Collection / Framework / Api all of them refers to same

* If we want to save data / information in the program we use variables

eg : int age , String name

* If the data we need to store is smaller amount then variables are good for that , but if we to store large amount of data it is difficult to so many variables . due to which the storage increases and performance decreases
* In that case we use array for that because

1. Single variable name
2. Store large volume of data
3. Index based data structure

Disadvantages:

1. Store similar / homogenous data cannot store heterogenous data
2. Size is fixed
3. Demands contagious memory locations
4. No inbuilt methods are present in the array , need to pass it to utility class Arrays , so it is a bit complicated.

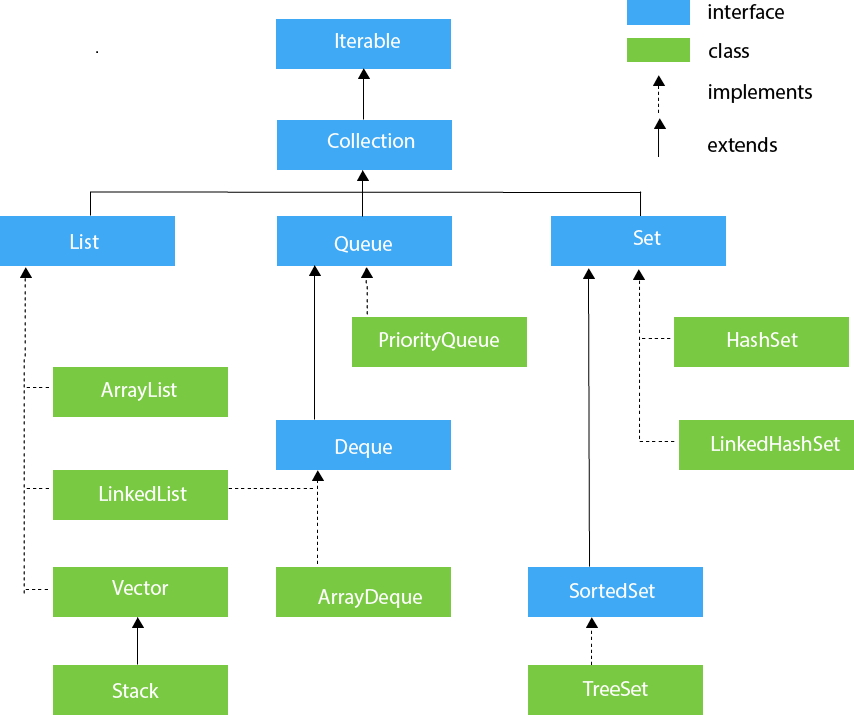
To store large amount of data in array there are many disadvantages

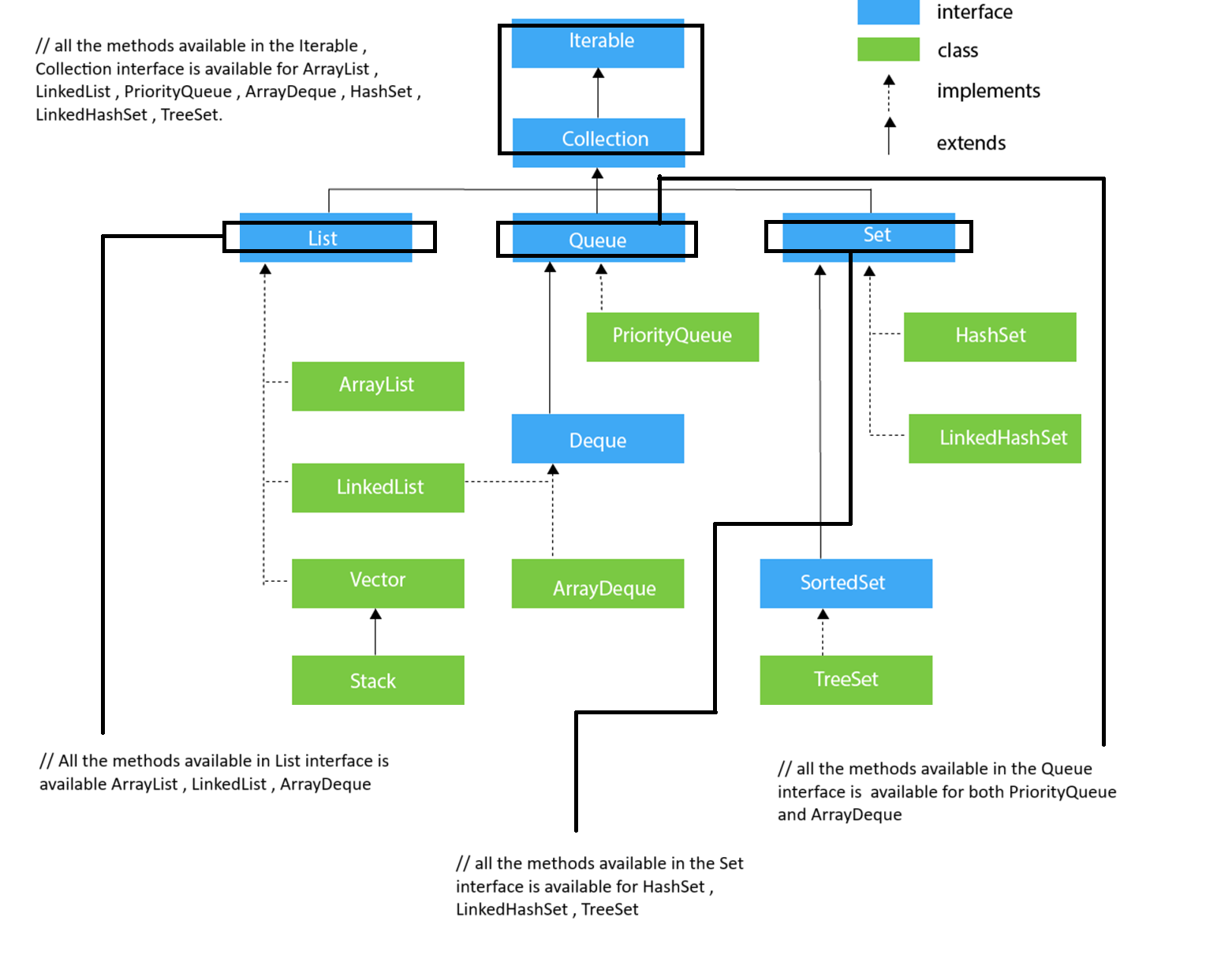
eg : there are different fields in the adhar like phone no , address , date of birth , name etc , all these fields are of different data types . we cannot store them in a single array. Due to array drawbacks.

To solve this problem java introduced legacy classes eg : vector , legacy classes means once they are used right now they are not used . because even legacy classes also lacks some features like uniform data structure etc.

Since java is open source joshva proposed collection API to sun microsystems. Which is then added to java from java 1.2 by sun team.

Hierarchy of collection framework :





Note :

List l = new LinkedList() ;

Loose coupling

// Using parent reference u cannot access specialized methods of child class.

LinkedList = new LinkedList();

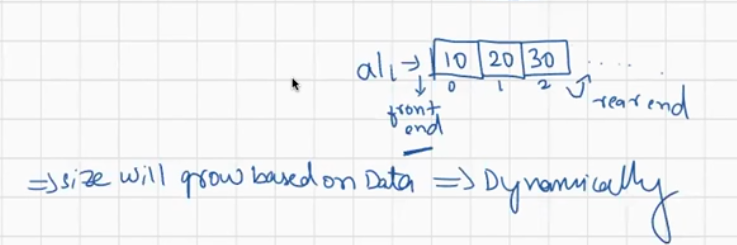
Tight coupling

// Using tight coupling all the methods of child and parent class can be accessed.

ArrayList

**ArrayList** class uses a dynamic [array](https://www.javatpoint.com/array-in-java)  data structure for storing the elements.





When try to add the object every time the size will keep on increasing. Data is stored in collection as object not like primitives .

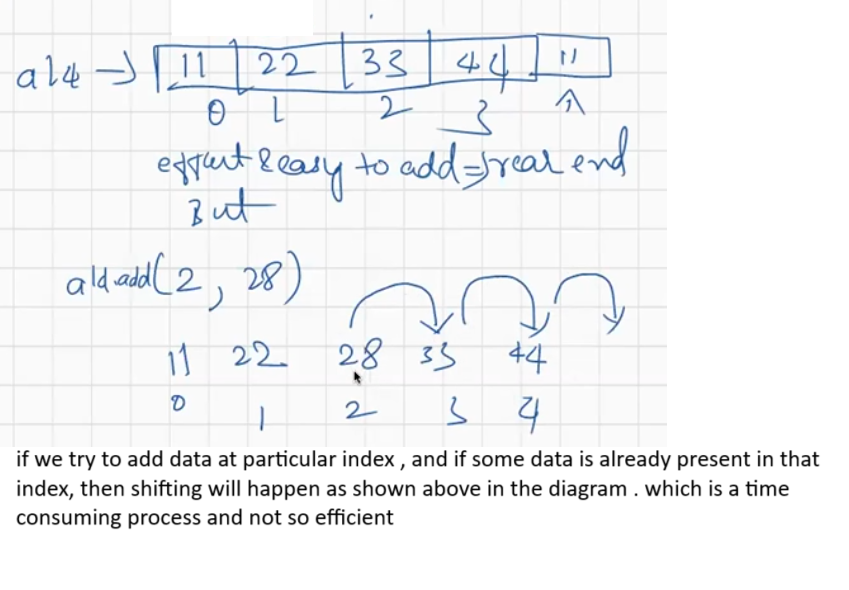
data structure means if you use this class to store the data, then that data in the memory is stored by following one structure.

It is like an array, but there is no size limit. We can add or remove elements anytime. So, it is much more flexible than the traditional array. It is found in the java.util package.

In ArrayList we can store both homogenous ( similar ) data / heterogenous data .

We can add entire collection to another collection ( check it in the below program)

We can add object in ArrayList at any index of our choice , but not recommended.



In ArrayList index based accessing is allowed.

Eg: ArrayList\_Eg1

// go through the code

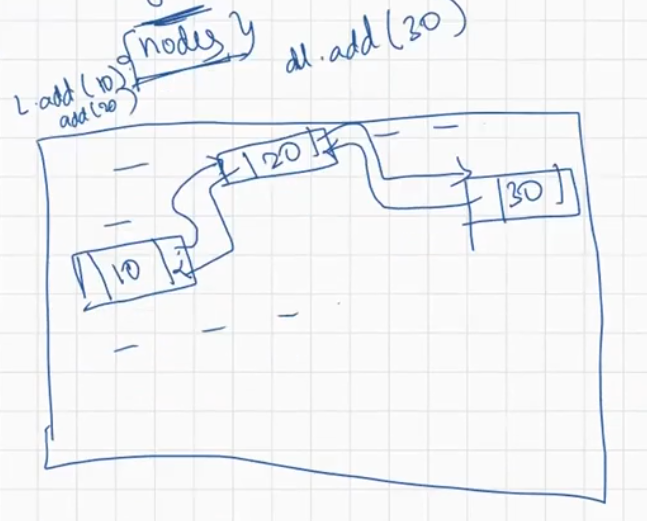
# LinkedList

Java LinkedList class internally uses a doubly linked list to store the elements.

In LinkedList we can store homogenous data, as well as heterogenous data

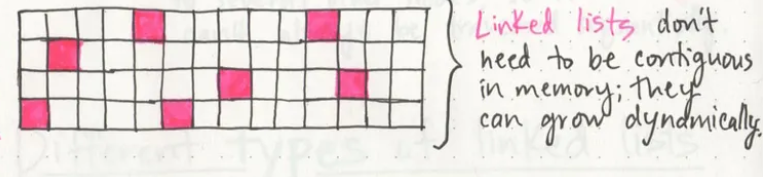
Every element is stored as Object.

All the methods available in ArrayList is also available for LinkedList also.

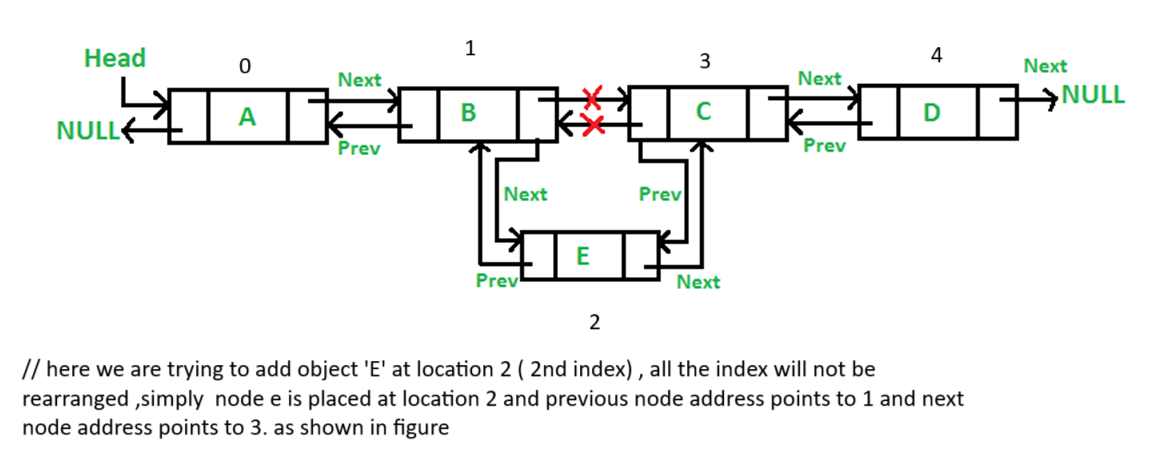


LinkedList internally uses doubly LinkedList data structure to store data.

LinkedList doesn’t need contagious memory locations . it can make use of disperse memory locations.



If we want to store object at particular location then simply



We can add data at any index of our choice in LinkedList

We can add both homogenous data , heterogenous data.

In both ArrayList and Linked every data is stored as a object.

After these concepts is Array concept gone or outdated ?

Ans : no

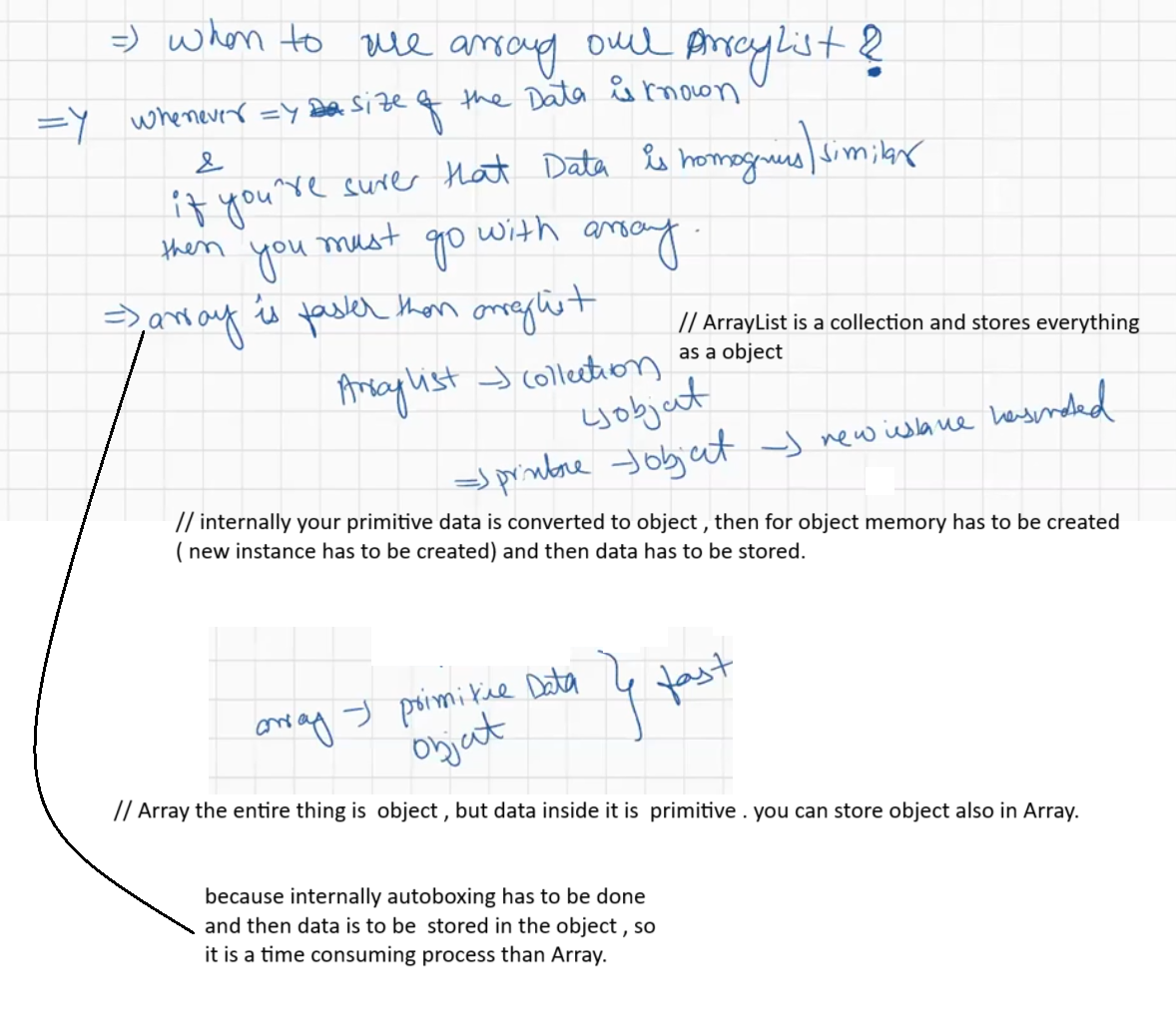
Whenever we are storing large amount of data we can use either

Array | collection

(LinkedList , ArrayList )

Eg: LinkedList\_Eg1

//go through the code



If you want to perform insertion at any index then ArrayList is not Suggestable , LinkedList is good for that.

In both ArrayList and LinkedList duplicates are allowed.

Eg: ArrayList\_Methods

// go through the code

ArrayList is efficient for insertion and deletion at rear end

Using inbuild methods we can perform operations on ArrayList.

Eg: LinkedList\_Methods

// go through the code

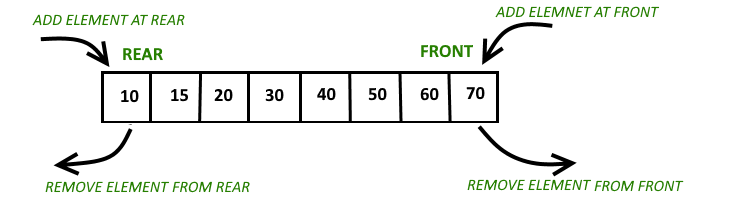
ArrayDeque:

Follows Double Ended queue data structure internally to store the elements.

Queue :



Double ended queue :



Index based accessing and inserting is not allowed. Can insert only at front and rear

Duplicates are allowed.

Array-Dqueue is useful when there is requirement for insertion and deletion at only front and rear end.

You can add heterogenous data

Eg: ArrayDeque

// go through the code

Priority Queue :

It implements min heap data structure internally

Duplicates are allowed

// go through the min heap lecture

<https://www.youtube.com/watch?v=mnPDP37RAGY>

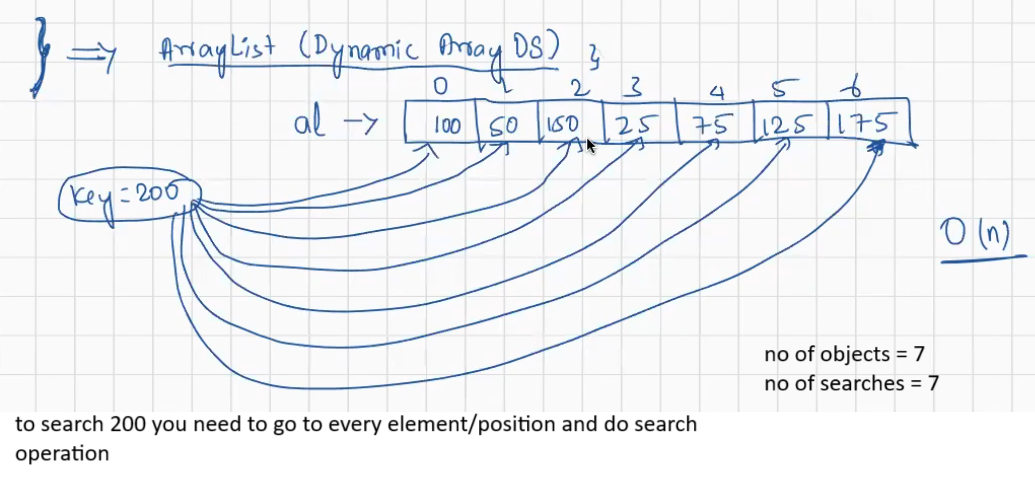
it accepts only homogenous data .

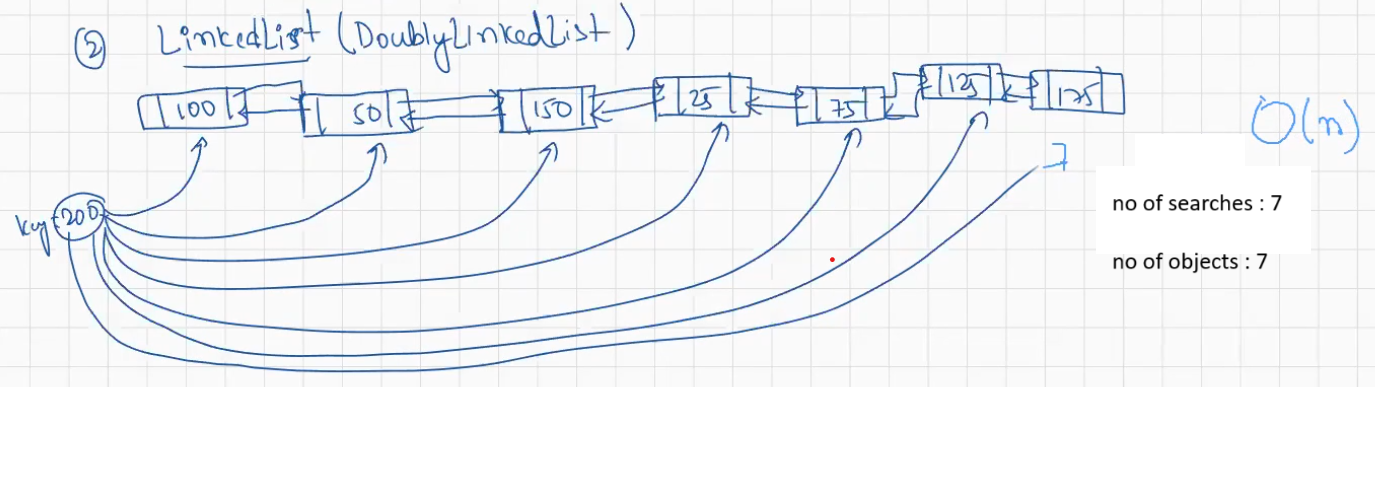
number data is sorted with min heap, other than number like String sorting will not happen.

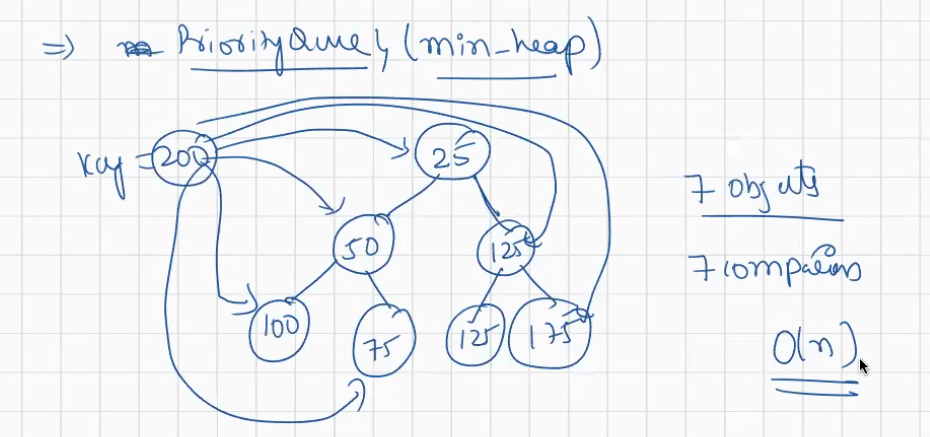
Cannot add elements at specific index you want , but can remove elements at any position , not based on index , by passing the object as argument directly.

Eg: PriorityQueue\_Eg1

// go through the code







TreeSet:

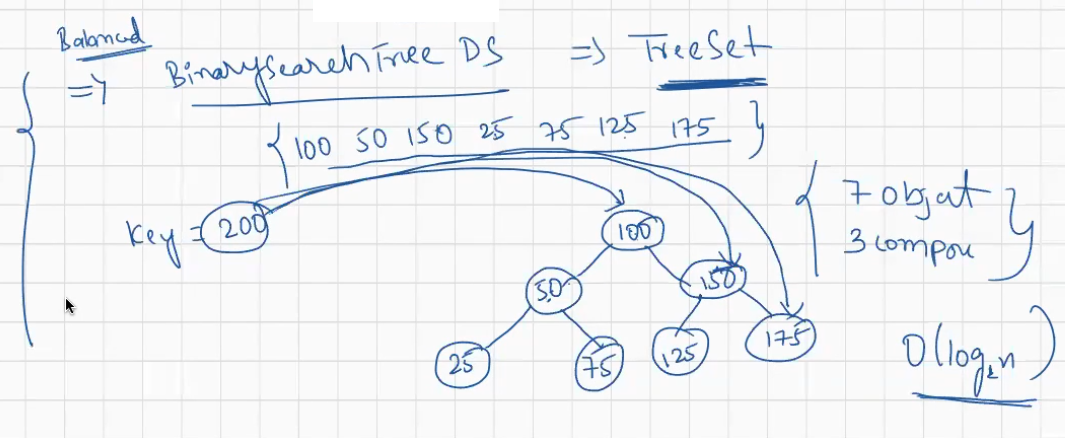
It internally uses Binary search tree data structure

Go through the binary search tree videos

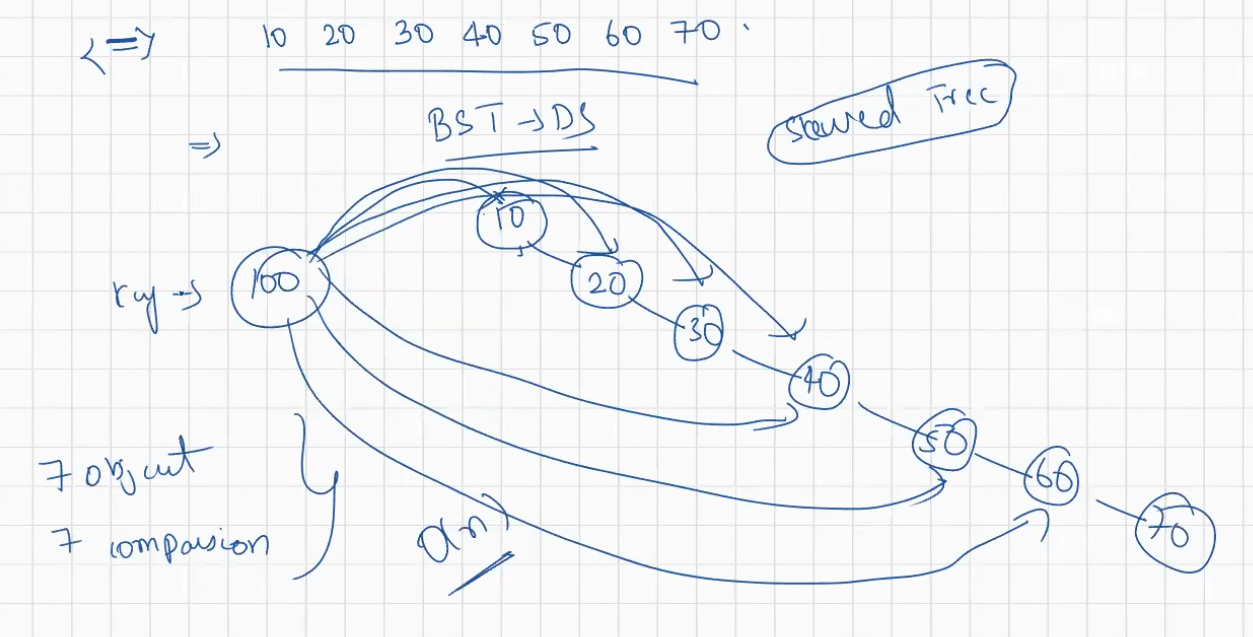
Supplicates are not allowed

<https://www.youtube.com/watch?v=YAdLFsTG70w&list=PLAuemBFSZV1oE256PESc8ku8MX3xMQYJF>

<https://www.youtube.com/watch?v=-b2lciNd2L4>



The searching operation takes less time incase of binary search tree as compared to the arraylist, linked list , array deque , priority queue.



Incase of skewed bst tree searching will take same time as arraylist, linked list , array deque , priority queue.

Eg: TreeSet\_Eg1

HashSet :

HashSet internally uses hashing algorithm .

<https://www.youtube.com/watch?v=mFY0J5W8Udk>

HashSet will not maintain order of insertion , however the searching will be very fast

We cant expect the output of the hashset

Eg: HashSet\_Eg1

// go through the code

LinkedHashSet:

It is the subclass of the HashSet , it follows the same hashing algorithm but along with it linked list is used.

Linked HashSet maintains order of insertion.

Eg: LinkedHashSet\_Eg1