Task 1:

Algorithm for selection sort

1. Start at the beginning of the list.
2. Look through the entire list to find the smallest value.
3. Swap that smallest value with the first value in the list.
4. Now, move to the second position in the list.
5. Again, look through the rest of the list (starting from the second position) to find the next smallest value.
6. Swap that value with the one at the second position.
7. Repeat this process for each position in the list:
   * For the third position, find the smallest value from the third to the last item and swap it.
   * Then for the fourth, and so on...
8. Keep going until the entire list is sorted.

Task 2:

Write a pseudo code for the selection sort

for i = 0 to n - 2:

min\_index = i

for j = i + 1 to n - 1:

if array[j] < array[min\_index]:

min\_index = j

swap array[i] and array[min\_index]

**Example**

Suppose we have this list:  
[5, 3, 8, 4, 2]

* Step 1: Find the smallest number → 2, swap with 5 → [2, 3, 8, 4, 5]
* Step 2: Find the smallest from index 1 onward → 3, already in place → [2, 3, 8, 4, 5]
* Step 3: Find the smallest from index 2 onward → 4, swap with 8 → [2, 3, 4, 8, 5]
* Step 4: Find the smallest from index 3 onward → 5, swap with 8 → [2, 3, 4, 5, 8]

Task 3:

Wap to make sure your list is sorted using selection sort.

package July11;  
  
public class SelectionSortExample {  
  
 // Method to perform selection sort  
 public static void selectionSort(int[] arr) {  
 int n = arr.length;  
  
 // Loop through the array  
 for (int i = 0; i < n - 1; i++) {  
 // Assume the current position is the minimum  
 int minIndex = i;  
  
 // Find the index of the smallest element in the remaining array  
 for (int j = i + 1; j < n; j++) {  
 if (arr[j] < arr[minIndex]) {  
 minIndex = j;  
 }  
 }  
  
 // Swap the found minimum with the current element  
 int temp = arr[minIndex];  
 arr[minIndex] = arr[i];  
 arr[i] = temp;  
 }  
 }  
  
 // Method to print the array  
 public static void printArray(int[] arr) {  
 for (int num : arr) {  
 System.*out*.print(num + " ");  
 }  
 System.*out*.println();  
 }  
  
 // Main method  
 public static void main(String[] args) {  
 int[] myList = {64, 25, 12, 22, 11};  
  
 System.*out*.println("Original array:");  
 *printArray*(myList);  
  
 // Sort the array using selection sort  
 *selectionSort*(myList);  
  
 System.*out*.println("Sorted array:");  
 *printArray*(myList);  
 }  
}

output:

Original array:

64 25 12 22 11

Sorted array:

11 12 22 25 64

Process finished with exit code 0

Task 4:

Write algorithm for the Bubble sort.

* Start at the beginning of the list.
* Compare the first and second elements.
* If the first is greater than the second, swap them.
* Move to the next pair (second and third), and repeat the comparison and swapping if needed.
* Continue this process until the end of the list—this completes one full pass.
* After each pass, the largest unsorted element "bubbles up" to its correct position.
* Repeat the entire process for the remaining (unsorted) portion of the list.
* Stop when no more swaps are needed, meaning the list is sorted.

Task 5:

Write pseudo code for the bubble sort

for i = 0 to n - 1:

for j = 0 to n - i - 2:

if array[j] > array[j + 1]:

swap array[j] and array[j + 1]

Task 6:

Wap to make sure your list is sorted using Bubble sort.

package July11;  
  
public class BubbleSortExample {  
  
 // Method to perform bubble sort  
 public static void bubbleSort(int[] arr) {  
 int n = arr.length;  
 boolean swapped;  
  
 // Outer loop for passes  
 for (int i = 0; i < n - 1; i++) {  
 swapped = false;  
  
 // Inner loop for comparing adjacent elements  
 for (int j = 0; j < n - i - 1; j++) {  
 // If elements are in the wrong order, swap them  
 if (arr[j] > arr[j + 1]) {  
 // Swap arr[j] and arr[j+1]  
 int temp = arr[j];  
 arr[j] = arr[j + 1];  
 arr[j + 1] = temp;  
  
 swapped = true;  
 }  
 }  
  
 // If no two elements were swapped in the inner loop, list is sorted  
 if (!swapped) {  
 break;  
 }  
 }  
 }  
  
 // Method to print the array  
 public static void printArray(int[] arr) {  
 for (int num : arr) {  
 System.*out*.print(num + " ");  
 }  
 System.*out*.println();  
 }  
  
 // Main method  
 public static void main(String[] args) {  
 int[] myList = {45, 12, 89, 33, 25};  
  
 System.*out*.println("Original array:");  
 *printArray*(myList);  
  
 // Sort the array using bubble sort  
 *bubbleSort*(myList);  
  
 System.*out*.println("Sorted array:");  
 *printArray*(myList);  
 }  
}

output:

Original array:

45 12 89 33 25

Sorted array:

12 25 33 45 89

Process finished with exit code 0

Task 7:

Write an algorithm for the Insertion sort.

* Start from the second element (index 1), since the first element is trivially sorted.
* Store the current element as key.
* Compare the key with the elements before it.
* Shift all larger elements one position to the right.
* Insert the key in its correct sorted position.
* Repeat this for all elements until the array is sorted.

Task 8:

Write pseudocode for the Insertion sort

InsertionSort(array, n):

for i from 1 to n - 1:

key = array[i]

j = i - 1

// Move elements that are greater than key to one position ahead

while j >= 0 and array[j] > key:

array[j + 1] = array[j]

j = j - 1

array[j + 1] = key

Task 9:

Wap to make sure your list is sorted using Insertion sort.

package July11;  
  
public class InsertionSortExample {  
  
 // Method to perform insertion sort  
 public static void insertionSort(int[] arr) {  
 int n = arr.length;  
  
 // Start from the second element  
 for (int i = 1; i < n; i++) {  
 int key = arr[i]; // Current element to insert  
 int j = i - 1;  
  
 // Move elements greater than key to one position ahead  
 while (j >= 0 && arr[j] > key) {  
 arr[j + 1] = arr[j];  
 j--;  
 }  
  
 // Place key at its correct position  
 arr[j + 1] = key;  
 }  
 }  
  
 // Method to print the array  
 public static void printArray(int[] arr) {  
 for (int num : arr) {  
 System.*out*.print(num + " ");  
 }  
 System.*out*.println();  
 }  
  
 // Main method  
 public static void main(String[] args) {  
 int[] myList = {29, 10, 14, 37, 14};  
  
 System.*out*.println("Original array:");  
 *printArray*(myList);  
  
 // Sort the array using insertion sort  
 *insertionSort*(myList);  
  
 System.*out*.println("Sorted array:");  
 *printArray*(myList);  
 }  
}

output:

Original array:

29 10 14 37 14

Sorted array:

10 14 14 29 37

Process finished with exit code 0

Task 10:

What are the advantages and disadvantages of Bubble sort Algo?

List them

note:

Poor performance - limitations of bubble sort

Advantages of Bubble Sort

1. Simple to understand and implement
   * Very straightforward algorithm suitable for beginners.
2. Does not require extra memory
   * It is an in-place sorting algorithm (space complexity = O(1)).
3. Stable sort
   * Maintains the relative order of equal elements.
4. Best case performance is better with optimization
   * If the array is already sorted, with an optimized version, it can stop early (O(n)).

Disadvantages (Limitations) of Bubble Sort

1. Poor overall performance
   * Time complexity is O(n²) in average and worst cases.
   * Very slow for large datasets.
2. Inefficient for real-world use
   * Not suitable for large or performance-critical applications.
3. Too many unnecessary comparisons and swaps
   * Even after most of the array is sorted, it continues checking all elements.
4. Does not scale well
   * The performance drastically drops as the number of elements increases.
5. Redundant passes
   * Without optimization, it will always do n-1 passes even if the list is already sorted.

13.01 to 13.05

Task 11:

This code is going overflow of stack.. Can you plz help me fix it guys.. ☹️

Note:

Plz be careful: Because recursive calls consume stack memory for every invocation and excessive depth can exceed system limits also..

public class RecLoop {

     public int calc(int n) {

        if (n == 0) return 0;

        return n + calc(n);

    }

Code:

public class RecLoop {  
  
 public int calc(int n) {  
 if (n == 0) return 0; // base case  
 return n + calc(n - 1); // recursive step toward base case  
 }  
  
 public static void main(String[] args) {  
 RecLoop rl = new RecLoop();  
 int result = rl.calc(5); // For example: 5 + 4 + 3 + 2 + 1 + 0 = 15  
 System.*out*.println("Result: " + result);  
 }  
}

output:

Result: 15

Process finished with exit code 0

Task 12:

Algorithm for merge sort,

Merge Sort is a divide-and-conquer algorithm:

1. Divide the list into two halves.
2. Recursively sort each half.
3. Merge the two sorted halves into a single sorted list.

Task 13

pseudo code for merge sort,

procedure mergeSort(array)

if length(array) <= 1

return array

mid ← length(array) / 2

left ← mergeSort(array[0 ... mid - 1])

right ← mergeSort(array[mid ... end])

return merge(left, right)

TSK 14

code for Merge sort

package July11;  
  
public class MergeSortExample {  
  
 // Method to sort the array using merge sort  
 public static void mergeSort(int[] array, int left, int right) {  
 if (left < right) {  
 // Find the middle point  
 int mid = (left + right) / 2;  
  
 // Recursively sort first and second halves  
 *mergeSort*(array, left, mid);  
 *mergeSort*(array, mid + 1, right);  
  
 // Merge the sorted halves  
 *merge*(array, left, mid, right);  
 }  
 }  
  
 // Method to merge two sorted halves  
 public static void merge(int[] array, int left, int mid, int right) {  
 // Sizes of the two subarrays  
 int n1 = mid - left + 1;  
 int n2 = right - 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] = array[left + i];  
 for (int j = 0; j < n2; j++)  
 R[j] = array[mid + 1 + j];  
  
 // Merge the temp arrays  
  
 int i = 0, j = 0, k = left;  
 while (i < n1 && j < n2) {  
 if (L[i] <= R[j]) {  
 array[k] = L[i];  
 i++;  
 } else {  
 array[k] = R[j];  
 j++;  
 }  
 k++;  
 }  
  
 // Copy remaining elements of L[]  
 while (i < n1) {  
 array[k] = L[i];  
 i++;  
 k++;  
 }  
  
 // Copy remaining elements of R[]  
 while (j < n2) {  
 array[k] = R[j];  
 j++;  
 k++;  
 }  
 }  
  
 // Method to print the array  
 public static void printArray(int[] array) {  
 for (int num : array)  
 System.*out*.print(num + " ");  
 System.*out*.println();  
 }  
  
 // Main method  
 public static void main(String[] args) {  
 int[] data = {38, 27, 43, 3, 9, 82, 10};  
  
 System.*out*.println("Original array:");  
 *printArray*(data);  
  
 *mergeSort*(data, 0, data.length - 1);  
  
 System.*out*.println("Sorted array:");  
 *printArray*(data);  
 }  
}

output:

Original array:

38 27 43 3 9 82 10

Sorted array:

3 9 10 27 38 43 82

Process finished with exit code 0

Task 15: Algo for quick sort

Quick Sort is a divide-and-conquer algorithm:

1. Choose a pivot element from the array.
2. Partition the array so that:
   * Elements less than the pivot go to the left.
   * Elements greater than the pivot go to the right.
3. Recursively apply the same steps to the left and right subarrays.
4. Combine the results (implicitly — as it's an in-place sort).

Task 16:

Pseudo code for quick sort

procedure quickSort(array, low, high)

if low < high then

// Partition the array and get pivot index

pivotIndex ← partition(array, low, high)

// Recursively apply to left part

quickSort(array, low, pivotIndex - 1)

// Recursively apply to right part

quickSort(array, pivotIndex + 1, high)

Task 17:

Code for Quick sort

package July11;  
public class QuickSortExample {  
  
 // Main method to call quick sort  
 public static void main(String[] args) {  
 int[] arr = { 34, 7, 23, 32, 5, 62 };  
  
 System.*out*.println("Original array:");  
 *printArray*(arr);  
  
 *quickSort*(arr, 0, arr.length - 1);  
  
 System.*out*.println("Sorted array:");  
 *printArray*(arr);  
 }  
  
 // QuickSort recursive method  
 public static void quickSort(int[] arr, int low, int high) {  
 if (low < high) {  
 // Partition the array and get the pivot index  
 int pivotIndex = *partition*(arr, low, high);  
  
 // Recursively sort elements before and after partition  
 *quickSort*(arr, low, pivotIndex - 1);  
 *quickSort*(arr, pivotIndex + 1, high);  
 }  
 }  
  
 // Partition method  
 public static int partition(int[] arr, int low, int high) {  
 int pivot = arr[high]; // choosing the last element as pivot  
 int i = low - 1; // index of smaller element  
  
 for (int j = low; j < high; j++) {  
 if (arr[j] < pivot) {  
 i++;  
 // swap arr[i] and arr[j]  
 int temp = arr[i];  
 arr[i] = arr[j];  
 arr[j] = temp;  
 }  
 }  
  
 // swap arr[i + 1] and arr[high] (pivot)  
 int temp = arr[i + 1];  
 arr[i + 1] = arr[high];  
 arr[high] = temp;  
  
 return i + 1; // return the pivot index  
 }  
  
 // Utility method to print the array  
 public static void printArray(int[] arr) {  
 for (int num : arr)  
 System.*out*.print(num + " ");  
 System.*out*.println();  
 }  
}

code:

Original array:

34 7 23 32 5 62

Sorted array:

5 7 23 32 34 62

Process finished with exit code 0