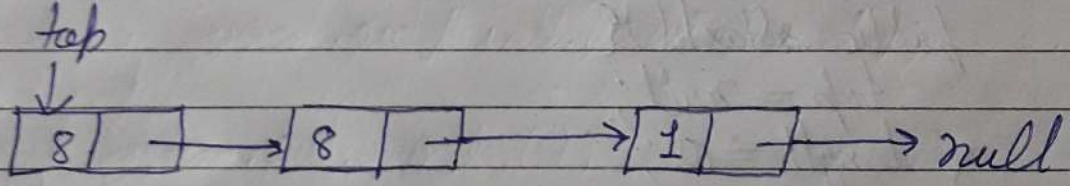
Stack Data Structure.

→ It is a linear data structure used for storing the data.

→ It is an ordered list in which insertion and deletion are done at one end, called as top.

→ The last element inserted is the first one to be deleted. hence it is called as Last in first Out (LIFO) list.
or FILO



push → add( )

pop → remove( )

Represent Stack.

```
import java.util.EmptyStackException;

public class Stack{
    private ListNode top;
    private int length;

    private class ListNode{
        private int data; //can be a generic ty
        private ListNode next; //Reference to n
                               ListNode in list

        public ListNode(int data){
            this.data = data;
            this.next = null;
        }
    }
    public Stack(){
        top = null;
        length = 0;
    }

    public int length(){
        return length;
    }
    public boolean isEmpty(){
        return length == 0;
    }

    public void push(int data){
        ListNode temp = new ListNode(data);
        temp.next = top;
        top = temp;
        length++;
    }
```

```
public int pop(){
  if (isEmpty()){
      throw new EmptyStackException();
  }
  int result = top.data;
  top = top.next;
  length--;
  return result;
}
public int peek(){
    if (isEmpty()){
        throw new EmptyStackException();
    }
    return top.data;
}
public static void main(String[] args){
   Stack stack = new Stack();
   stack.push(10);
   stack.push(15);
   stack.push(20);
   System.out.println(stack.peek());
   stack.pop();
   System.out.println(stack.peek());
   stack.pop();
   System.out.println(stack.peek());
}
}
```

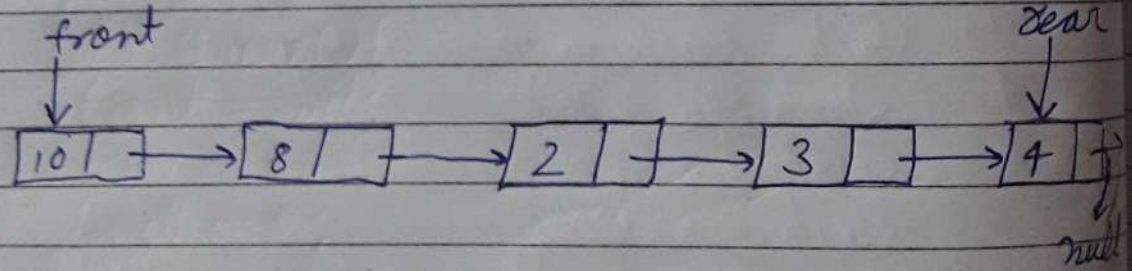
Output:- 20
15
10

Queue Data Structure.

→ It is a linear data structure used for storing the data.

→ It is an ordered list in which insertions are done at one end, called as rear and deletion are done at other end called as front.

→ The first element inserted is the first one to be deleted. Hence, it is called as First In First Out (FIFO) list



Queue

← Exit ——— Enter

If Queue is Empty.

front → null
rear → null

ENQUEUE → add()

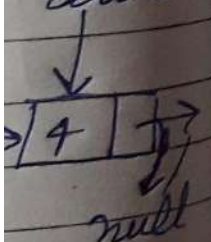
DEQUEUE → remove()

used

insertion
rear
end

first
called

rear



```
import java.util.NoSuchElementException;

public class Queue{
    private ListNode front;
    private ListNode rear;
    private int length;

    private class ListNode{
        private int data; //Can be a generic type
        private ListNode next; //Reference to next ListNode
                               in List
        public ListNode(int data){
            this.data = data;
            this.next = null;
        }
    }
    public Queue(){
     front = null;
     rear = null;
     length = 0;
    }
    public int length(){
      return length;
    }
    public boolean isEmpty(){
       return length==0;
    }
    public void enqueue(int data){
      ListNode temp = new ListNode(data);
      if(isEmpty()){
```

```
            front = temp;
        }else{
            rear.next = temp;
        }
        rear = temp;
        length++;
    }
    public int dequeue(){
        if(isEmpty()){
            throw new NoSuchElementException("
                                Queue is already empty");
        }
        int result = front.data;
        front = front.next;
        if(front == null){
            rear = null;
        }
        length--;
        return result;
    }
    public void print(){
        if(isEmpty()){
            return;
        }
        ListNode current = front;
        while(current != null){
            System.out.print(current.data + " --> ");
            current = current.next;
        }
        System.out.println("null");
    }
```

```
public static void main(String[] args){
    Queue queue = new Queue();
    queue.enqueue(10);
    queue.enqueue(15);
    queue.enqueue(20);
    queue.print();
    queue.dequeue();
    queue.dequeue();
    queue.print();
  }
}
```

Output :- 10 --> 15 --> 20 --> null
20 --> null.

Best sorting algorithm sort an array in nlogn complexity.

SAGAR
Page No.
Date

Arrays

```
int a[] = new int[4];
a[2] = 4;
```

→ Find the majority element in an array.

→

a = | 1 | 2 | 3 | 1 | 1 | 4 |

$\frac{6}{2} = 3$

In this ~~element~~ array not majority element present

→ a = | 1 | 2 | 1 | 1 | 4 |

$\frac{5}{2} = 2$

In this array majority element is 1

```
import java.util.HashMap;

class Main
{
  private static void findMajority(int[] arr)
  {
    HashMap<Integer, Integer> map = new HashMap<
                                Integer, Integer>();
    for (int i=0; i<arr.length; i++) {
      if (map.containsKey(arr[i])) {
          int count = map.get(arr[i]) + 1;
          if (count > arr.length/2) {
              System.out.println("Majority found:-
                                  " + arr[i]);
              return;
          }else
              map.put(arr[i], 1);
          }
          System.out.println("No Majority element");
    }}
    public static void main(String[] args)
    {
    int a[] = new int[]{2,2,2,2,5,5,2,3,3};
    findMajority(a);
    }
}
```

output :- Majority found :- 2.

Q. Find the sub array in a given array whose sum is gratter then all subarrays.

```
int maxSubArray(int a[]){
    int maxSum = 0;
    int curSum = 0;
    for(int i=1; i<a.length; i++){
        curSum = curSum + a[i];
        if(curSum > maxSum){
            maxSum = curSum;
        }
        if(curSum < 0){
            curSum = 0;
        }
    }
    return maxSum;
  }
}
```

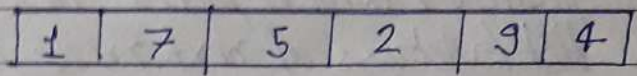
a = [5, -4, -2, 6, -1]

```
import java.util.*;
Class Main
{
public static void main(String[] args)
{
  int[] a = {-2, -3, 4, -1, -2, 1, 5, -3};
  System.out.println("Maximum contiguous sum is " +
                                  maxSubArraySum(a));
}
static int maxSubArraySum(int a[])
{
   int size = a.length;
   int max_so_far = Integer.MIN_VALUE,
                    max_ending_here = 0;
   for (int i = 0; i < size; i++)
   {
     max_ending_here = max_ending_here + a[i];
     if (max_so_far < max_ending_here)
         max_so_far = max_ending_here
     if (max_ending_here < 0)
        max_ending_here = 0;
   }
   return max_so_far;
}
}
```

Output:- Maximum contiguous sum is 7.

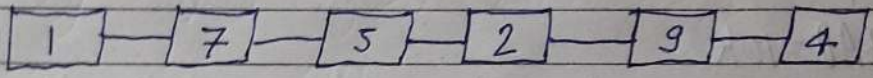
Linked List

Array



Single block of memory with partitions

Linked List



Multiple blocks of memory linked to each other

Limitations in Arrays

- Fixed size
- Contiguous block of memory
- Inserting or deleting is costly.

Properties of Linked List

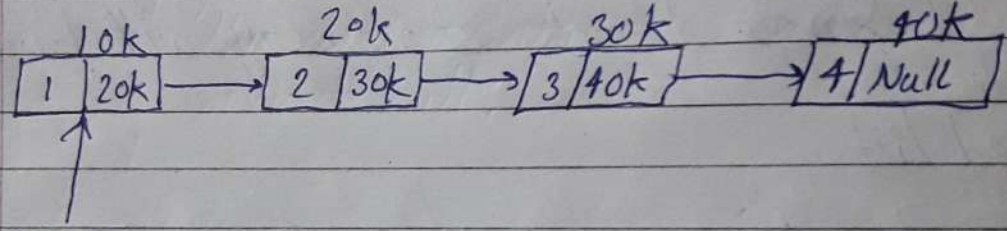
- Size can be modified
- Non-contiguous memory
- Insertion and deletion at any point is easier.

In java not addressive we use reference.

Structure of Linked List :-

NODE | Data | Next |

Data :- int, char, float ~~or~~ double etc.
Next :- it is pointer which point next node in the list (address of next node).



HEAD = 10k
(head pointer store the address of the first node of the linked list.

Important point :- If we find element in a list then Array list is faster then linked list.

→ Linkedlist is a linear data structures.

```
package linkedlists;
import java.util.*;

public class MainLinkedList{
        public static void main(String[] args){

Interface
    List<Integer> ll = new LinkedList<>();

    ll.add(12);
    ll.add(2);
    ll.add(32);

    System.out.println(ll.get(1));
    }
}
```

Nat. `ll.set(2,13);`

```
package linkedlistx;

public class MyLinkedList {
    static class Node {
        int data;
        Node next;
        public Node(int data) {
            this.data = data;
            next = null;
        }
    }

    Node head;
    void add(int data) {
        Node toAdd = new Node(data);
        Node temp = head;
        while (temp.next != null) {
            temp = temp.next;
        }
        temp.next = toAdd;
    }
```

node class

head node

If list is empty then.

```
Node head;
void add(int data) {
    Node toAdd = new Node(data);
    if (head == null) {
        head = toAdd;
        return;
    }
```

add 1

```
import java.util.*;

public class Ved extends Thread{
    public static void main(String[] args){
        LinkedList<Integer> li = new LinkedList<>();
        li.add(11);
        li.add(22);
        li.add(3,77);
        System.out.println("Li Linked list:"+li);
    }
}
```

Remove 2. remove()

```
li.remove(2);
```

set 3. set()

```
li.set(2,10);
```

output :- 11,22,10,77.

addLast 4 addLast() , addLast()

```
li.addLast(100)
```

output :- 11, 22, 10, 77, 100

addFirst 5 addFirst()

```
li.addFirst(0);
```

output :- 0, 11, 22, 10, 77, 100

6. contains()

contains

```
System.out.println(l1.contains(27));
```

Output:- False (because 27 is not in list)
True (because 27 is present in list)

7. indexOf()

indexOf

```
System.out.println(l1.indexOf(22));
```

Output:- 1

8. 788, 566, 18, 19, 1, 5, 6, 7, 9, 6, 676,
lastIndexOf()

```
System.out.println(l1.lastIndexOf(6));
```

Output:- 9

9.

```
for(int i=0; i<l1.size(); i++){
    System.out.print(l1.get(i));
    System.out.print(", ");
}
```

String → immutable

Java String are object that allows us to store sequence of character which may contain alpha numeric Values enclosed in double quoters ("Ram").

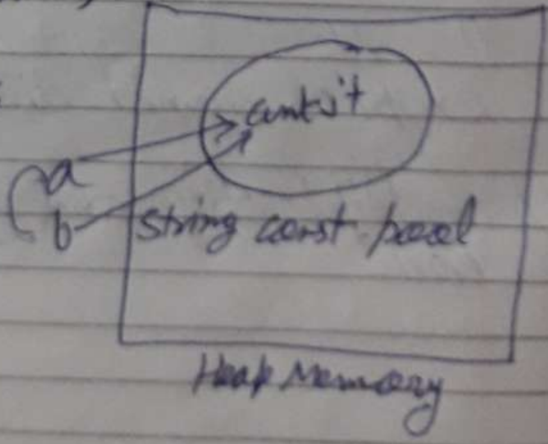
Note:- 1) String are immutable in java.
2) It contains methods that can perform certain operations on Strings.
(concat(), equals( ), length() etc...)

There are two way to create string object:-
① String literal
② new keyword

1. String literal.

String a = "ankit";

String b = "ankit";



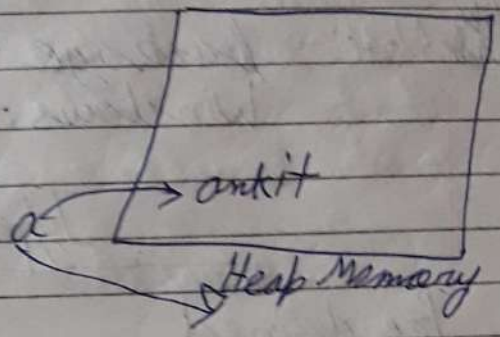
* string const pool
A memory build

String:- final predefine class
with the help of assignment operator (=) forcefully we can create string as a mutable.

2. new keyword:-
new is a keyword use to create dynamic memory.

String a = new String("ankit");

a.concat("kumar");x
thus String is immutable.



class A
{
public static void main(String[] args){
String a=new String ("ankit");
System.out.println(a);

String b=new String("ankit");
System.out.println(b);
a.concat("kumar");
System.out.println(a);
}
}

output ankit
ankit
ankit

if a=a.concat("kumar"); or a="kumar";
then last answer = ankitkumar.
forcefully string can be concatinate.