1. **Bubble Sort**

class Bsort{

public static void bubblesort(int[] arr){

int n = arr.length;

for(int i=0;i<n;i++){ //no. of rounds

//we take -i because in next round we can't compare last element

for(int j=0;j<n-1-i;j++){ //compare adjecent elements

if(arr[j]>arr[j+1]){ //to swap elements

int temp = arr[j];

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

arr[j+1]=temp;

}

}

}

}

public static void main(String[] args) {

int[] array = {36,19,29,12,5};

bubblesort(array);

for(int num : array){

System.out.print(num+" ");

}

}

}

**============================================================================**

1. **Quick Sort**

class QuickSort {

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

//this function returns pivot element index

int pivot = arr[high]; //take last element as pivot element

int i = low-1 ; //starting no element

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

if(arr[j]<pivot){ //to insert small elements than pivot

i++;

int temp = arr[i]; //swapping

arr[i] = arr[j];

arr[j] =temp;

}

}

//pivot ke liye khali jagah tayaar karenge

int temp = arr[i+1]; //swapping

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

arr[high] = temp;

return i + 1; //return pivot index

}

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

if(low<high){

int pivot\_index=partition(arr,low,high);

quicksort(arr,low,pivot\_index-1); //small elements than pivot at left

quicksort(arr,pivot\_index+1,high);//large elements than pivot at right

}

}

public static void main(String[] args) {

int[] arr = {6,3,9,5,2,8};

int n = arr.length;

quicksort(arr,0,n-1);

for(int num: arr){

System.out.print(num+" ");

}

}

}

**==========================================================================**

**3. Selection Sort**

class Ssort{

public static void selectionsort(int[] arr){

int n =arr.length;

int min;

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

min=i;

for(int j=i+1;j<n;j++){ //start with second element to compare

if(arr[j] < arr[min]){ //condition true then put min value in j

min=j;

}

}

int temp = arr[i]; //swap

arr[i]= arr[min];

arr[min] = temp;

}

}

public static void main(String[] args) {

int[] array = {36,19,29,12,5};

selectionsort(array);

for(int num : array){

System.out.print(num+" ");

}

}

}

**================================================================================**

**4. Insertion Sort**

class InsertionSort {

public static void insertionSort(int[] arr) {

int n = arr.length;

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

int temp = arr[i];

int j;

for (j = i; j > 0 && arr[j - 1] > temp; j--) {

arr[j] = arr[j - 1];

}

arr[j] = temp;

}

}

public static void main(String[] args) {

int[] array = {36, 19, 29, 12, 5};

insertionSort(array);

for (int num : array) {

System.out.print(num + " ");

}

}

}

**==============================================================================**

**5. Merge Sort**

public class MergeSort {

public static void mergeSort(int[] array) {

//Find the middle index

int mid = array.length / 2;

//Create left and right subarrays

int[] left = new int[mid];

int[] right = new int[array.length - mid];

//Fill the left and right subarrays

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

left[i] = array[i];

}

for (int i = mid; i < array.length; i++) {

right[i - mid] = array[i];

}

//Recursively sort the left and right subarrays

mergeSort(left);

mergeSort(right);

//Merge the sorted subarrays

merge(left, right, array);

}

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

int leftIndex = 0, rightIndex = 0, resultIndex = 0;

//Compare elements from left and right subarrays and merge them into the result array

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

if (left[leftIndex] <= right[rightIndex]) {

result[resultIndex++] = left[leftIndex++];

} else {

result[resultIndex++] = right[rightIndex++];

}

}

//Copy remaining elements from left subarray, if any

while (leftIndex < left.length) {

result[resultIndex++] = left[leftIndex++];

}

//Copy remaining elements from right subarray, if any

while (rightIndex < right.length) {

result[resultIndex++] = right[rightIndex++];

}

}

public static void main(String[] args) {

int[] array = {12, 11, 13, 5, 6, 7};

MergeSort.mergeSort(array);

System.out.println("\nSorted array:");

for (int num : array) {

System.out.print(num + " ");

}

}

}

**================================================================================**

**6. Quick Sort**

class QuickSort {

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

//this function returns pivot element index

int pivot = arr[high]; //take last element as pivot element

int i = low-1 ; //starting no element

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

if(arr[j]<pivot){ //to insert small elements than pivot

i++;

int temp = arr[i]; //swapping

arr[i] = arr[j];

arr[j] =temp;

}

}

//pivot ke liye khali jagah tayaar karenge

int temp = arr[i+1]; //swapping

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

arr[high] = temp;

return i + 1; //return pivot index

}

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

if(low<high){

int pivot\_index=partition(arr,low,high);

quicksort(arr,low,pivot\_index-1); //small elements than pivot at left

quicksort(arr,pivot\_index+1,high);//large elements than pivot at right

}

}

public static void main(String[] args) {

int[] arr = {6,3,9,5,2,8};

int n = arr.length;

quicksort(arr,0,n-1);

for(int num: arr){

System.out.print(num+" ");

}

}

}

**=====================================================================================**

**7. Sorting Strings using Bubble Sort**

public class Bubble\_Sort\_Strings {

// Bubble Sort algorithm for sorting an array of strings

public static void bubble\_Sort(String[] arr) {

int n = arr.length;

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

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

if (arr[j].compareTo(arr[j + 1]) > 0) {

// Swap arr[j] and arr[j+1]

String temp = arr[j];

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

arr[j + 1] = temp;

}

}

}

}

public static void main(String[] args) {

// Sample array of strings

String[] arr = {"swaraj","abhi","omkar","raj"};

// Sort the array using Bubble Sort

bubble\_Sort(arr);

for (String str : arr) {

System.out.print(str + " ");

}

System.out.println();

}

}

**=================================================================================**

9. **Bubble Sort for Linked List by Swapping nodes**

class LinkedList {

Node head;

// Node class to create nodes

class Node {

int data;

Node next;

Node(int d) {

data = d;

next = null;

}

}

// Function to add a node at the end of the linked list

public void append(int new\_data) {

Node new\_node = new Node(new\_data);

if (head == null) {

head = new\_node;

return;

}

Node last = head;

while (last.next != null) {

last = last.next;

}

last.next = new\_node;

}

// Function to print the linked list

public void printList() {

Node temp = head;

while (temp != null) {

System.out.print(temp.data + " ");

temp = temp.next;

}

System.out.println();

}

// Function to perform bubble sort on the linked list

public void bubbleSort() {

int swap;

Node ptr1;

Node lptr = null;

// If linked list is empty, return

if (head == null)

return;

do {

swap = 0;

ptr1 = head;

while (ptr1.next != lptr) {

if (ptr1.data > ptr1.next.data) {

// Swap nodes

int temp = ptr1.data;

ptr1.data = ptr1.next.data;

ptr1.next.data = temp;

swap = 1;

}

ptr1 = ptr1.next;

}

lptr = ptr1;

} while (swap != 0);

}

}

public class Main {

public static void main(String[] args) {

LinkedList list = new LinkedList();

list.append(64);

list.append(34);

list.append(25);

list.append(12);

list.append(22);

list.append(11);

System.out.println("Linked List before sorting:");

list.printList();

list.bubbleSort();

System.out.println("Linked List after sorting:");

list.printList();

}

}

**=================================================================================**

**10.Bubble Sort On Doubly Linked List**

class Node {

int data;

Node prev, next;

public Node(int data) {

this.data = data;

this.prev = null;

this.next = null;

}

}

class DoublyLinkedList {

Node head, tail;

public DoublyLinkedList() {

this.head = null;

this.tail = null;

}

// Method to add a new node to the end of the doubly linked list

public void addNode(int data) {

Node newNode = new Node(data);

if (head == null) {

head = tail = newNode;

head.prev = null;

tail.next = null;

} else {

tail.next = newNode;

newNode.prev = tail;

tail = newNode;

tail.next = null;

}

}

// Method to perform bubble sort on the doubly linked list

public void bubbleSort() {

if (head == null)

return;

Node current, index;

int temp;

for (current = head; current.next != null; current = current.next) {

for (index = current.next; index != null; index = index.next) {

if (current.data > index.data) {

temp = current.data;

current.data = index.data;

index.data = temp;

}

}

}

}

// Method to print the doubly linked list

public void display() {

Node current = head;

if (head == null) {

System.out.println("List is empty");

return;

}

System.out.println("Nodes of doubly linked list:");

while (current != null) {

System.out.print(current.data + " ");

current = current.next;

}

System.out.println();

}

}

public class Main {

public static void main(String[] args) {

DoublyLinkedList list = new DoublyLinkedList();

list.addNode(9);

list.addNode(5);

list.addNode(7);

list.addNode(1);

list.addNode(3);

System.out.println("Original Doubly Linked List:");

list.display();

// Sorting the list using Bubble Sort

list.bubbleSort();

System.out.println("\nDoubly Linked List after Bubble Sort:");

list.display();

}

}