Task 1:

Write an algorithm / steps for selection sort.

1. Start with an unsorted array.
2. Find the minimum element in the unsorted part
3. Swap it with the first element of unsorted part
4. Move the boundary of unsorted part one element to the right
5. Repeat until the array is sorted

Task 2:

Write a pseudo code for the selection sort

n = length of array

for i from 0 to n-1

minIndex = i

for j from i+1 to n

if array[j] < array[minIndex]

minIndex = j

swap array[i] and array[minIndex]

Task 3:

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

public class task03 {

    public static void main(String[] args) {

        int[] array = {64, 34, 25, 12, 22, 11, 90};

        System.out.println("Given array:");

        printArray(array);

        for (int i = 0; i < array.length - 1; i++) {

            int minIndex = i;

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

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

                    minIndex = j;

                }

            }

            int temp = array[minIndex];

            array[minIndex] = array[i];

            array[i] = temp;

        }

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

        printArray(array);

    }

    public static void printArray(int[] arr) {

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

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

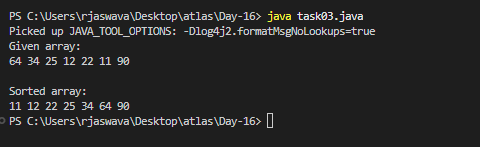
        }

        System.out.println();

    }

}

Output:



Task 4:

Write algorithm for the Bubble sort.

1. Start with an unsorted array

2. Compare adjacent elements (first with second, second with third, etc.)

3. If the first element is greater than the second element, swap them

4. Move to the next pair of adjacent elements

5. Repeat steps 3-4 until reaching the end of array

6. One complete pass will move the largest element to the end

7. Repeat steps 2-6 for the remaining unsorted portion

8. Continue until no more swaps are needed

Task 5:

Write pseudo code for the bubble sort

bubbleSort(array)

n = length of array

for i from 0 to n-1

swapped = false

for j from 0 to n-1-i

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

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

swapped = true

if swapped equals false

break

Task 6:

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

public class task06 {

    public static void main(String[] args) {

        int[] array = {64, 34, 25, 12, 22, 11, 90};

        System.out.println("Given array:");

        printArray(array);

        boolean swapped;

        for (int i = 0; i < array.length - 1; i++) {

            swapped = false;

            for (int j = 0; j < array.length - 1 - i; j++) {

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

                    int temp = array[j];

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

                    array[j + 1] = temp;

                    swapped = true;

                }

            }

            if (!swapped) {

                break;

            }

        }

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

        printArray(array);

    }

    public static void printArray(int[] arr) {

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

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

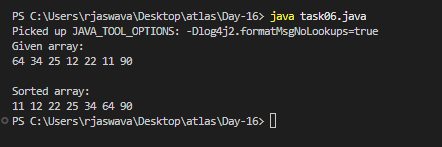
        }

        System.out.println();

    }

}

Output:



Task 7:

Write an algorithm for the Insertion sort.

1. Start with the first element (consider it as sorted)

2. Take the next element

3. Compare it with elements in sorted portion from right to left

4. Insert the element at the correct position in sorted portion

5. Repeat steps 2-4 until all elements are processed

6. The array will be sorted when all elements are inserted in their correct positions

Task 8:

Write pseudocode for the Insertion sort

insertionSort(array)

n = length of array

for i from 1 to n-1

key = array[i]

j = i - 1

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.

public class task09 {

    public static void main(String[] args) {

        int[] array = {64, 34, 25, 12, 22, 11, 90};

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

        printArray(array);

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

            int key = array[i];

            int j = i - 1;

            while (j >= 0 && array[j] > key) {

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

                j = j - 1;

            }

            array[j + 1] = key;

            System.out.println("\nAfter iteration " + i + ":");

            printArray(array);

        }

        System.out.println("\nFinal sorted array:");

        printArray(array);

    }

    public static void printArray(int[] arr) {

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

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

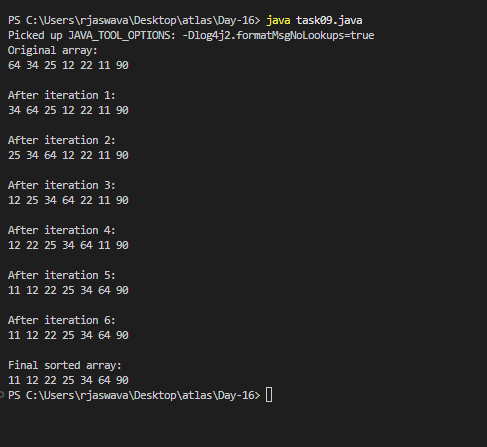
        }

        System.out.println();

    }

}

Output:



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 Implementation

- Easy to understand and code

- Requires only a few lines of code

- Good for teaching sorting concepts

2. Memory Efficient

- In-place sorting algorithm

- No extra memory space needed (except for one temp variable)

- Space complexity is O(1)

3. Stable Sorting

- Maintains relative order of equal elements

- Good when maintaining original order is important

4. Adaptive

- Can detect if array is already sorted

- Performance improves for partially sorted arrays

- Can be optimized with a flag to detect sorted array

DISADVANTAGES OF BUBBLE SORT:

1. Poor Time Complexity

- Worst case: O(n²)

- Average case: O(n²)

- Not suitable for large datasets

2. Performance Limitations

- Requires multiple passes through the array

- Many unnecessary swaps even if array is nearly sorted

- Each element is compared with every other element

3. Inefficient Compared to Other Algorithms

- Much slower than advanced algorithms like:

\* Quick Sort O(n log n)

\* Merge Sort O(n log n)

\* Heap Sort O(n log n)

4. Scaling Issues

- Performance degrades quadratically with input size

- For n elements, requires n² steps in worst case

- Example:

\* 100 elements = 10,000 steps

\* 1000 elements = 1,000,000 steps

5. Resource Usage

- High number of swap operations

- More CPU cycles used due to multiple passes

- Energy inefficient for large datasets

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);

    }

public class task11 {

    //sol-1

    public int calc(int n) {

        if (n == 0) return 0;

        return n + calc(n - 1);

    }

   //sol-2

    public int calcIterative(int n) {

        int sum = 0;

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

            sum += i;

        }

        return sum;

    }

    public static void main(String[] args) {

        task11 obj = new task11();

        int n = 5;

        System.out.println("Using recursion: " + obj.calc(n));

        System.out.println("Using iteration: " + obj.calcIterative(n));

        try {

            int largeNumber = 10000;

            System.out.println("Sum of large number (iterative): " +

                             obj.calcIterative(largeNumber));

        } catch (StackOverflowError e) {

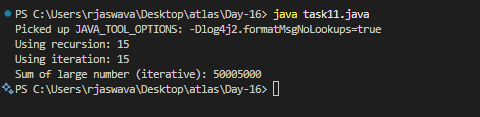
            System.out.println("Stack overflow occurred!");

        }

    }

}

Output:



TAsk 12:

Algo for merge sort,

Merge Sort Algorithm Steps:

1. Divide Phase:

- Divide the unsorted array into two halves

- Continue dividing until sub-arrays have only one element

2. Merge Phase:

- Compare elements from two sub-arrays

- Merge them in sorted order

- Continue merging until all sub-arrays are combined

Task 13

pseudo code for merge sort,

mergeSort(array, left, right)

if left < right then

mid = (left + right) / 2

mergeSort(array, left, mid)

mergeSort(array, mid + 1, right)

merge(array, left, mid, right)

end if

procedure merge(array, left, mid, right)

n1 = mid - left + 1

n2 = right - mid

create leftArray[n1], rightArray[n2]

for i = 0 to n1-1

leftArray[i] = array[left + i]

for j = 0 to n2-1

rightArray[j] = array[mid + 1 + j]

i = 0, j = 0, k = left

while i < n1 and j < n2

if leftArray[i] <= rightArray[j]

array[k] = leftArray[i]

i = i + 1

else

array[k] = rightArray[j]

j = j + 1

k = k + 1

while i < n1

array[k] = leftArray[i]

i = i + 1

k = k + 1

while j < n2

array[k] = rightArray[j]

j = j + 1

k = k + 1

TSK 14

code for Merge sort

public class task14 {

    public static void main(String[] args) {

        int[] array = {64, 34, 25, 12, 22, 11, 90};

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

        printArray(array);

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

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

        printArray(array);

    }

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

        if (left < right) {

            int mid = left + (right - left) / 2;

            mergeSort(array, left, mid);

            mergeSort(array, mid + 1, right);

            merge(array, left, mid, right);

        }

    }

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

        int n1 = mid - left + 1;

        int n2 = right - mid;

        int[] leftArray = new int[n1];

        int[] rightArray = new int[n2];

        for (int i = 0; i < n1; i++)

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

        for (int j = 0; j < n2; j++)

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

        int i = 0;

        int j = 0;

        int k = left;

        while (i < n1 && j < n2) {

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

                array[k] = leftArray[i];

                i++;

            } else {

                array[k] = rightArray[j];

                j++;

            }

            k++;

        }

        while (i < n1) {

            array[k] = leftArray[i];

            i++;

            k++;

        }

        while (j < n2) {

            array[k] = rightArray[j];

            j++;

            k++;

        }

    }

    static void printArray(int[] arr) {

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

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

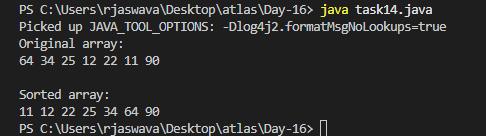
        }

        System.out.println();

    }

}

Output:



Task 15:

Algo fro quick sort

1. Choose a Pivot:

- Select a pivot element from the array

- Common choices: last element, first element, or random element

2. Partitioning:

- Place pivot in its correct position

- Move smaller elements to left of pivot

- Move larger elements to right of pivot

3. Recursion:

- Recursively apply steps 1-2 to sub-array before pivot

- Recursively apply steps 1-2 to sub-array after pivot

4. Base Case:

- Stop when sub-array has 1 or 0 elements

- Array is sorted when all recursions complete

Task 16:

Pseudo code for quick sort

quickSort(array, low, high)

if low < high then

// pi is partitioning index

pi = partition(array, low, high)

quickSort(array, low, pi - 1)

quickSort(array, pi + 1, high)

end if

end procedure

partition(array, low, high)

// Select rightmost element as pivot

pivot = array[high]

i = low - 1

for j = low to high - 1

if array[j] <= pivot then

i = i + 1

swap array[i] with array[j]

swap array[i + 1] with array[high]

return i + 1

Task 17:

Code for Quick sort

public class task17 {

    public static void main(String[] args) {

        int[] array = {64, 34, 25, 12, 22, 11, 90};

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

        printArray(array);

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

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

        printArray(array);

    }

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

        if (low < high) {

            int pi = partition(array, low, high);

            System.out.println("\nAfter partitioning around pivot:");

            printArray(array);

            quickSort(array, low, pi - 1);

            quickSort(array, pi + 1, high);

        }

    }

    static int partition(int[] array, int low, int high) {

        int pivot = array[high];

        int i = (low - 1);

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

            if (array[j] <= pivot) {

                i++;

                int temp = array[i];

                array[i] = array[j];

                array[j] = temp;

            }

        }

        int temp = array[i + 1];

        array[i + 1] = array[high];

        array[high] = temp;

        return i + 1;

    }

    static void printArray(int[] arr) {

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

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

        }

        System.out.println();

    }

}

Output:

