**Search ing:**

**Sequential/Liner Search =>(Apply on unsorted Array) => using for loop**

*// Search Item with integer aray*

void LinerSearch(){

int arr[] = {3, 1, 11, 10, 13, 4, 50, 5};

int findItem = 10;

for(int i=0; i < arr.length; i++){

if(arr[i] == findItem){

System.*out*.println("Item Index is:"+i +" and item is "+findItem);

}

}

}

*//****Liner Search*** *Item with String aray*

void LinerSearch(){

String arr[] = {"Ram", "Shayam", "Geeta", "Sheeta"};

String findItem = "Geeta";

for(int i=0; i < arr.length; i++){

if(arr[i].equals(findItem)){

System.*out*.println("Item Index is:"+i +" and item is "+findItem);

}

}

}

*//****Liner Search*** *Item with String aray*

*//The* ***compareTo()*** *method compares two strings* ***lexicographically****. ... The method returns 0 if the string is equal to the other string. A value less than 0 is returned if the string is less than the other string (less characters) and a value greater than 0 if the string is greater than the other string (more characters).*

void LinerSearch(){

String arr[] = {“Abhi”,"Ram","Shayam","Geeta", “Shatish”};

String findItem = "Geeta";

for(int i=0; i < arr.length; i++){

System.*out*.println("Item Index is:" + i + " and item is " + arr[i].compareTo(findItem));

if(arr[i].compareTo(findItem) == 0) {

System.*out*.println("Item Index is:" + i + " and item is " + findItem);

}

}

}

**Search ing: Binary Search: it’s**

Binary search is a fast search algorithm with run-time complexity of Ο(log n) for sorted item collection. This search algorithm works on the principle of divide and conquer and

*// Binary Search Item with int aray*

void BinarySearch(){

int[] arr = {2,7, 11, 16,49, 55};

int findItem = 49;

int firstIndex = 0;

int lastIndex= arr.length -1;

int midIndex = (firstIndex + lastIndex) / 2;

while (firstIndex < lastIndex){

if(findItem == arr[midIndex]){

System.*out*.println("Find Item:"+findItem +" and index:"+ midIndex);

break;

}else if(findItem < arr[midIndex] ){

lastIndex = midIndex -1;

}else{

firstIndex = midIndex +1;

}

midIndex = (firstIndex + lastIndex) / 2;

}

}

*// Binary Search Item with int aray*

void BinarySearch(){

String[] arr = {"Abhi","Geeta", "Hitesh", "Indra", "Ram", "Shayam","Sheeta"};

String findItem = "Ram";

int firstIndex = 0;

int lastIndex= arr.length -1;

int midIndex = (firstIndex + lastIndex) / 2;

while (firstIndex <= lastIndex){

int comPareVal = findItem.compareTo(arr[midIndex]);

System.*out*.println("Item:"+lastIndex +" comPareVal:"+comPareVal);

if(comPareVal == 0){

System.*out*.println("Find Item:"+findItem +" and index:"+ midIndex);

break;

}else if(comPareVal < 0){

lastIndex = midIndex -1;

System.*out*.println("Item not found and less "+lastIndex);

}else{

firstIndex = midIndex +1;

System.*out*.println("Item not found and gretter "+lastIndex);

}

midIndex = (firstIndex + lastIndex) / 2;

}

}

**Sorting algorithm**

*//In* ***selection sort algorithm****, we search for the lowest element and arrange it to the proper location. We swap the current element with the next lowest number and* time complexity is O(n2).

void SalectionSort(){

int arr[] = {2, 50, 1, 20, 4, 3};

System.*out*.println("Before Sorting: "+ Arrays.*toString*(arr));

for(int i=0; i<arr.length; i++){

int minItemIndex = i;

for(int j = i+1; j<arr.length; j++){

if(arr[j] < arr[minItemIndex]){

minItemIndex = j;

}

}

int temp = arr[minItemIndex];

arr[minItemIndex] = arr[i];

arr[i] = temp;

System.*out*.println("In Sorting: "+ Arrays.*toString*(arr));

}

System.*out*.println("After Sorting: "+ Arrays.*toString*(arr));

*/\**

*Before Sorting: [2, 50, 1, 20, 4, 3]*

*In Sorting: [1, 50, 2, 20, 4, 3]*

*In Sorting: [1, 2, 50, 20, 4, 3]*

*In Sorting: [1, 2, 3, 20, 4, 50]*

*In Sorting: [1, 2, 3, 4, 20, 50]*

*In Sorting: [1, 2, 3, 4, 20, 50]*

*In Sorting: [1, 2, 3, 4, 20, 50]*

*After Sorting: [1, 2, 3, 4, 20, 50]*

*\*/*

}

*//***Bubble Sort:** *Bubble sort works on the repeatedly swapping of adjacent elements until they are not in the intended order.* time complexity is O(n).

void BubbleSort(){

int arr[] = {2, 1, 4, 3, 0, -1};

for(int i=0; i<arr.length; i++){

for(int j = i+1; j<arr.length; j++){

if(arr[j] < arr[i]){

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

System.*out*.println("Inner In Sorting: "+ Arrays.*toString*(arr));

}

System.*out*.println("Inner Out Sorting: "+ Arrays.*toString*(arr));

}

System.*out*.println("After Sorting: "+ Arrays.*toString*(arr));

*/\**

*Before Sorting: [2, 1, 4, 3, 0, -1]*

*Inner In Sorting: [1, 2, 4, 3, 0, -1]*

*Inner In Sorting: [1, 2, 4, 3, 0, -1]*

*Inner In Sorting: [1, 2, 4, 3, 0, -1]*

*Inner In Sorting: [0, 2, 4, 3, 1, -1]*

*Inner In Sorting: [-1, 2, 4, 3, 1, 0]*

*Inner Out Sorting: [-1, 2, 4, 3, 1, 0]*

*Inner In Sorting: [-1, 2, 4, 3, 1, 0]*

*Inner In Sorting: [-1, 2, 4, 3, 1, 0]*

*Inner In Sorting: [-1, 1, 4, 3, 2, 0]*

*Inner In Sorting: [-1, 0, 4, 3, 2, 1]*

*Inner Out Sorting: [-1, 0, 4, 3, 2, 1]*

*Inner In Sorting: [-1, 0, 3, 4, 2, 1]*

*Inner In Sorting: [-1, 0, 2, 4, 3, 1]*

*Inner In Sorting: [-1, 0, 1, 4, 3, 2]*

*Inner Out Sorting: [-1, 0, 1, 4, 3, 2]*

*Inner In Sorting: [-1, 0, 1, 3, 4, 2]*

*Inner In Sorting: [-1, 0, 1, 2, 4, 3]*

*Inner Out Sorting: [-1, 0, 1, 2, 4, 3]*

*Inner In Sorting: [-1, 0, 1, 2, 3, 4]*

*Inner Out Sorting: [-1, 0, 1, 2, 3, 4]*

*Inner Out Sorting: [-1, 0, 1, 2, 3, 4]*

*After Sorting: [-1, 0, 1, 2, 3, 4]*

*\*/*

}

**//Marge Sort: The merge sort algorithm is based on the principle of divide and conquer algorithm where a problem is divided into multiple sub-problems. Each sub-problem is solved individually and finally, sub-problems are combined to form the final solutions. time complexity is O(n).**

void margeSort(){

int[] array = {11, 7, 3, 2, 6, 5, 12, 10, 9, 1,0 };

divideIntoSubArr(array, 0, array.length - 1);

}

void divideIntoSubArr(int[] arr, int leftIndex, int rightIndex) {

if(leftIndex < rightIndex){

*// midIndex is the point where the array is divided into two sub arrays*

int midIndex = (leftIndex + rightIndex) / 2;

*// recursive call to each sub arrays*

divideIntoSubArr(arr, leftIndex, midIndex);

divideIntoSubArr(arr, midIndex + 1, rightIndex);

*// Merge the sorted sub arrays*

mareSortedArr(arr, leftIndex, midIndex, rightIndex);

}

}

void mareSortedArr(int[] array, int leftIndex, int midIndex, int ightIndex) {

int LAS = midIndex - leftIndex + 1;

int RAS = rightIndex - midIndex;

int LA[] = new int[LAS];

int RA[] = new int[RAS];

for (int i = 0; i < LAS; i++){

LA[i] = array[leftIndex + i];

}

for (int j=0; j<RAS; j++){

RA[j] = array[midIndex + 1 + j];

}

*// Maintain current index of sub-arrays and main array*

*// Until we reach either end of either L or M, pick larger among*

*// elements L and M and place them in the correct position at A[p..r]*

*// for sorting in descending*

*// use if(L[i] >= <[j])*

int i, j, k;

i = 0;

j = 0;

k = leftIndex;

while (i < LAS && j < RAS){

if( LA[i] <= RA[j] ){

array[k] = LA[i];

i++;

}else{

array[k] = RA[j];

j++;

}

k++;

}

*// When we run out of elements in either L or M,*

*// pick up the remaining elements and put in A[p..r]*

while ( i < LAS){

array[k] = LA[i];

i++;

k++;

}

while ( j < RAS){

array[k] = RA[j];

j++;

k++;

}

System.*out*.println("Sorted Arrr:"+Arrays.*toString*(array));

}

*/\*Before Sorting: = [11, 7, 3, 2, 6, 5, 12, 10, 9, 1,0 ];*

*Sorted Arrr:[7, 11, 3, 2, 6, 5, 12, 10, 9, 1, 0]*

*Sorted Arrr:[3, 7, 11, 2, 6, 5, 12, 10, 9, 1, 0]*

*Sorted Arrr:[3, 7, 11, 2, 6, 5, 12, 10, 9, 1, 0]*

*Sorted Arrr:[3, 7, 11, 2, 5, 6, 12, 10, 9, 1, 0]*

*Sorted Arrr:[2, 3, 5, 6, 7, 11, 12, 10, 9, 1, 0]*

*Sorted Arrr:[2, 3, 5, 6, 7, 11, 10, 12, 9, 1, 0]*

*Sorted Arrr:[2, 3, 5, 6, 7, 11, 9, 10, 12, 1, 0]*

*Sorted Arrr:[2, 3, 5, 6, 7, 11, 9, 10, 12, 0, 1]*

*Sorted Arrr:[2, 3, 5, 6, 7, 11, 0, 1, 9, 10, 12]*

*Sorted Arrr:[0, 1, 2, 3, 5, 6, 7, 9, 10, 11, 12]*

*\*/*

Find **Duplicate items in Array & ArrayList**

List<String> aList = new ArrayList<>();

aList.add("Abhishek");

aList.add("Chandan");

aList.add("Rai");

aList.add("Abhishek");

aList.add("Rudra");

aList.add("Abhishek");

aList.add("Rudra");

aList.add("Abhishek");

String[] arr = {"Chandan", "Rudra", "Rai", "Rudra", "Rudra", "Rai"};

*// List<String> list = new ArrayList<String >(Arrays.asList(arr));*

*// Set<String> uniqList = new HashSet<>(aList);*

*// Set<String> uniqLis = new HashSet<>(Arrays.asList(arr));*

aList.addAll(Arrays.*asList*(arr));

*//****Find duplicate*** *items Using second ArrayList & check items is contain or not*

List<String> uniqArrayList = new ArrayList<>();

for (String item: aList) {

if(uniqArrayList.contains(item)){

System.*out*.println("duplicate Items found in AL:"+item);

}else{

uniqArrayList.add(item);

}

}

*/\* duplicate Items found in AL:Abhishek*

*duplicate Items found in AL:Abhishek*

*duplicate Items found in AL:Rudra*

*duplicate Items found in AL:Abhishek*

*\*/*

*//****Remove duplicate items*** *|| get only unque item Using HashSet and add items at time of inialization or addAll methos*

Set<String> uniqSet = new HashSet<>(aList);

System.*out*.println("Unique items List :"+uniqSet);

*//uniqSet.addAll(aList);*

*/\**

*Unique items List :[Abhishek, Rudra, Rai, Chandan]*

*\*/*

*//****Remove duplicate items*** *Using others LinkedHashSet & check items add or not in list*

Set<String> hashSet = new HashSet<>();

for (String item: aList) {

boolean isAdd = hashSet.add(item);

if(!isAdd){

System.*out*.println("Duplicate Items found using LHS :"+item);

}

}

*/\* Duplicate Items found using LHS :Abhishek*

*Duplicate Items found using LHS :Abhishek*

*Duplicate Items found using LHS :Rudra*

*Duplicate Items found using LHS :Abhishek*

*\*/*

*//****find & Count the duplicate*** *items Using HashMap & check items is contain(if contain count++) or if not(count 1)*

HashMap<String,Integer> hashMap = new HashMap<>();

for(String i: aList) {

if (hashMap.containsKey(i)) {

hashMap.put(i, hashMap.get(i) + 1);

}else {

hashMap.put(i, 1);

}

}

System.*out*.println("HashMap with Duplicate count:" + hashMap);

*/\**

*HashMap with Duplicate count:{Abhishek=4, Rudra=2, Rai=1, Chandan=1}*

*\*/*

*//find & Count the duplicate items Using HashMap & check items is contain(if contain count++) or if not(count 1)*

*// Print it in custome tag*

HashMap<String,Integer> hashMap1 = new HashMap<>();

for(String i: aList) {

if (hashMap1.containsKey(i)) {

hashMap1.put(i, hashMap1.get(i) + 1);

}else {

hashMap1.put(i, 1);

}

}

//for(Map.Entry<String, Integer> mapx :hashMap.entrySet()){

for(HashMap.Entry<String, Integer> mapx :hashMap.entrySet()){

String key = mapx.getKey();

Integer count = mapx.getValue();

System.*out*.println("HashMap with Duplicate Items:"+key + " count:"+count);

}

*/\* HashMap with Duplicate Items:Abhishek count:4*

*HashMap with Duplicate Items:Rudra count:2*

*HashMap with Duplicate Items:Rai count:1*

*HashMap with Duplicate Items:Chandan count:1*

*\*/*