# LAB # 05

**LAB TASKS 1:**

import java.util.Scanner;

public class SelectionSortExample {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Get the number of elements

System.out.print("Enter the number of elements in the array: ");

int n = scanner.nextInt();

// Initialize the array

int[] arr = new int[n];

// Get the elements of the array

System.out.println("Enter the elements of the array:");

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

arr[i] = scanner.nextInt();

}

// Perform Selection Sort

selectionSort(arr);

scanner.close();

}

public static void selectionSort(int[] arr) {

int n = arr.length;

// Selection Sort algorithm

for (int i = 0; i < n - 1; i++) {

int minIndex = i;

// Find the minimum element in the unsorted portion of the array

for (int j = i + 1; j < n; j++) {

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

minIndex = j;

}

}

// Swap the found minimum element with the first element of the unsorted portion

int temp = arr[minIndex];

arr[minIndex] = arr[i];

arr[i] = temp;

// Print the array values with their locations after each pass

System.out.println("After pass " + (i + 1) + ":");

for (int k = 0; k < n; k++) {

System.out.print(arr[k] + "(" + k + ") ");

}

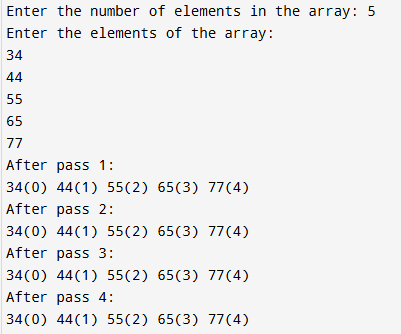
System.out.println();

}

}

}

**OUTPUT**



**LAB TASKS 2:**

import java.util.Scanner;

public class BubbleSortExample {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Array size of 10

int[] arr = new int[10];

// Input: Take 10 numbers

System.out.println("Enter 10 numbers:");

for (int i = 0; i < 10; i++) {

arr[i] = scanner.nextInt();

}

// Perform Bubble Sort and print each iteration

bubbleSort(arr);

scanner.close();

}

public static void bubbleSort(int[] arr) {

int n = arr.length;

// Bubble Sort algorithm

for (int i = 0; i < n - 1; i++) {

boolean swapped = false;

// Traverse the array from the beginning to the (n-i-1)th element

for (int j = 0; j < n - 1 - i; j++) {

// Swap if the current element is greater than the next element

if (arr[j] > arr[j + 1]) {

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

swapped = true;

}

}

// Print the array after each iteration

System.out.println("After pass " + (i + 1) + ":");

for (int k = 0; k < n; k++) {

System.out.print(arr[k] + " ");

}

System.out.println();

// If no elements were swapped in this pass, the array is sorted

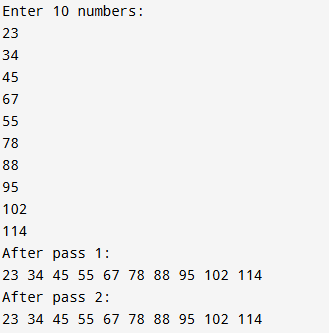
if (!swapped) break;

}

}

}

**OUTPUT**



**LAB TASKS 3:**

import java.util.Arrays;

import java.util.Random;

public class MergeSortExample {

public static void main(String[] args) {

// Initialize array with 10 random numbers

int[] arr = new int[10];

Random rand = new Random();

System.out.println("Original Array:");

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

arr[i] = rand.nextInt(100); // Random numbers between 0 and 99

System.out.print(arr[i] + " ");

}

System.out.println();

// Perform Merge Sort

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

}

// Recursive merge sort function

public static void mergeSort(int[] arr, int left, int right) {

if (left < right) {

// Find the middle point

int mid = (left + right) / 2;

// Recursively sort first and second halves

mergeSort(arr, left, mid);

mergeSort(arr, mid + 1, right);

// Merge the sorted halves

merge(arr, left, mid, right);

// Print the array after each merge

System.out.println("After merging from index " + left + " to " + right + ": " + Arrays.toString(arr));

}

}

// Merge function to combine two halves

public static void merge(int[] arr, int left, int mid, int right) {

// Sizes of two subarrays to be merged

int n1 = mid - left + 1;

int n2 = right - mid;

// Temporary arrays

int[] leftArray = new int[n1];

int[] rightArray = new int[n2];

// Copy data to temporary arrays

for (int i = 0; i < n1; i++) {

leftArray[i] = arr[left + i];

}

for (int j = 0; j < n2; j++) {

rightArray[j] = arr[mid + 1 + j];

}

// Merge the temporary arrays back into arr[left..right]

int i = 0, j = 0;

int k = left;

while (i < n1 && j < n2) {

if (leftArray[i] <= rightArray[j]) {

arr[k] = leftArray[i];

i++;

} else {

arr[k] = rightArray[j];

j++;

}

k++;

}

// Copy remaining elements of leftArray, if any

while (i < n1) {

arr[k] = leftArray[i];

i++;

k++;

}

// Copy remaining elements of rightArray, if any

while (j < n2) {

arr[k] = rightArray[j];

j++;

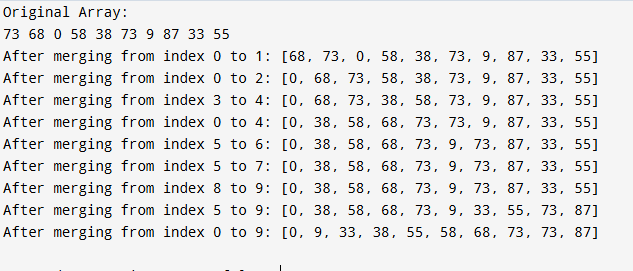
k++;

}

}

}

**OUTPUT**



**HOME TASK 1:**

import java.util.Arrays;

import java.util.Comparator;

import java.util.Random;

class Account {

int accountNumber;

int balance;

Account(int accountNumber, int balance) {

this.accountNumber = accountNumber;

this.balance = balance;

}

@Override

public String toString() {

return "Account No. " + accountNumber + " Balance " + balance;

}

}

public class AccountSorter {

public static void main(String[] args) {

int n = 10; // Change this to any size you want for testing

Account[] accounts = new Account[n];

Random random = new Random();

// Initialize the accounts with random account numbers and balances

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

int accountNumber = 1000 + random.nextInt(9000); // Account numbers between 1000 and 9999

int balance = random.nextInt(100001); // Balances between 0 and 100000

accounts[i] = new Account(accountNumber, balance);

}

// Display accounts before sorting

System.out.println("Accounts before sorting:");

for (Account account : accounts) {

System.out.println(account);

}

// Sort the accounts by balance in descending order using Arrays.sort with a custom comparator

Arrays.sort(accounts, new Comparator<Account>() {

@Override

public int compare(Account a1, Account a2) {

return Integer.compare(a2.balance, a1.balance); // For descending order

}

});

// Display accounts after sorting

System.out.println("\nAccounts after sorting by highest balance:");

for (Account account : accounts) {

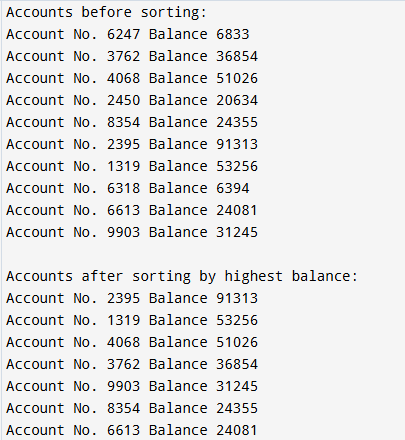
System.out.println(account);

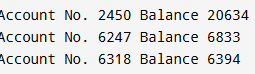
}

}

}

**OUTPUT**





**HOME TASK 2:**

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

public class GenericMergeSort {

public static void main(String[] args) {

// Example with integers

List<Integer> intList = Arrays.asList(5, 3, 8, 6, 2, 7, 4, 1);

System.out.println("Original integer list: " + intList);

List<Integer> sortedIntList = mergeSort(intList);

System.out.println("Sorted integer list: " + sortedIntList);

// Example with strings

List<String> strList = Arrays.asList("banana", "apple", "cherry", "date");

System.out.println("\nOriginal string list: " + strList);

List<String> sortedStrList = mergeSort(strList);

System.out.println("Sorted string list: " + sortedStrList);

}

// Generic merge sort method

public static <T extends Comparable<T>> List<T> mergeSort(List<T> list) {

if (list.size() <= 1) {

return list;

}

int mid = list.size() / 2;

List<T> left = new ArrayList<>(list.subList(0, mid));

List<T> right = new ArrayList<>(list.subList(mid, list.size()));

return merge(mergeSort(left), mergeSort(right));

}

// Merges two sorted lists into a single sorted list

private static <T extends Comparable<T>> List<T> merge(List<T> left, List<T> right) {

List<T> merged = new ArrayList<>();

int leftIndex = 0, rightIndex = 0;

while (leftIndex < left.size() && rightIndex < right.size()) {

if (left.get(leftIndex).compareTo(right.get(rightIndex)) <= 0) {

merged.add(left.get(leftIndex));

leftIndex++;

} else {

merged.add(right.get(rightIndex));

rightIndex++;

}

}

// Add remaining elements

while (leftIndex < left.size()) {

merged.add(left.get(leftIndex));

leftIndex++;

}

while (rightIndex < right.size()) {

merged.add(right.get(rightIndex));

rightIndex++;

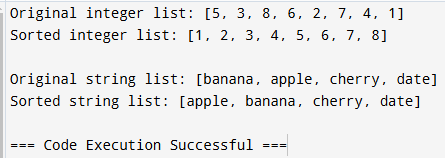
}

return merged;

}

}

**OUTPUT**



**HOME TASK 3:**

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

import java.util.Scanner;

public class SimpleMergeSort {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Read input from the user

System.out.print("Enter elements (space-separated): ");

String input = scanner.nextLine();

String[] elements = input.split(" ");

// Sort and display the result

List<String> list = Arrays.asList(elements);

list = mergeSort(list);

System.out.println("Sorted list: " + list);

scanner.close();

}

// Merge Sort method that sorts strings (can be used for integers or strings)

public static List<String> mergeSort(List<String> list) {

if (list.size() <= 1) return list;

int mid = list.size() / 2;

List<String> left = mergeSort(list.subList(0, mid));

List<String> right = mergeSort(list.subList(mid, list.size()));

return merge(left, right);

}

// Merge method to combine sorted halves

private static List<String> merge(List<String> left, List<String> right) {

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

int i = 0, j = 0;

while (i < left.size() && j < right.size()) {

if (left.get(i).compareTo(right.get(j)) <= 0) {

result.add(left.get(i));

i++;

} else {

result.add(right.get(j));

j++;

}

}

// Add remaining elements

result.addAll(left.subList(i, left.size()));

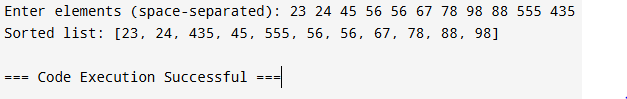
result.addAll(right.subList(j, right.size()));

return result;

}

}

**OUTPUT**



**HOME TASK 4:**

import java.util.Arrays;

import java.util.Random;

public class BankAccountSorter {

public static void main(String[] args) {

int n = 10; // Number of accounts

int[] balances = new int[n];

Random random = new Random();

// Initialize each balance with a random value between 0 and 100,000

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

balances[i] = random.nextInt(100001); // 100001 to include 100,000 as max

}

// Sort balances in descending order using Quick Sort

quickSort(balances, 0, n - 1);

// Print sorted list of balances

System.out.println("Sorted account balances (descending):");

for (int balance : balances) {

System.out.println(balance);

}

}

// Quick Sort function

public static void quickSort(int[] arr, int low, int high) {

if (low < high) {

int pivotIndex = partition(arr, low, high);

quickSort(arr, low, pivotIndex - 1);

quickSort(arr, pivotIndex + 1, high);

}

}

// Partition function for descending order

public static int partition(int[] arr, int low, int high) {

int pivot = arr[high];

int i = low - 1;

for (int j = low; j < high; j++) {

if (arr[j] > pivot) { // Change to ">" for descending order

i++;

// Swap arr[i] and arr[j]

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

// Swap arr[i + 1] and pivot (arr[high])

int temp = arr[i + 1];

arr[i + 1] = arr[high];

arr[high] = temp;

return i + 1;

}

}

**OUTPUT**

