**DS Assignments**

1. Recursive algorithm is one that calls itself till a base condition.
   1. Study the following code to calculate factorial of a number.
   2. **public** **static** **long** fact(**long** number)

{

* + 1. System.**out**.println(number);
    2. **if**(number == 1) **return** 1;
    3. **long** f1 = fact(number-1);
    4. System.**out**.println("fact:"+f1);
    5. **return** number \* f1;

}

* 1. Based on the above approach write a program to print fibanocci series.

1. Create a datastructure called MyList that adds the first element of any type and keeps on adding only the same type , throws exception when any other type is added.
   1. Implement using array that increases its capacity for 3 elements.
   2. Sample client code
      1. System.**out**.println(list1.add("abc"));
      2. System.**out**.println(list1.add(1));
      3. System.**out**.println(list1.add("xyz"));
      4. System.**out**.println(list1);
      5. System.**out**.println(list1.add("xyz"));
      6. System.**out**.println(list1.add(1.7));
      7. System.**out**.println(list1.add("xyz"));
         1. System.**out**.println(list1);
      8. **Sample Output:**
      9. abc added
      10. Type MisMatch
      11. xyz added
      12. [abc xyz ]
      13. xyz added
      14. Type MisMatch
      15. xyz added
      16. [abc xyz xyz xyz ]
2. B. Create a class MyArrays that has a method
   1. **public** **static** MyList asList(Object... args)
   2. {
   3. //creates and returns a list with args.
   4. }
3. Create a class MyLinkedList<T> that has following implementations for a singly linked list.

A Adding element at the beginning.

B Deleting element from the beginning.

C Displays elements in the order.

D Implement the above using doubly linked list.

1. Implement Queue implementations.
   1. Enque(..) adds element from Rear.
   2. Deque () processes element from front.

Sample main method

**public** **static** **void** main(String[] args) {

Queue queue = **new** Queue();

queue.enque(10);

queue.display();

queue.enque(20);

queue.display();

queue.enque(30);

queue.display();

queue.deque();queue.display();

queue.deque();queue.display();

queue.deque();queue.display();

queue.deque();

}

Sample output:

Front: 10-->: Rear

Front: 10-->20-->: Rear

Front: 10-->20-->30-->: Rear

data dequed:10

Front: 20-->30-->: Rear

data dequed:20

Front: 30-->: Rear

data dequed:30

Front: : Rear

Queue Empty

1. C. Add method to implement LinearSearch on the list.
   1. MyArrays.linearSearch(array,key) that returns position of the element or -1 if not found.
2. D. Add method to implemeny BinarySearch on the list.
   1. MyArrays.binarySearch(array,key) that returns position of the element or -1 if not found.
   2. Using Iterative algorithm.
   3. Recursive algorithm.
3. Write a java program to evaluate prefix expression.
   1. Eg: +\*2354
4. Write a Java Program that reads an expression from the user and checks if brackets are closed properly.
   1. Eg : (3+4)\*(5/2) -- > results true
   2. ((2-1) \*(4\*5) 🡪 results false

9.

**public** **class** TreeSetDemo {

**public** **static** **void** main(String[] args) {

TreeSet<Name> names=**new** TreeSet<>();

names.add(**new** Name("raj", "mishra"));

names.add(**new** Name("wills", "smith"));

names.add(**new** Name("steve", "Smith"));

names.add(**new** Name("walter", "Robert"));

names.add(**new** Name("sam", "smith"));

System.***out***.println(names);

}

}

Complete the above program and observe the working of TreeSet.

1. **public** **class** QuickSort1 {
   1. **public** **static** **void** main(String[] args) {
      1. **int** i;
      2. **int**[] arr={10,16,8,12,15,6,3,9,5,17,11,7};
      3. quickSort(arr, 0, arr.length-1);
      4. System.**out**.println("\n The sorted array is: \n");
      5. **for**(i=0;i<arr.length;i++)
         1. System.**out**.println(arr[i]);
   2. }
   3. **public** **static** **int** partition(**int** a[], **int** start, **int** end)
   4. {
      1. **int** i, j, temp, pivot;
      2. pivot = i = start;
      3. j = end;
      4. **boolean** flag = **false**;
      5. **while**(!flag)
      6. {
         1. // iterate from right to left till the element is less than the pivot element.
         2. **if**(pivot==j)
            1. flag =**true**;
         3. **else** **if**(a[pivot]>a[j])
         4. {
            1. //swap
         5. }
         6. **if**(!flag)
         7. {
         8. // iterate from left to right till the element is less than the pivot element.
            1. **while**((a[i] < a[pivot]) && (pivot!=i))

i++;

* + - * 1. **if**(pivot==i)

flag =**true**;

* + - * 1. **else** **if**(a[pivot] <a[i])
        2. {

temp = a[pivot];

a[pivot] = a[i];

a[i] = temp;

pivot = i;

* + - * 1. }
      1. }
    1. }
    2. **return** pivot;
  1. }
  2. **static** **void** quickSort(**int** a[], **int** start, **int** end)
  3. {
     1. **int** pivot;
     2. **if**(start<end)
     3. {
        1. pivot = partition(a, start, end);
        2. quickSort(a, start, pivot-1);
        3. quickSort(a, pivot+1, end);
     4. }
  4. }

1. }
2. 5.