🔁

Recursion

Check if the array is sorted

```
import java.util.*;

public class Practice {
    public static void main(String[] args) {
        int[] arr = { 2, 4, 6, 8, 10 };
        boolean b = recur(arr, 0);
        System.out.println(b);
    }

    public static boolean recur(int[] arr, int index) {
        if (index == arr.length - 1) {
            return true;
        } else if (arr[index] > arr[index + 1]) {
            return false;
        } else {
            return recur(arr, index + 1);
        }

    }
}
```

Fibonacci series

```
public class Recursion {
    public static void main(String[] args) {
        int n = 6;
        int ans = fibo(n);
        System.out.println(ans);
    }

    static int fibo(int n) {

        if (n == 0)
            return 0;

        if (n == 1)
            return 1;

        else {
            return fibo(n - 1) + fibo(n - 2);
        }
    }

}
```

Binary search using Recursion

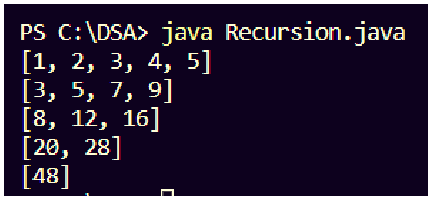
```
public class Recursion {
    public static void main(String[] args) {
        int[] arr = { 2, 5, 8, 12, 80, 99 };
        int low = 0;
        int high = arr.length - 1;
        int target = 89;
        int ans = BS(arr, target, low, high);
        System.out.println(ans);
    }

    static int BS(int[] arr, int target, int low, int high) {

        int mid = low + (high - low) / 2;

        if (low > high) {
            return -1;
        }
        if (arr[mid] == target)
            return mid;

        if (target > arr[mid]) {
            return BS(arr, target, mid + 1, high);
        }
        return BS(arr, target, low, mid - 1);

    }

}
```

Special triangle (inverse) -

Sum triangle from array - GeeksforGeeks

A Computer Science portal for geeks. It contains well written, well thought and well explained computer science and programming articles, quizzes and practice/competitive programming/company interview Questions.

https://www.geeksforgeeks.org/sum-triangle-from-array/



```
PS C:\DSA> java Recursion.java
[1, 2, 3, 4, 5]
[3, 5, 7, 9]
[8, 12, 16]
[20, 28]
[48]
```

```
import java.util.ArrayList;
import java.util.Arrays;

public class Recursion {
    public static void main(String[] args) {
        int[] arr = { 1, 2, 3, 4, 5 };
        System.out.println(Arrays.toString(arr));
        int[] nums = pattern(arr, arr.length);
    }

    static int[] pattern(int[] arr, int n) {
        int[] nums = new int[n - 1];
        if (n == 1) {
            return nums;
        }

        for (int i = 0; i < n - 1; i++) {
            int nextitem = arr[i] + arr[i + 1];
            nums[i] = nextitem;
        }

        System.out.println(Arrays.toString(nums));
        return pattern(nums, n - 1);
    }

}
```

solution -

```
import java.util.*;
import java.lang.*;

public class Recursion {
    // Function to generate Special Triangle.
    public static void printTriangle(int[] arr) {
        // Base case
        if (arr.length < 1)
            return;

        // Creating new array which contains the
        // Sum of consecutive elements in
        // the array passes as parameter.
        int[] nums = new int[arr.length - 1];
        for (int i = 0; i < arr.length - 1; i++) {
            int x = arr[i] + arr[i + 1];
            nums[i] = x;
        }

        // Make a recursive call and pass
        // the newly created array
        printTriangle(nums);

        // Print current array in the end so
        // that smaller arrays are printed first
        System.out.println(Arrays.toString(arr));
    }

    // Driver function
    public static void main(String[] args) {
        int[] arr = { 1, 2, 3, 4, 5 };
        printTriangle(arr);
    }
}
```

344. Reverse String

```
class Solution {
    public void reverseString(char[] s) {

    char[] ans = reverse(s, 0, s.length - 1);
    System.out.println(Arrays.toString(ans));
    }

    public static char[] reverse(char[] c, int low, int high) {
        if (low > high) {
            return c;
        }
        char temp = c[low];
        c[low] = c[high];
        c[high] = temp;
        // System.out.println(Arrays.toString(c));
        return reverse(c, low + 1, high - 1);
    }
}
```

First uppercase letter in a string -

First uppercase letter in a string (Iterative and Recursive) - GeeksforGeeks

A Computer Science portal for geeks. It contains well written, well thought and well explained computer science and programming articles, quizzes and practice/competitive programming/company interview Questions.

https://www.geeksforgeeks.org/first-uppercase-letter-in-a-string-iterative-and-recursive/

```
public class Recursion {
    public static void main(String[] args) {
        String s = "smaraNi";
        char ans = reverse(s, 0);
        System.out.println(ans);
    }

    public static char reverse(String s, int i) {

        if (i == s.length())
            return 'Z';
        char c = s.charAt(i);
        int ascii = c;

        if (ascii >= 65 && ascii <= 90)
            return c;

        return reverse(s, (i + 1));

    }
}
```

GFG code -

```
// Java program to create Special triangle.
import java.util.*;
import java.lang.*;
```

```
public class Recursion {
    public static void main(String[] args) {
        String str = "geeKS";
        char res = reverse(str, 0);
        if (res == 0)
            System.out.println("No uppercase letter");
        else
            System.out.println(res);
    }

    public static char reverse(String str, int i) {

        if (str.charAt(i) == '\0') {
            return 0;
        }
        if (Character.isUpperCase(str.charAt(i))) {
            return str.charAt(i);
        }
        try {
            return reverse(str, i + 1);
        } catch (Exception e) {
            System.out.println("Exception occurs");
        }
        return 0;
    }
}
```

231. Power of Two

```
class Solution {
    public boolean isPowerOfTwo(int n) {
        boolean b = power(n) ;
        return b ;
    }

    public boolean power(int n){
    if(n==0)
    return false ;
    else if(n==1)
        return true ;
    else if(n%2==0){
          n/=2 ;
          return power(n) ;
      }
      else
      return false ;

    }
}
```

326. Power of Three

```
class Solution {
    public boolean isPowerOfThree(int n) {
        boolean b = power(n);
        return b;
    }
    public static boolean power(int n) {
        if( n==0 )
        return false  ;
       else if (n == 1)
            return true;
        else if (n % 3 == 0) {
            n /= 3;
            return power(n);
        } else
            return false;
    }
}
```

1342. Number of Steps to Reduce a Number to Zero

```
class Solution {
    public int numberOfSteps(int num) {
        int count  = 0 ;
        int ans = zero(num , count) ;
        return ans ;
    }

    public int zero( int num , int count){
       if(num==0){
           return count ;
       }
        if(num%2 == 0)
        num/=2 ;

        else
        num-- ;
        count++ ;

        return zero(num,count) ;
    }
}
```

Write a recursive function that returns the factorial of a number.

Day 9: Recursion 3 | HackerRank

Objective Today, we are learning about an algorithmic concept called recursion. Check out the Tutorial tab for learning materials and an instructional video. Recursive Method for Calculating Factorial Function Description Complete the factorial function in the editor below. Be sure to use recursion.

https://www.hackerrank.com/challenges/30-recursion/problem

```
// Java program to create Special triangle.
import java.util.*;
import java.lang.*;

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

        int n = 5;
        int ans = factorial(n);
        System.out.println(ans);
    }

    public static int factorial(int n) {
        if (n == 1)
            return 1;

        return n * factorial(n - 1);
    }
}
```

Check if array is sorted -

Check if array is sorted | Practice | GeeksforGeeks

Given an array arr[] of size N, check if it is sorted in non-decreasing order or not. Example 1: Input: N = 5 arr[] = {10, 20, 30, 40, 50} Output: 1 Explanation: The given array is sorted. Example 2: Input: N = 6 ar

https://practice.geeksforgeeks.org/problems/check-if-an-array-is-sorted0701/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article

```
class Solution {
    boolean arraySortedOrNot(int[] arr, int n) {
     int low = 0;
        int high = 1;
        boolean ans = factorial(arr, low, high);
return ans;
    }

       public static boolean factorial(int[] arr, int low, int high) {
        if (high > arr.length - 1)
            return true;

        if (arr[low] > arr[high])
            return false;

        return factorial(arr, low + 1, high + 1);

    }

}
```

Subsequences code

```
import java.util.*;
import java.lang.*;

public class Recursion {
    public static void main(String[] args) {
        int[] arr = { 3, 1, 2 };
        subsequences(arr, new ArrayList<>(), 0);
    }

    public static void subsequences(int[] arr, ArrayList<Integer> list, int index) {
        if (index >= arr.length) {
            System.out.println(list);
            return;
        }
        list.add(arr[index]);
        subsequences(arr, list, index + 1);
        list.remove(Integer.valueOf(arr[index]));
        subsequences(arr, list, index + 1);

    }
}
```

OR

78. Subsets

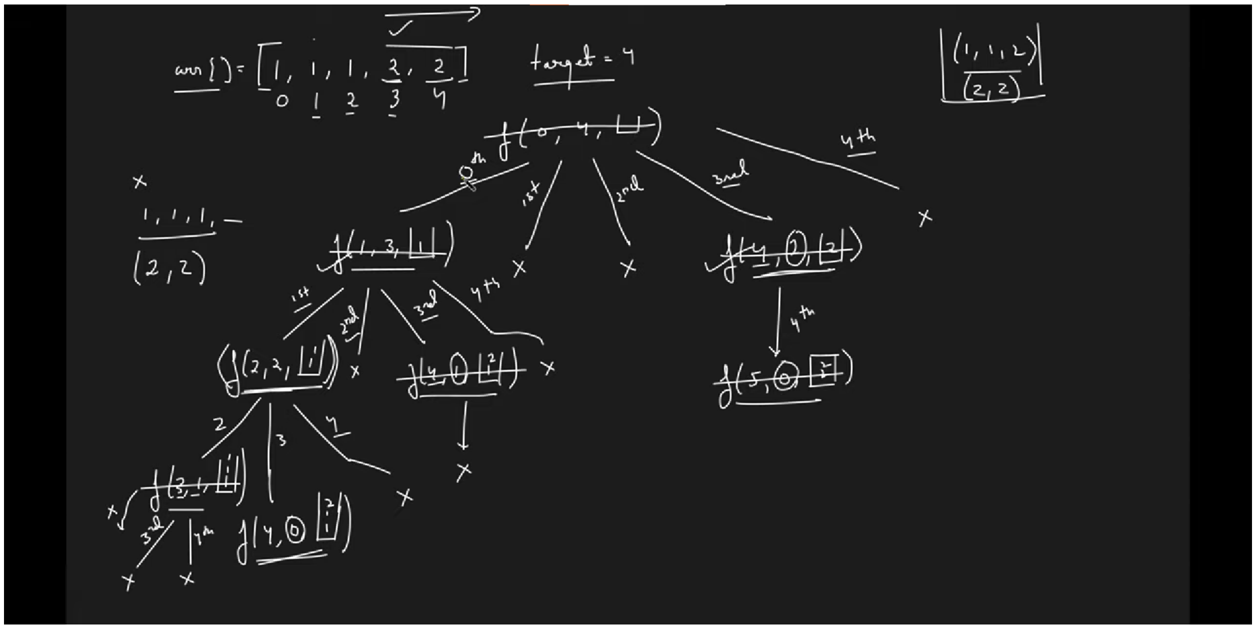
```
class Solution {
    public List<List<Integer>> subsets(int[] nums) {
        List<List<Integer>> finalList = new ArrayList<>() ;
        subsequences(nums, new ArrayList<>(), 0, finalList);
        return finalList;
    }

    public void subsequences(int[] arr, ArrayList<Integer> list, int index,List<List<Integer>> finalList ) {
        if (index >= arr.length) {
            finalList.add(new ArrayList<>(list));
            return;

        }
        list.add(arr[index]);
        subsequences(arr, list, index + 1,finalList);
        list.remove(Integer.valueOf(arr[index]));
        subsequences(arr, list, index + 1 ,finalList);

    }
}
```

39. Combination Sum (Strivers)

```
class Solution {
    public List<List<Integer>> combinationSum(int[] candidates, int target) {
        List<List<Integer>> finalList = new ArrayList<>() ;
       subsequences(candidates, new ArrayList<>(), 0, target, finalList);
        return finalList ;
    }

 public static void subsequences(int[] arr, ArrayList<Integer> list, int index, int target,
            List<List<Integer>> finalList) {
        if (arr.length == index) {
            if (target == 0) {
                finalList.add(new ArrayList<>(list));

            }
            return;
        }
        if (arr[index] <= target) {
            list.add(arr[index]);
            subsequences(arr, list, index, target - arr[index], finalList);
            list.remove(list.size() - 1);
        }
        subsequences(arr, list, index + 1, target, finalList);
    }
}
```

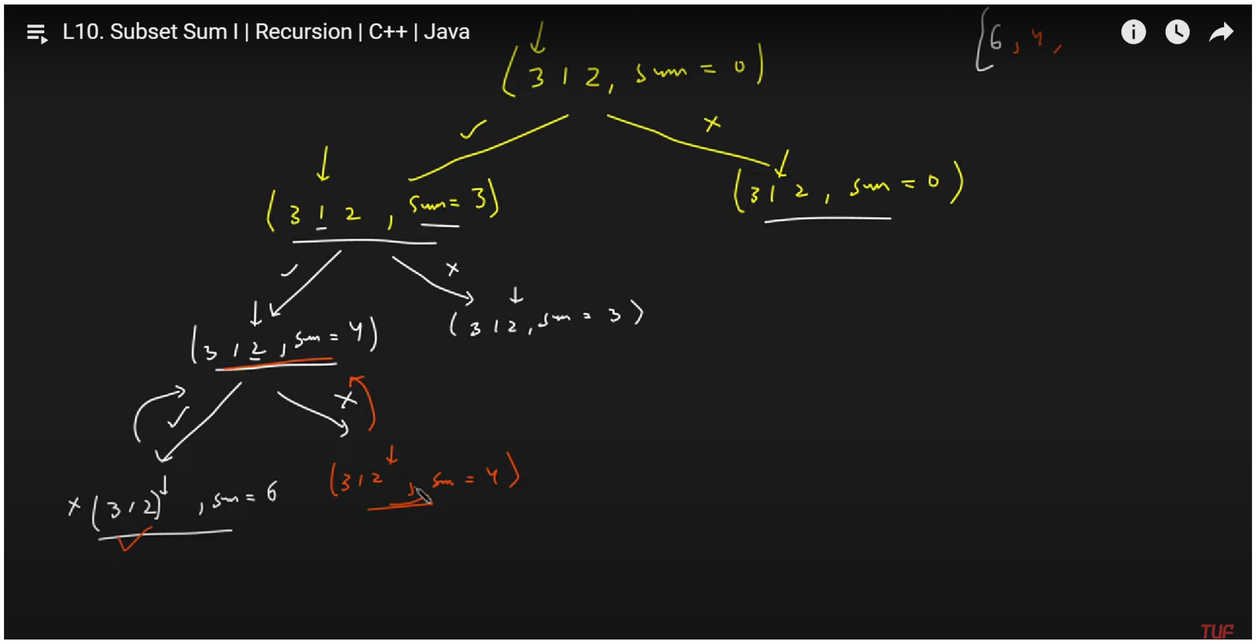
40. Combination Sum II (Strivers)

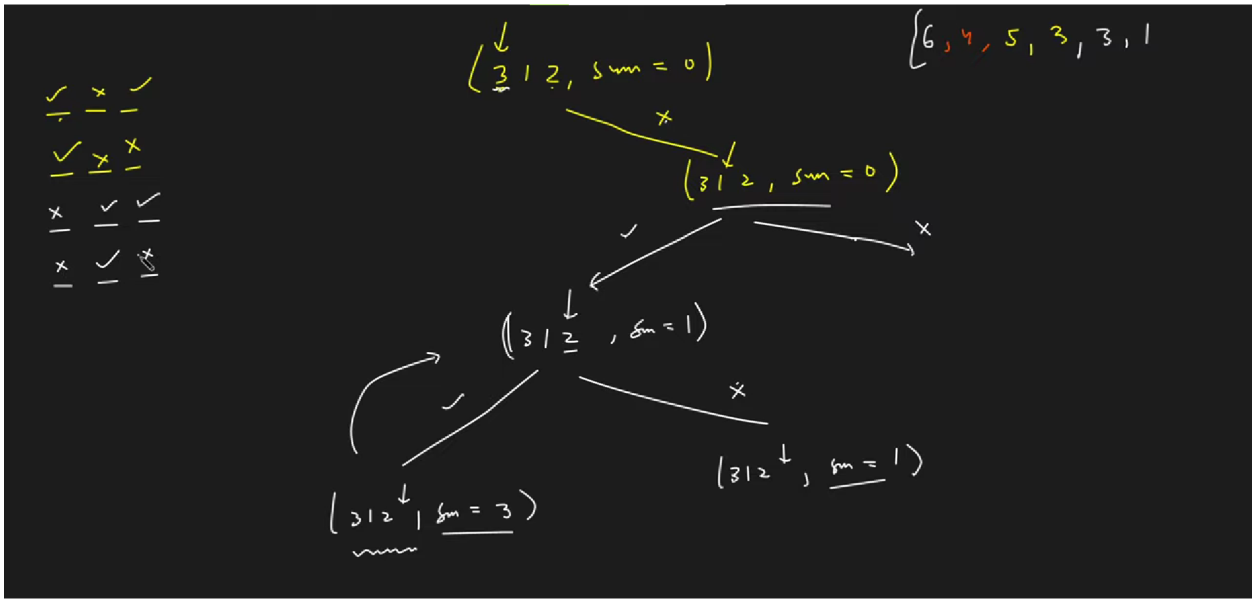
💡 The array is to be sorted for the solution to work. Because it will compare an index to the next index so that there is no duplication of elements .



```
class Solution {
    public List<List<Integer>> combinationSum2(int[] candidates, int target) {
       List<List<Integer>> finalList = new ArrayList<>() ;
       Arrays.sort(candidates) ;
       subsequences(candidates, new ArrayList<>() , 0 , target , finalList );
       return finalList ;
    }
        public static void subsequences(int[] arr, ArrayList<Integer> list, int index, int target,List<List<Integer>> finalList) {
        if (target == 0) {
            finalList.add(new ArrayList<>(list));
            return;
        }
        for (int i = index; i < arr.length; i++) {
            if (i > index && arr[i] == arr[i - 1]) {
                continue;
            }
            if (arr[i] > target)
                break;
            list.add(arr[i]);
            subsequences(arr, list, i + 1, target - arr[i], finalList);
            list.remove(list.size() - 1);
        }
    }
}
```

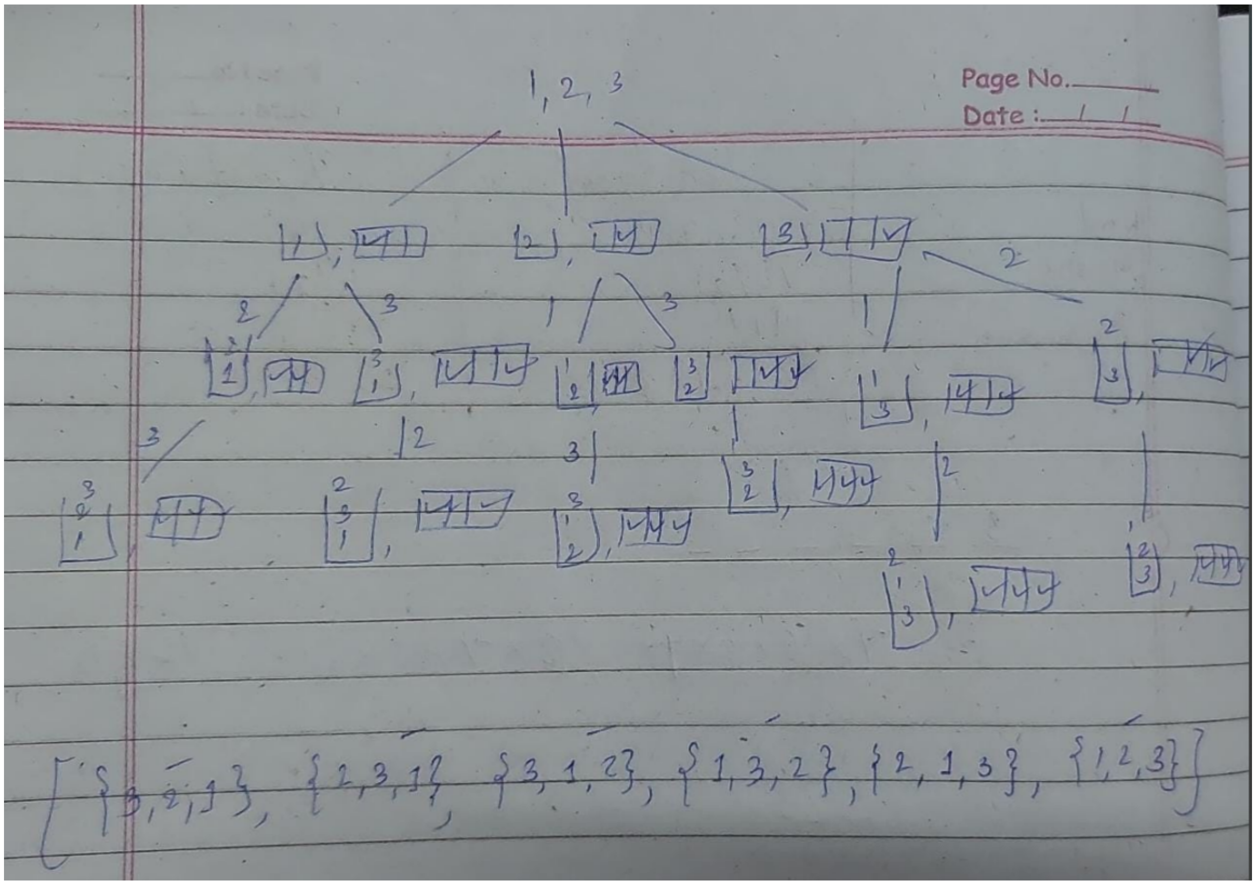
Subset sum I (not leetcode)





```
public class Recursion1 {
    public static void main(String[] args) {
        int[] arr = { 3, 1, 2, 4 };
        ArrayList<Integer> list = new ArrayList<>();
        subsetAdd(arr, list, arr.length, 0, 0);
        Collections.sort(list);
        System.out.println(list);
    }

    public static void subsetAdd(int[] arr, ArrayList<Integer> list, int n, int i, int sum) {
        if (i == n) {
            list.add(sum);
            return;
        }

        subsetAdd(arr, list, n, i + 1, sum + arr[i]); // picking

        subsetAdd(arr, list, n, i + 1, sum);       // not picking

    }
}
```

46.Permutations



```
class Solution {
public List<List<Integer>> permute(int[] nums) {
List<Integer> list = new ArrayList<>() ;
List<List<Integer>> finalList = new ArrayList<>() ;
boolean[] freq = new boolean[nums.length] ;
Permutation(nums, finalList,list,freq ) ;
return finalList ;

}

public void Permutation(int[] nums, List<List<Integer>> finalList, List<Integer> list , boolean[] freq){
    if(list.size()==nums.length){
        finalList.add(new ArrayList<>(list)) ;
        return ;
    }

    for( int i = 0; i<nums.length;i++){
        if(!freq[i]){
            freq[i] = true ;
            list.add(nums[i]) ;
            Permutation(nums, finalList, list ,freq) ;
            list.remove(list.size()-1) ;
            freq[i] = false ;
        }
    }
}
}
```

LOVE BABBAR - Recursion

Check if the array is a Sorted array

```
public class Practice2 {
    public static void main(String[] args) {
        int[] nums = {2, 4, 6, 8, 19};
        boolean b = isSorted(nums, nums.length);
    }

    public static boolean isSorted(int[] nums, int n) {
        if (n == 0 || n == 1) {
            return true;
        } else if (nums[n-2] > nums[n-1]) {
            return false;
        } else {
            return isSorted(nums, n-1);      // checks from the back
        }
    }
}
```

Sum of the elements of the array

```
import java.util.*;

public class Practice2 {
    public static void main(String[] args) {
        int[] nums = { 2, 4, 6, 8, 90, 34, 56 };
        int sum = Sum(nums, 0, 0);
        System.out.println(sum);
    }

    public static int Sum(int[] nums, int index, int sum) {
if (nums.length == 0) {
            return 0;
        }
        if (nums.length == 1) {
            return nums[0];
        }

        if (index == nums.length) {
            return sum;
        } else {
            return Sum(nums, index + 1, sum + nums[index]);
        }
    }
}
```

Linear Search

```
import java.util.*;

public class Practice2 {
    public static void main(String[] args) {
        int[] nums = { 3, 52, 76, 12, 88 };
        int index = Find(nums, 0, 3);
        System.out.println(index);
    }

    public static int Find(int[] nums, int ind, int target) {
        if (nums.length == 0 || nums.length == 1 && target != nums[0] || ind == nums.length || mid>=nums.length)
            return -1;
        if (nums.length == 1 && target == nums[0])
            return 0;
        if (nums[ind] == target) {
            return ind;
        } else {
            return Find(nums, ind + 1, target);
        }
    }
}
```

Binary Search

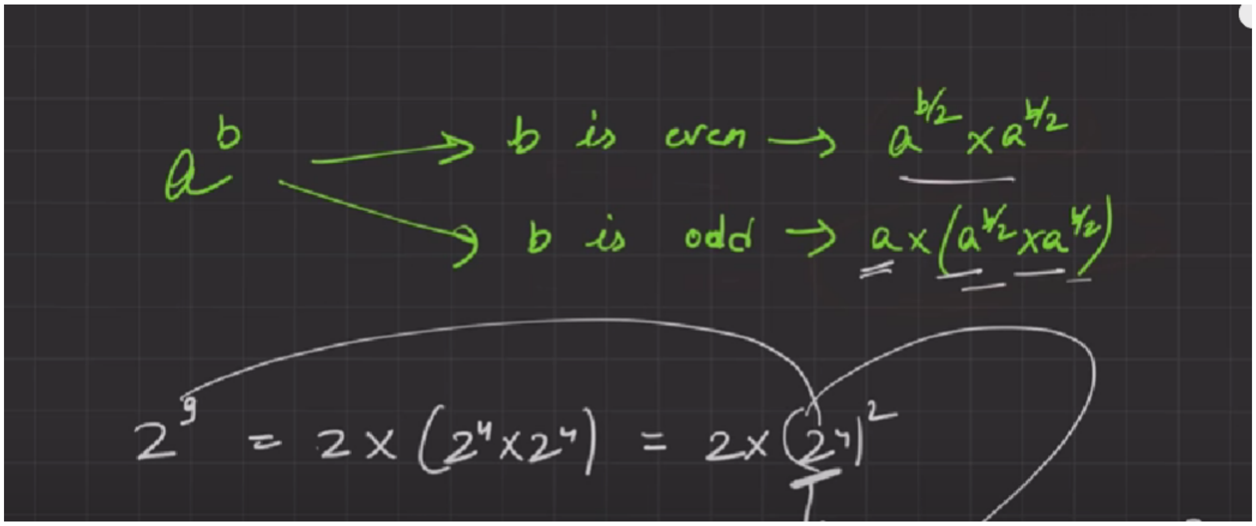
```
import java.util.*;

public class Practice2 {
    public static void main(String[] args) {
        int[] nums = { 7 };
        int index = BS(nums, 0, nums.length, 28);
        System.out.println(index);
    }

    public static int BS(int[] nums, int low, int high, int target) {
        int mid = low + (high - low) / 2;
        if (low > high || nums.length == 0 || nums.length == 1 && target != nums[0]) {
            return -1;
        }
        if (nums[mid] == target) {
            return mid;
        }
        if (nums[mid] > target) {
            return BS(nums, low, mid - 1, target);
        } else {
            return BS(nums, mid + 1, high, target);
        }
    }
}
```

Reverse a string - Approach 1

```
import java.util.*;

public class Practice2 {
    public static void main(String[] args) {
        String str = "";
        String s = Reverse(str, str.length() - 1, "");
        System.out.println(s);
    }

    public static String Reverse(String str, int index, String s) {

        if (index < 0) {
            return s;
        }

        s = str.charAt(index) + Reverse(str, index - 1, s);
        return s;
    }
}
```

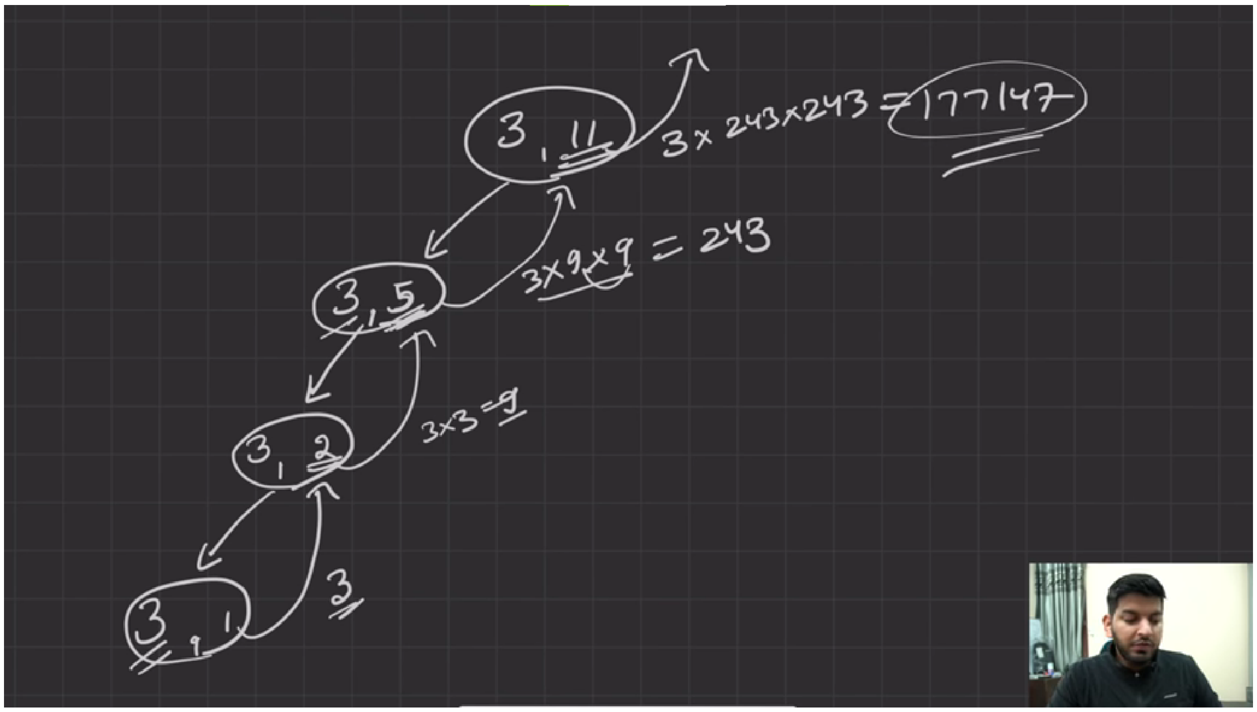
Appraoch 2 - Swap tne start and the end element

Power of a number (Optimized approach)



a^b
b is even → $a^{b/2} \times a^{b/2}$
b is odd → $a \times (a^{b/2} \times a^{b/2})$
$2^9 = 2 \times (2^4 \times 2^4) = 2 \times (2^4)^2$

Recursion tree -



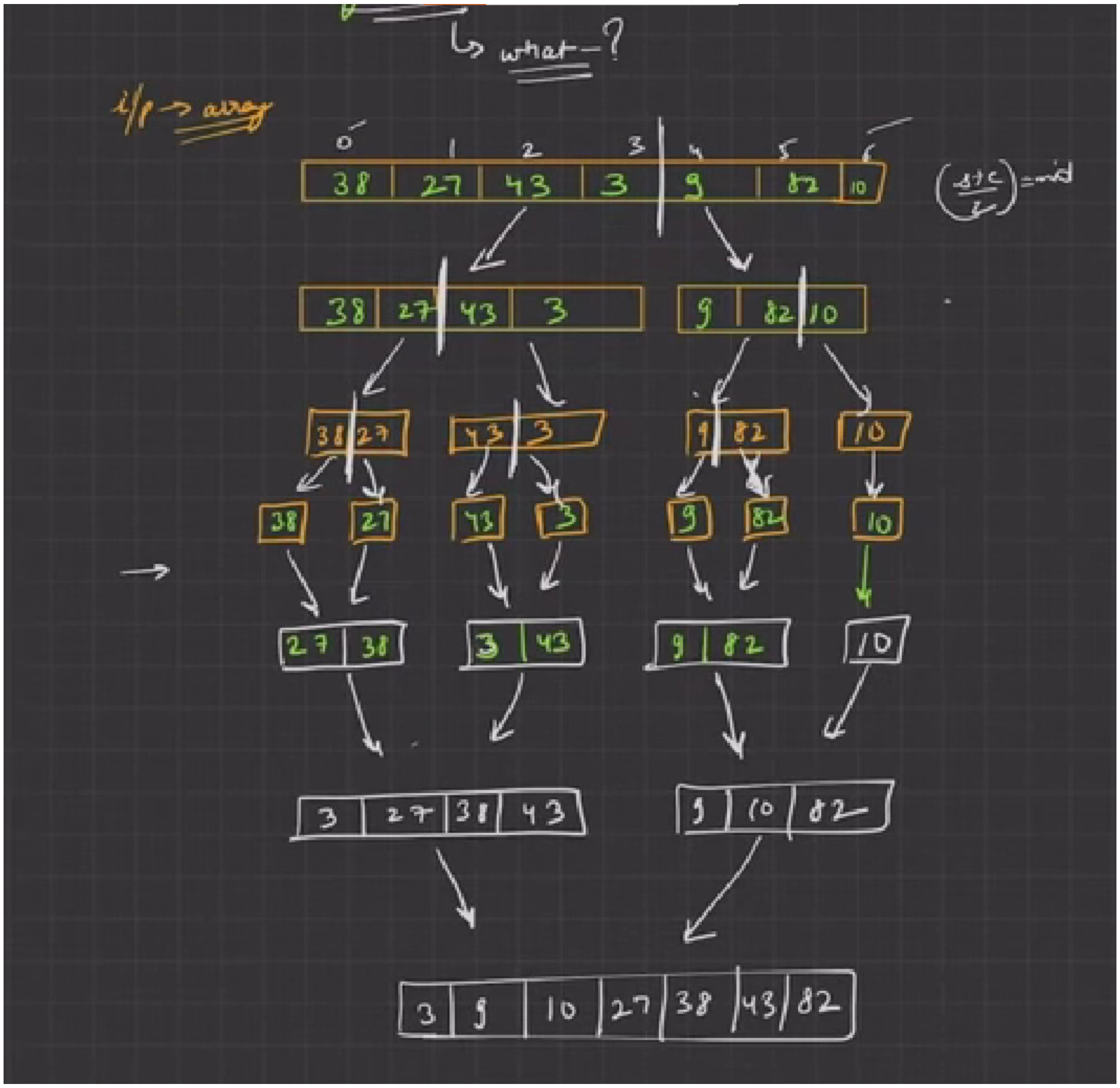
```
import java.util.*;

public class Practice2 {
    public static void main(String[] args) {
        int a = 5;
        int b = 3;
        int ans = Power(a, b);
        System.out.println(ans);

    }

    public static int Power(int a, int b) {
        if (b == 0)
            return 1;

        if (b == 1)
            return a;
        // Recurrive call
        int ans = Power(a, b / 2);

        /*
         * if b is even
         *
         * Formula is => ans = a^(b/2) * a^(b/2)
         *
         * if b is odd
         *
         * Formula is => ans = a * a^(b/2) * a^(b/2)
         */
        if (b % 2 == 0)

            return ans * ans;
        else
            return a * ans * ans;

    }
}
```

Bubble Sort

```
import java.util.*;

public class Practice2 {
    public static void main(String[] args) {
        int[] nums = { 2, 45, 33, 16, 87, 76 };
        int[] sorted = Bubblesort(nums, nums.length - 1);
        System.out.println(Arrays.toString(sorted));

    }

    public static int[] Bubblesort(int[] arr, int n) {
        if (n == 0 || n == 1) {
            return arr;
        }

        for (int i = 0; i < n; i++) {
            if (arr[i] > arr[i + 1]) {
                int temp = arr[i];
                arr[i] = arr[i + 1];
                arr[i + 1] = temp;
            }
        }

        return Bubblesort(arr, n - 1);

    }
}
```

Merge Sort - Quickest of all sorting algorithms

Merge sort *is defined as a sorting algorithm that works by dividing an array into smaller subarrays, sorting each subarray, and then merging the sorted subarrays back together to form the final sorted array.*



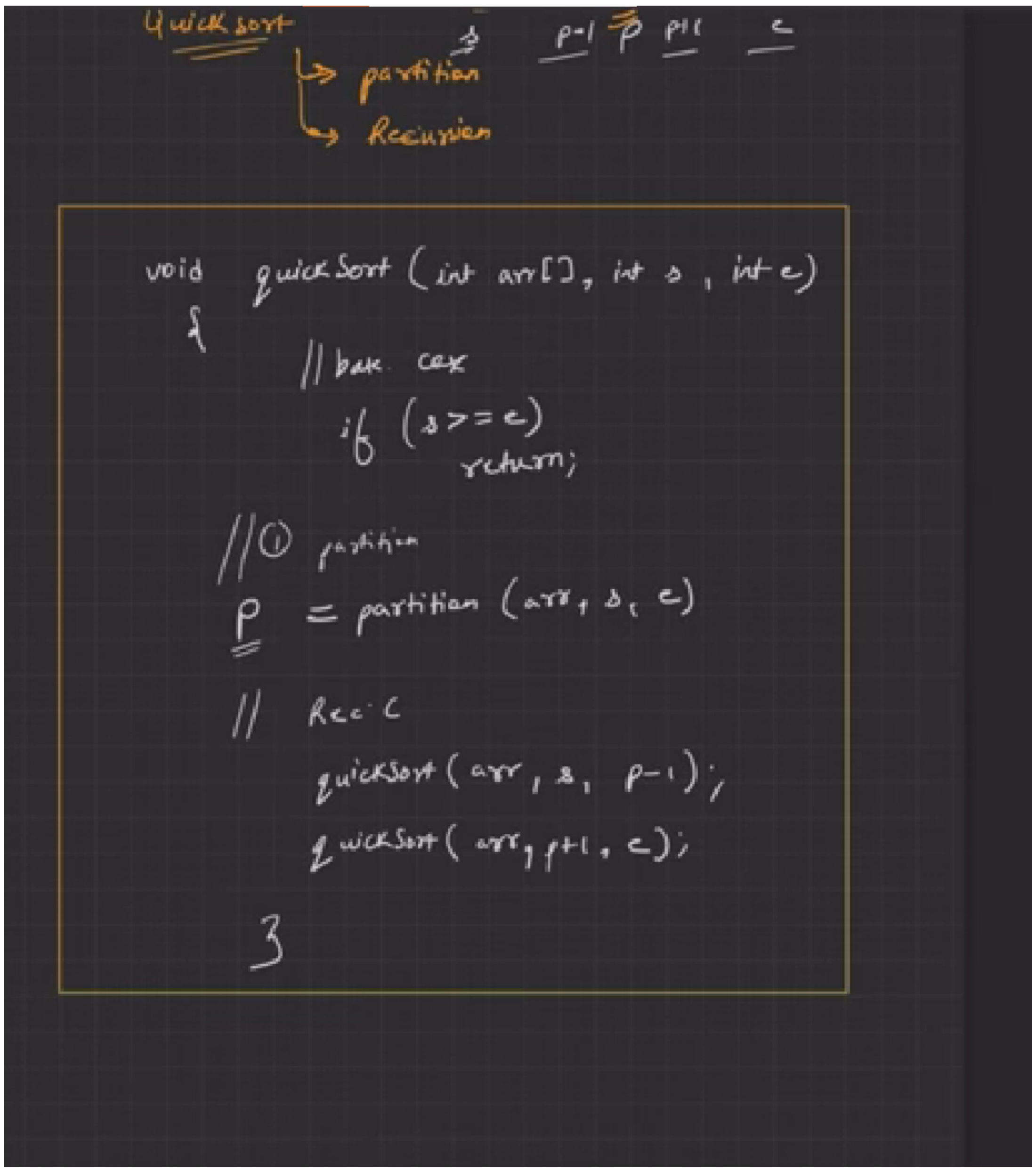
```
import java.util.*;

public class Practice2 {
    public static void main(String[] args) {
        int arr[] = { 12, 11, 13, 5, 6, 7 };

        System.out.println("Given Array");
        printArray(arr);

        sort(arr, 0, arr.length - 1);

        System.out.println("\nSorted array");
        printArray(arr);
    }

    // Merges two subarrays of arr[].
    // First subarray is arr[low..mid]
    // Second subarray is arr[mid+1..high]
    static void merge(int arr[], int low, int mid, int high) {
        // Find sizes of two subarrays to be merged
        int n1 = mid - low + 1;
        int n2 = high - mid;

        /* Create temp arrays */
        int L[] = new int[n1];
        int R[] = new int[n2];

        /* Copy data to temp arrays */
        for (int i = 0; i < n1; ++i)
            L[i] = arr[low + i];
        for (int j = 0; j < n2; ++j)
            R[j] = arr[mid + 1 + j];

        /* Merge the temp arrays */

        // Initial indexes of first and second subarrays
        int i = 0, j = 0;

        // Initial index of merged subarray array
        int k = low;
        while (i < n1 && j < n2) {
            if (L[i] <= R[j]) { // if element of left array is smaller than the element of right array, then add
                                // L[i] to the array and increment i
                arr[k] = L[i];
                i++;
            } else {
                arr[k] = R[j]; // if element of right array is smaller than the element of left array, then add
                               // R[j] to the array and increment j
                j++;
            }
            k++;
        }

        /* Copy remaining elements of L[] if any (in case the size of L[] > R[]) */
        while (i < n1) {
            arr[k] = L[i];
            i++;
            k++;
        }

        /* Copy remaining elements of R[] if any (in case the size of R[] > L[]) */
        while (j < n2) {
            arr[k] = R[j];
            j++;
            k++;
        }
    }

    // Main function that sorts arr[low..high] using
    // merge()
    static void sort(int arr[], int low, int high) {
        if (low < high) {
            // Find the middle point
            int mid = low + (high - low) / 2;

            // Sort first and second halves
            sort(arr, low, mid);
            sort(arr, mid + 1, high);

            // Merge the sorted halves
            merge(arr, low, mid, high);
        }
    }

    /* A utility function to print array of size n */
    static void printArray(int arr[]) {
        int n = arr.length;
        for (int i = 0; i < n; ++i)
            System.out.print(arr[i] + " ");
        System.out.println();
    }
```

```
}
```

Inversions in an array using Merge Sort

```java
// Java implementation of the approach
import java.util.Arrays;

public class GFG {

  // Function to count the number of inversions
  // during the merge process
  private static int mergeAndCount(int[] arr, int l,
                  int m, int r)
  {

    // Left subarray
    int[] left = Arrays.copyOfRange(arr, l, m + 1);

    // Right subarray
    int[] right = Arrays.copyOfRange(arr, m + 1, r + 1);

    int i = 0, j = 0, k = l, swaps = 0;

    while (i < left.length && j < right.length) {
      if (left[i] <= right[j])
        arr[k++] = left[i++];
      else {
        arr[k++] = right[j++];
        swaps += (m + 1) - (l + i);
      }
    }
    while (i < left.length)
      arr[k++] = left[i++];
    while (j < right.length)
      arr[k++] = right[j++];
    return swaps;
  }

  // Merge sort function
  private static int mergeSortAndCount(int[] arr, int l,
                    int r)
  {

    // Keeps track of the inversion count at a
    // particular node of the recursion tree
    int count = 0;

    if (l < r) {
      int m = (l + r) / 2;

      // Total inversion count = left subarray count
      // + right subarray count + merge count

      // Left subarray count
      count += mergeSortAndCount(arr, l, m);

      // Right subarray count
      count += mergeSortAndCount(arr, m + 1, r);

      // Merge count
      count += mergeAndCount(arr, l, m, r);
    }

    return count;
  }

  // Driver code
  public static void main(String[] args)
  {
    int[] arr = { 1, 20, 6, 4, 5 };

    System.out.println(
      mergeSortAndCount(arr, 0, arr.length - 1));
  }
}

// This code is contributed by Pradip Basak
```

Quick Sort using Recursion



```
void quickSort (int arr[], int s , int e)
{
    // base case
    if (s >= e)
        return;

    // ① partition
    p = partition (arr, s, e)

    // Rec C
    quickSort (arr, s, p-1);
    quickSort (arr, p+1, e);

}
```

```java
import java.util.*;

public class Practice2 {
    public static void main(String[] args) {
        int arr[] = { 12, 11, 13, 5, 6, 7, 1, 67, 45, 5 };

        Quicksort(arr, 0, arr.length - 1);
        System.out.println(Arrays.toString(arr));

    }

    static void Quicksort(int[] arr, int s, int e) {
        if (s > e) {
            return;
        }

        int p = partition(arr, s, e); // find the pivot index of the starting element

        Quicksort(arr, s, p - 1); // Sort the left side of the array
        Quicksort(arr, p + 1, e); // sort the right side of the array

    }

    static int partition(int[] arr, int s, int e) {
        int pivot = arr[s]; // pivot element to put in place
        int count = 0;

        for (int i = s + 1; i <= e; i++) { // no of elements smaller than the pivot on its right side
            if (arr[i] <= pivot) {
                count++;
            }
        }

        // the correct pivotindex is its original index plus the count of number of
        // elements smaler than it
        int pivotindex = s + count;
        swap(arr, pivotindex, s); // Swap the element at pivot index with s to assign correct index to the pivot
                                  // element

        int i = s, j = e;
        // checking from the front and the back

        // checking if the elements on the left of the pivot are smaller than it and the
        // elements on the right are larger than it
        while (i < pivotindex && j > pivotindex) {
            while (arr[i] <= pivot) { // (on left) if smaller, check the next element
                i++;
            }

            while (arr[j] >= pivot) { // (on right) , if larger ,check for the previous elements
                j--;
            }

            // in case an element violates the rule of being smaller than the pivot on the
            // left side or being greater than the pivot on the right side
            if (i < pivotindex && j > pivotindex) {
                swap(arr, i, j); // swap the elements with each other
                i++;
                j--;
            }
        }
        return pivotindex; // return the right index of the pivot

    }

    static int[] swap(int[] arr, int i, int j) {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;

        return arr;
    }
```