictable order, while [HashMap](https://www.geeksforgeeks.org/java-util-hashmap-in-java/" \t "_blank)does not.
3. There are two interfaces for implementing Map in java: Map and [SortedMap](https://www.geeksforgeeks.org/sortedmap-java-examples/" \t "_blank), and three classes: [HashMap](https://www.geeksforgeeks.org/java-util-hashmap-in-java/" \t "_blank), [TreeMap](https://www.geeksforgeeks.org/treemap-in-java/" \t "_blank) and [LinkedHashMap](https://www.geeksforgeeks.org/linkedhashmap-class-java-examples/" \t "_blank).

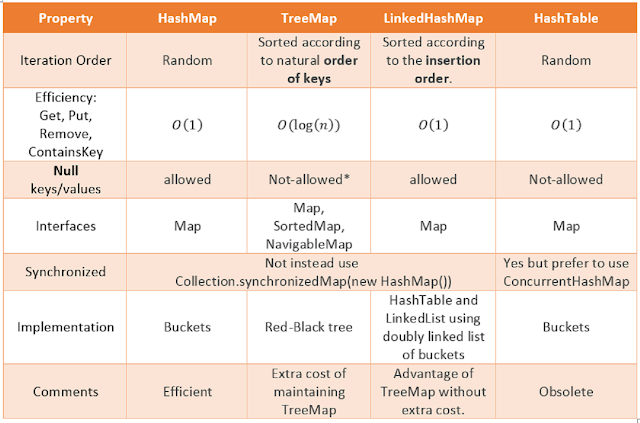
HashMap and LinkedHashMap allow null keys and values, but TreeMap doesn't allow any null key or value.



**IntelliJ : knowHashMap> DemoHashMap1**

**IntelliJ : knowHashMap> DemoHashMap2**

|  |  |
| --- | --- |
| **Class** | **Description** |
| [HashMap](https://www.javatpoint.com/java-hashmap) | HashMap is the implementation of Map, but it doesn't maintain any order. |
| [LinkedHashMap](https://www.javatpoint.com/java-linkedhashmap) | LinkedHashMap is the implementation of Map. It inherits HashMap class. It maintains insertion order. |



Difference between HashMap and HashTable:

HashMap and Hashtable store key/value pairs in a hash table. When using a Hashtable or HashMap, we specify an object that is used as a key, and the value that you want linked to that key. The key is then hashed, and the resulting hash code is used as the index at which the value is stored within the table.

There are several differences between [HashMap](http://java.sun.com/javase/7/docs/api/java/util/HashMap.html) and [Hashtable](http://java.sun.com/javase/7/docs/api/java/util/Hashtable.html) in Java:

1. Hashtable is [synchronized](https://stackoverflow.com/questions/1085709/what-does-synchronized-mean), whereas HashMap is not. This makes HashMap better for non-threaded applications, as unsynchronized Objects typically perform better than synchronized ones.
2. Hashtable does not allow null keys or values. HashMap allows one null key and any number of null values.
3. One of HashMap's subclasses is [LinkedHashMap](http://java.sun.com/javase/7/docs/api/java/util/LinkedHashMap.html), so in the event that you'd want predictable iteration order (which is insertion order by default), you could easily swap out the HashMap for a LinkedHashMap. This wouldn't be as easy if you were using Hashtable.

Since synchronization is not an issue for you, I'd recommend HashMap. If synchronization becomes an issue, you may also look at [ConcurrentHashMap](http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ConcurrentHashMap.html).

**IntelliJ : knowHashMap> DemoHashMap3**

Why HashTable doesn’t allow null and HashMap does?

To successfully store and retrieve objects from a HashTable, the objects used as keys must implement the hashCode method and the equals method. Since null is not an object, it can’t implement these methods. HashMap is an advanced version and improvement on the Hashtable. HashMap was created later.

**Discussion about List, Map, HashMap:**

Lists and Maps are different data structures. Maps are used for when you want to associate a key with a value and Lists are an ordered collection.

Map is an interface in the Java Collection Framework and a HashMap is one implementation of the Map interface. HashMap are efficient for locating a value based on a key and inserting and deleting values based on a key. The entries of a HashMap are not ordered.

ArrayList and LinkedList are an implementation of the List interface. LinkedList provides sequential access and is generally more efficient at inserting and deleting elements in the list, however, it is it less efficient at accessing elements in a list. ArrayList provides random access and is more efficient at accessing elements but is generally slower at inserting and deleting elements.

**Advantage of using map, HashMap/HashTable over ArrayList and LinkedList:**

Searching is easier. If we put all the phone numbers in US or all the cities of the world in ArrayList or LinkedList, the searching will be difficult. But if we can introduce Map/HashMap, we can put key and value, key is the country name and Value is the city name. Thus the searching become faster. This is one of the biggest advantage of Map over ArrayList and Linked List. Map gives the option of Constant search. The sublist could be ArrayList or Linked List. For example: city name, the later 7 digits of phone number. Those are values. Country name and the area codes are key.

Key Differences Between Linear and Non-linear Data Structure:

1. In the linear data structure, the data is organized in a linear order in which elements are linked one after the other. As against, in the non-linear data structure the data elements are not stored in a sequential manner rather the elements are hierarchically related.
2. The traversing of data in the linear data structure is easy as it can make all the data elements to be traversed in one go, but at a time only one element is directly reachable. On the contrary, in the non-linear data structure, the nodes are not visited sequentially and cannot be traversed in one go.
3. Data elements are adjacently attached in the linear data structure, which means only two elements can be linked to two other elements while this is not the case in the non-linear data structure where one data element can be connected to numerous other elements.
4. The linear data structures are easily implemented relative to the non-linear data structure.
5. A single level of elements is incorporated in the linear data structure. Conversely, non-linear data structure involves multiple levels.
6. Examples of the linear data structure are array, queue, stack, linked list, etc. In contrast, tree and graph are the examples of the non-linear data structure.
7. The memory is utilized efficiently in the non-linear data structure where linear data structure tends to waste the memory.

# HashSet in Java

The HashSet class implements the Set interface, backed by a hash table which is actually a HashMap instance. No guarantee is made as to the iteration order of the set which means that the class does not guarantee the constant order of elements over time. This class permits the null element. The class also offers constant time performance for the basic operations like add, remove, contains and size assuming the hash function disperses the elements properly among the buckets, which we shall see further in the article.

Few important features of HashSet are:

* Implements [Set Interface](https://www.geeksforgeeks.org/set-in-java/).
* Underlying data structure for HashSet is hashtable.
* As it implements the Set Interface, duplicate values are not allowed.
* Objects that you insert in HashSet are not guaranteed to be inserted in same order. Objects are inserted based on their hash code.
* NULL elements are allowed in HashSet.
* HashSet also implements Searlizable and Cloneable interfaces.

**IntelliJ : knowHashMap> DemoHashSet4**

# What is the difference between HashSet, HashMap and hash table? How do they behave in a multi-threaded environment?

<https://swethakogatamblogs.wordpress.com/2016/04/02/hashmap-vs-hashtable-vs-hashset/>

<https://www.quora.com/What-is-the-difference-between-HashSet-HashMap-and-hash-table-How-do-they-behave-in-a-multi-threaded-environment>

**Tree:**

Unlike Arrays, Linked Lists, Stack and queues, which are linear data structures, trees are hierarchical data structures (non-linear).

**Tree Vocabulary:**The topmost node is called root of the tree. The elements that are directly under an element are called its children. The element directly above something is called its parent. For example, ‘a’ is a child of ‘f’, and ‘f’ is the parent of ‘a’. Finally, elements with no children are called leaves.

tree

----

j <-- root

/ \

f k

/ \ \

a h z <-- leaves

**Why Trees?**

**1.** One reason to use trees might be because you want to store information that naturally forms a hierarchy. For example, the file system on a computer:

file system

-----------

/ <-- root

/ \

... home

/ \

ugrad course

/ / | \

... cs101 cs112 cs113

**2.** Trees (with some ordering e.g., BST) provide moderate access/search (quicker than Linked List and slower than arrays).

**3.** Trees provide moderate insertion/deletion (quicker than Arrays and slower than Unordered Linked Lists).

**4.** Like Linked Lists and unlike Arrays, Trees don’t have an upper limit on number of nodes as nodes are linked using pointers.

**Main applications of trees include:**

**1.** Manipulate hierarchical data.

**2.** Make information easy to search (see tree traversal).

**3.** Manipulate sorted lists of data.

**4.** As a workflow for compositing digital images for visual effects.

**5.**Router algorithms

**6.**Form of a multi-stage decision-making (see business chess).

TreeSet:

TreeSet is a class from Java.util package.

public class ****TreeSet<E>****

extends [AbstractSet](https://docs.oracle.com/javase/7/docs/api/java/util/AbstractSet.html)<E>

implements [NavigableSet](https://docs.oracle.com/javase/7/docs/api/java/util/NavigableSet.html)<E>, [Cloneable](https://docs.oracle.com/javase/7/docs/api/java/lang/Cloneable.html), [Serializable](https://docs.oracle.com/javase/7/docs/api/java/io/Serializable.html)

Two things must be kept in mind while creating and adding elements into a TreeSet:

* Firstly, insertion of null into a TreeSet throws *[NullPointerException](https://www.geeksforgeeks.org/null-pointer-exception-in-java/)* because while insertion of null, it gets compared to the existing elements and null cannot be compared to any value.
* Secondly, if we are depending on default natural sorting order, compulsory the object should be **homogeneous**and **comparable** otherwise we will get **RuntimeException:***ClassCastException*

**IntelliJ : knowTree> DemoTree1**

TreeMap:

TreeMap is a class from the package java.util.

public class ****TreeMap<K,V>****

extends [AbstractMap](https://docs.oracle.com/javase/7/docs/api/java/util/AbstractMap.html)<K,V>

implements [NavigableMap](https://docs.oracle.com/javase/7/docs/api/java/util/NavigableMap.html)<K,V>, [Cloneable](https://docs.oracle.com/javase/7/docs/api/java/lang/Cloneable.html), [Serializable](https://docs.oracle.com/javase/7/docs/api/java/io/Serializable.html)

* This class is a member of Java Collections Framework.
* The class implements Map interfaces including NavigableMap, SortedMap and extends AbstractMap
* TreeMap in Java does not allow null keys (like Map) and thus a NullPointerException is thrown. However, multiple null values can be associated with different keys.
* All Map.Entry pairs returned by methods in this class and its views represent snapshots of mappings at the time they were produced. They do not support the Entry.setValue method.

**IntelliJ : knowTree> DemoTree2**

As we can see that we have inserted the data in random order however when we displayed the TreeMap content we got the sorted result in the ascending order of keys.

**Graph:**

A Graph is a non-linear data structure consisting of nodes and edges. The nodes are sometimes also referred to as vertices and the edges are lines or arcs that connect any two nodes in the graph. More formally a Graph can be defined as,

*A Graph consists of a finite set of vertices(or nodes) and set of Edges which connect a pair of nodes.*



In the above Graph, the set of vertices V = {0,1,2,3,4} and the set of edges E = {01, 12, 23, 34, 04, 14, 13}.

Graphs are used to solve many real-life problems. Graphs are used to represent networks. The networks may include paths in a city or telephone network or circuit network. Graphs are also used in social networks like linkedIn, Facebook. For example, in Facebook, each person is represented with a vertex(or node). Each node is a structure and contains information like person id, name, gender, locale etc.

Use of Graphs:

* In **Computer science** graphs are used to represent the flow of computation.
* **Google maps** uses graphs for building transportation systems, where intersection of two(or more) roads are considered to be a vertex and the road connecting two vertices is considered to be an edge, thus their navigation system is based on the algorithm to calculate the shortest path between two vertices.
* In **Facebook**, users are considered to be the vertices and if they are friends then there is an edge running between them. Facebook’s Friend suggestion algorithm uses graph theory.
* In **Operating System**, we come across the Resource Allocation Graph where each process and resources are considered to be vertices. Edges are drawn from resources to the allocated process, or from requesting process to the requested resource. If this leads to any formation of a cycle then a deadlock will occur.

For more about basic of graph:

<https://adrianmejia.com/blog/2018/05/14/data-structures-for-beginners-graphs-time-complexity-tutorial/>

Sorting:

Sorting an Array:

**IntelliJ : knowSort> DemoSort1**

Sorting ArrayList in Ascending and Descending order:

**IntelliJ : knowSort> DemoSort2**

Sorting Array in Ascending order by using for loop:

**IntelliJ : knowSort> DemoSort3**

**What is Collection Framework in Java?**

The Java platform includes a *collections framework*. A *collection* is an object that represents a group of objects (such as the classic [Vector](https://docs.oracle.com/javase/8/docs/api/java/util/Vector.html) class). A collections framework is a unified architecture for representing and manipulating collections, enabling collections to be manipulated independently of implementation details. The Collection interface (**java.util.Collection**) and Map interface (**java.util.Map**) are the two main “root” interfaces of Java collection classes.

The primary advantages of a collections framework are that it:

* **Reduces programming effort** by providing data structures and algorithms so you don't have to write them yourself.
* **Increases performance** by providing high-performance implementations of data structures and algorithms. Because the various implementations of each interface are interchangeable, programs can be tuned by switching implementations.
* **Provides interoperability between unrelated APIs** by establishing a common language to pass collections back and forth.
* **Reduces the effort required to learn APIs** by requiring you to learn multiple ad hoc collection APIs.
* **Reduces the effort required to design and implement APIs** by not requiring you to produce ad hoc collections APIs.
* **Fosters software reuse** by providing a standard interface for collections and algorithms with which to manipulate them.

<https://www.geeksforgeeks.org/collections-in-java-2/>

<https://docs.oracle.com/javase/8/docs/technotes/guides/collections/overview.html>

What is Thread in Java:

Thread is a class from Java.lang package. A thread, in the context of Java, is the path followed when executing a program. All Java programs have at least one thread, known as the main thread, which is created by the Java Virtual Machine (JVM) at the program’s start, when the main() method is invoked with the main thread.

In Java, creating a thread is accomplished by implementing an interface and extending a class. Every Java thread is created and controlled by the java.lang.Thread class.

Java is a multi-threaded application that allows multiple thread execution at any particular time.

What is synchronization in Java:

Synchronization in Java is an important concept since Java is a multi-threaded language where multiple threads run in parallel to complete program execution. In multi-threaded environment *synchronization of Java object or synchronization of Java class becomes extremely important*. Synchronization in Java is possible by usingJava keyword ***"synchronized".***

Synchronization can introduce thread contention, which occurs when two or more threads try to access the same resource simultaneously and cause the Java runtime to execute one or more threads more slowly, or even suspend their execution.

Multi-threaded programs may often come to a situation where multiple threads try to access the same resources and finally produce erroneous and unforeseen results.

So it needs to be made sure by some synchronization method that only one thread can access the resource at a given point of time.

Java provides a way of creating threads and synchronizing their task by using synchronized blocks. Synchronized blocks in Java are marked with the synchronized keyword. A synchronized block in Java is synchronized on some object. All synchronized blocks synchronized on the same object can only have one thread executing inside them at a time. All other threads attempting to enter the synchronized block are blocked until the thread inside the synchronized block exits the block.

What is generics in Java?

Generic types are instantiated to form parameterized types by providing actual type arguments that replace the formal type parameters. A class like LinkedList<E> is a generic type, that has a type parameter E . Instantiations, such as LinkedList<Integer> or a LinkedList<String>, are called parameterized types, and String and Integer are the respective actual type arguments.

For example, suppose you have a method that adds two numbers together. In order to work with the types themselves, you might have to create multiple versions of this method. For instance:

public int Add(int a, int b)

public double Add(double a, double b)

public float Add(float a, float b)

Generics allow you to create a single method that is customized for the type that invokes it.

public T Add<T>(T a, T b)

T is substituted for whatever type you use.

More about generics: <https://howtodoinjava.com/java/generics/complete-java-generics-tutorial/>

Wrapper Class, AutoBoxing, UnBoxing:

<https://www.baeldung.com/java-wrapper-classes>

**Types of Sorting in Java Data Structure:**

**Sorting** arranges data in a sequence which makes searching easier.

Sorting is nothing but arranging the data in ascending or descending order. The term **sorting** came into picture, as humans realised the importance of searching quickly.

There are so many things in our real life that we need to search for, like a particular record in database, roll numbers in merit list, a particular telephone number in telephone directory, a particular page in a book etc. All this would have been a mess if the data was kept unordered and unsorted, but fortunately the concept of **sorting** came into existence, making it easier for everyone to arrange data in an order, hence making it easier to search.

The two main criterias to judge which algorithm is better than the other have been:

1. Time taken to sort the given data.
2. Memory Space required to do so.

Bubble Sort:

**Bubble Sort** is a simple algorithm which is used to sort a given set of n elements provided in form of an array with n number of elements. Bubble Sort compares all the element one by one and sort them based on their values.

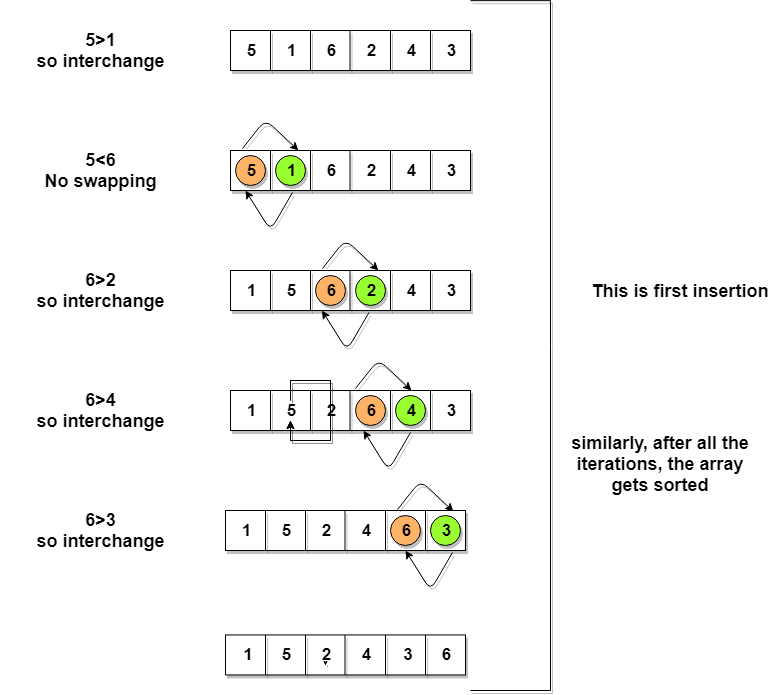
If the given array has to be sorted in ascending order, then bubble sort will start by comparing the first element of the array with the second element, if the first element is greater than the second element, it will **swap** both the elements, and then move on to compare the second and the third element, and so on.

If we have total n elements, then we need to repeat this process for n-1 times.

It is known as **bubble sort**, because with every complete iteration the largest element in the given array, bubbles up towards the last place or the highest index, just like a water bubble rises up to the water surface.

Let's consider an array with values {5, 1, 6, 2, 4, 3}

Below, we have a pictorial representation of how bubble sort will sort the given array.



So as we can see in the representation above, after the first iteration, 6 is placed at the last index, which is the correct position for it.

Similarly after the second iteration, 5 will be at the second last index, and so on.

**Complexity Analysis of Bubble Sort**

In Bubble Sort, n-1 comparisons will be done in the 1st pass, n-2 in 2nd pass, n-3 in 3rd pass and so on. So the total number of comparisons will be,

(n-1) + (n-2) + (n-3) + ..... + 3 + 2 + 1

Sum = n(n-1)/2

i.e O(n2)

Hence the **time complexity** of Bubble Sort is **O(n2)**.

The main advantage of Bubble Sort is the simplicity of the algorithm.

The **space complexity** for Bubble Sort is **O(1)**, because only a single additional memory space is required i.e. for temp variable.

Also, the **best case time complexity** will be **O(n)**, it is when the list is already sorted.

Following are the Time and Space complexity for the Bubble Sort algorithm.

* Worst Case Time Complexity [ Big-O ]: **O(n2)**
* Best Case Time Complexity [Big-omega]: **O(n)**
* Average Time Complexity [Big-theta]: **O(n2)**
* Space Complexity: **O(1)**

**IntelliJ : knowBubbleSort> DemoBubbleSort, TestBubbleSort**

**Selection Sort:**

Selection sort is conceptually the most simplest sorting algorithm. This algorithm will first find the **smallest** element in the array and swap it with the element in the **first** position, then it will find the **second smallest** element and swap it with the element in the **second** position, and it will keep on doing this until the entire array is sorted.

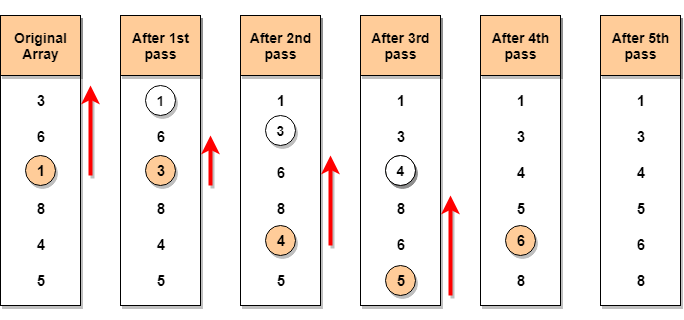
It is called selection sort because it repeatedly **selects** the next-smallest element and swaps it into the right place.

Following are the steps involved in selection sort(for sorting a given array in ascending order):

1. Starting from the first element, we search the smallest element in the array, and replace it with the element in the first position.
2. We then move on to the second position, and look for smallest element present in the subarray, starting from index 1, till the last index.
3. We replace the element at the **second** position in the original array, or we can say at the first position in the subarray, with the second smallest element.
4. This is repeated, until the array is completely sorted.

Let's consider an array with values {3, 6, 1, 8, 4, 5}

Below, we have a pictorial representation of how selection sort will sort the given array.



In the **first** pass, the smallest element will be 1, so it will be placed at the first position.

Then leaving the first element, **next smallest** element will be searched, from the remaining elements. We will get 3 as the smallest, so it will be then placed at the second position.

Then leaving 1 and 3(because they are at the correct position), we will search for the next smallest element from the rest of the elements and put it at third position and keep doing this until array is sorted.

### Complexity Analysis of Selection Sort:

Selection Sort requires two nested for loops to complete itself, one for loop is in the function selectionSort, and inside the first loop we are making a call to another function indexOfMinimum, which has the second(inner) for loop.

Hence for a given input size of n, following will be the time and space complexity for selection sort algorithm:

Worst Case Time Complexity [ Big-O ]: **O(n2)**

Best Case Time Complexity [Big-omega]: **O(n2)**

Average Time Complexity [Big-theta]: **O(n2)**

Space Complexity: **O(1)**

**IntelliJ : knowSelectionSort> DemoSelectionSort**

**Insertion Sorting:**

Insertion sort is a simple sorting algorithm, it builds the final sorted array one item at a time. It is much less efficient on large lists than other sort algorithms.

Advantages of Insertion Sort:

1) It is very simple.

2) It is very efficient for small data sets.

3) It is stable; i.e., it does not change the relative order of elements with equal keys.  
4) In-place; i.e., only requires a constant amount O(1) of additional memory space.



### Complexity Analysis of Insertion Sort

As we mentioned above that insertion sort is an efficient sorting algorithm, as it does not run on preset conditions using for loops, but instead it uses one while loop, which avoids extra steps once the array gets sorted.

Even though insertion sort is efficient, still, if we provide an already sorted array to the insertion sort algorithm, it will still execute the outer for loop, thereby requiring n steps to sort an already sorted array of n elements, which makes its **best case time complexity** a linear function of n.

Worst Case Time Complexity [ Big-O ]: **O(n2)**

Best Case Time Complexity [Big-omega]: **O(n)**

Average Time Complexity [Big-theta]: **O(n2)**

Space Complexity: **O(1)**

**IntelliJ : knowInsertionSort> DemoInsertionSort**

**Merge Sorting:**

In merge sort, we break the given array midway, for example if the original array had 6 elements, then merge sort will break it down into two subarrays with 3 elements each.

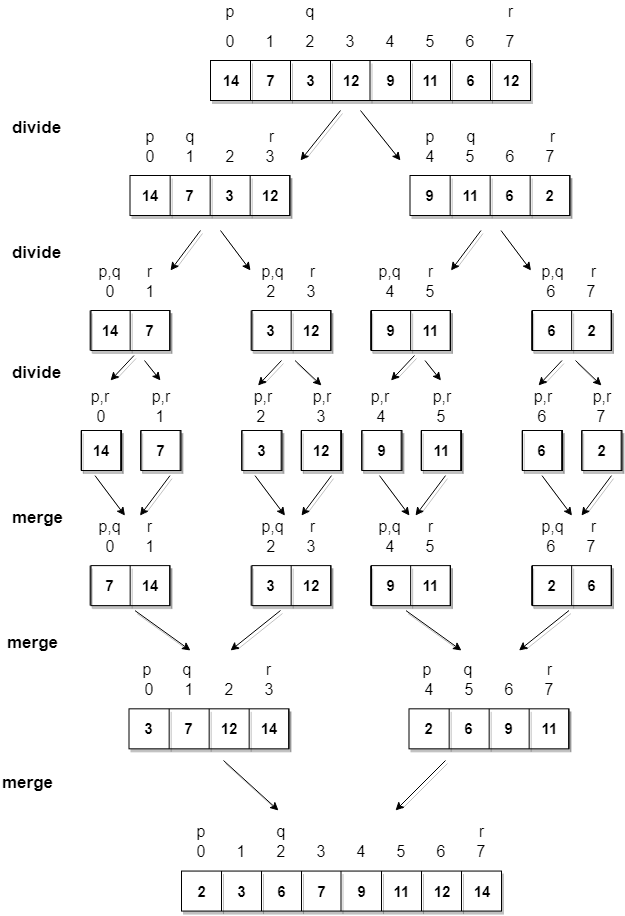
But breaking the orignal array into 2 smaller subarrays is not helping us in sorting the array.

So we will break these subarrays into even smaller subarrays, until we have multiple subarrays with **single element** in them. Now, the idea here is that an array with a single element is already sorted, so once we break the original array into subarrays which has only a single element, we have successfully broken down our problem into base problems.

And then we have to merge all these sorted subarrays, step by step to form one single sorted array.

Let's consider an array with values {14, 7, 3, 12, 9, 11, 6, 12}

Below, we have a pictorial representation of how merge sort will sort the given array.



In merge sort we follow the following steps:

1. We take a variable p and store the starting index of our array in this. And we take another variable r and store the last index of array in it.
2. Then we find the middle of the array using the formula (p + r)/2 and mark the middle index as q, and break the array into two subarrays, from p to q and from q + 1 to r index.
3. Then we divide these 2 subarrays again, just like we divided our main array and this continues.
4. Once we have divided the main array into subarrays with single elements, then we start merging the subarrays.

### Complexity Analysis of Merge Sort:

Merge Sort is quite fast, and has a time complexity of O(n\*log n). It is also a stable sort, which means the "equal" elements are ordered in the same order in the sorted list.

In this section we will understand why the running time for merge sort is O(n\*log n).

As we have already learned in [Binary Search](https://www.studytonight.com/data-structures/binary-search-algorithm) that whenever we divide a number into half in every stpe, it can be represented using a logarithmic function, which is log n and the number of steps can be represented by log n + 1(at most)

Also, we perform a single step operation to find out the middle of any subarray, i.e. O(1).

And to **merge** the subarrays, made by dividing the original array of n elements, a running time of O(n) will be required.

Hence the total time for mergeSort function will become n(log n + 1), which gives us a time complexity of O(n\*log n).

Worst Case Time Complexity [ Big-O ]: **O(n\*log n)**

Best Case Time Complexity [Big-omega]: **O(n\*log n)**

Average Time Complexity [Big-theta]: **O(n\*log n)**

Space Complexity: **O(n)**

* Time complexity of Merge Sort is O(n\*Log n) in all the 3 cases (worst, average and best) as merge sort always **divides** the array in two halves and takes linear time to **merge** two halves.
* It requires **equal amount of additional space** as the unsorted array. Hence its not at all recommended for searching large unsorted arrays.
* It is the best Sorting technique used for sorting **Linked Lists**.

**IntelliJ : knowMergeSort> DemoMergeSort**

**Quick Sorting:**

It is also called **partition-exchange sort**. This algorithm divides the list into three main parts:

1. Elements less than the **Pivot** element
2. Pivot element(Central element)
3. Elements greater than the pivot element

**Pivot** element can be any element from the array, it can be the first element, the last element or any random element. In this tutorial, we will take the rightmost element or the last element as **pivot**.

**For example:** In the array {52, 37, 63, 14, 17, 8, 6, 25}, we take 25 as **pivot**. So after the first pass, the list will be changed like this.

{6 8 17 14 **25** 63 37 52}

Hence after the first pass, pivot will be set at its position, with all the elements **smaller** to it on its left and all the elements **larger** than to its right. Now 6 8 17 14 and 63 37 52 are considered as two separate sunarrays, and same recursive logic will be applied on them, and we will keep doing this until the complete array is sorted.



### Complexity Analysis of Quick Sort

For an array, in which **partitioning** leads to unbalanced subarrays, to an extent where on the left side there are no elements, with all the elements greater than the **pivot**, hence on the right side.

And if keep on getting unbalanced subarrays, then the running time is the worst case, which is O(n2)

Where as if **partitioning** leads to almost equal subarrays, then the running time is the best, with time complexity as **O(n\*log n)**.

Worst Case Time Complexity [ Big-O ]: **O(n2)**

Best Case Time Complexity [Big-omega]: **O(n\*log n)**

Average Time Complexity [Big-theta]: **O(n\*log n)**

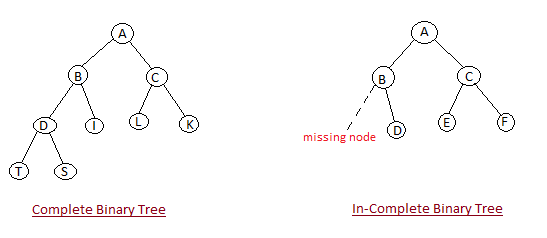
Space Complexity: **O(n\*log n)**

**IntelliJ : knowQuickSort> DemoQuickSort**

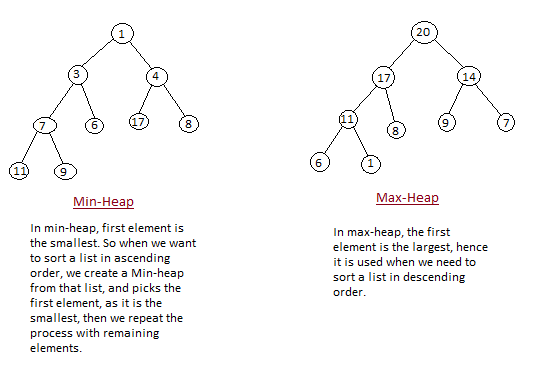
**Heap Sorting:**

Heap is a special tree-based data structure, that satisfies the following special heap properties:

1. **Shape Property:** Heap data structure is always a Complete Binary Tree, which means all levels of the tree are fully filled.



1. **Heap Property:** All nodes are either **greater than or equal to** or **less than or equal to** each of its children. If the parent nodes are greater than their child nodes, heap is called a **Max-Heap**, and if the parent nodes are smaller than their child nodes, heap is called **Min-Heap**.



Heap sort algorithm is divided into two basic parts:

* Creating a Heap of the unsorted list/array.
* Then a sorted array is created by repeatedly removing the largest/smallest element from the heap, and inserting it into the array. The heap is reconstructed after each removal.

Initially on receiving an unsorted list, the first step in heap sort is to create a Heap data structure(Max-Heap or Min-Heap). Once heap is built, the first element of the Heap is either largest or smallest(depending upon Max-Heap or Min-Heap), so we put the first element of the heap in our array. Then we again make heap using the remaining elements, to again pick the first element of the heap and put it into the array. We keep on doing the same repeatedly untill we have the complete sorted list in our array.

In the below algorithm, initially heapsort() function is called, which calls heapify() to build the heap.

Input data: 4, 10, 3, 5, 1

4(0)

/ \

10(1) 3(2)

/ \

5(3) 1(4)

The numbers in bracket represent the indices in the array

representation of data.

Applying heapify procedure to index 1:

4(0)

/ \

10(1) 3(2)

/ \

5(3) 1(4)

Applying heapify procedure to index 0:

10(0)

/ \

5(1) 3(2)

/ \

4(3) 1(4)

The heapify procedure calls itself recursively to build heap

in top down manner.