Day 16- 11th july 2025

Selection Sort:

**Task 👍1**: Write an algorithm / steps for selection sort.

Step 1: Create an array with size n

Step 2: Search for the largest number from 0 to n-1 in the array

Step 3: Swap the largest number with the (n-1)th number

Step 4: Move to step 2 by reducing the n value by 1

**Task 2**: Write a pseudo code for the selection sort

Function SelectionSort(arr)

 For i = 0 to length(arr) - 1

   minIndex = i

   For j = i + 1 to length(arr) - 1

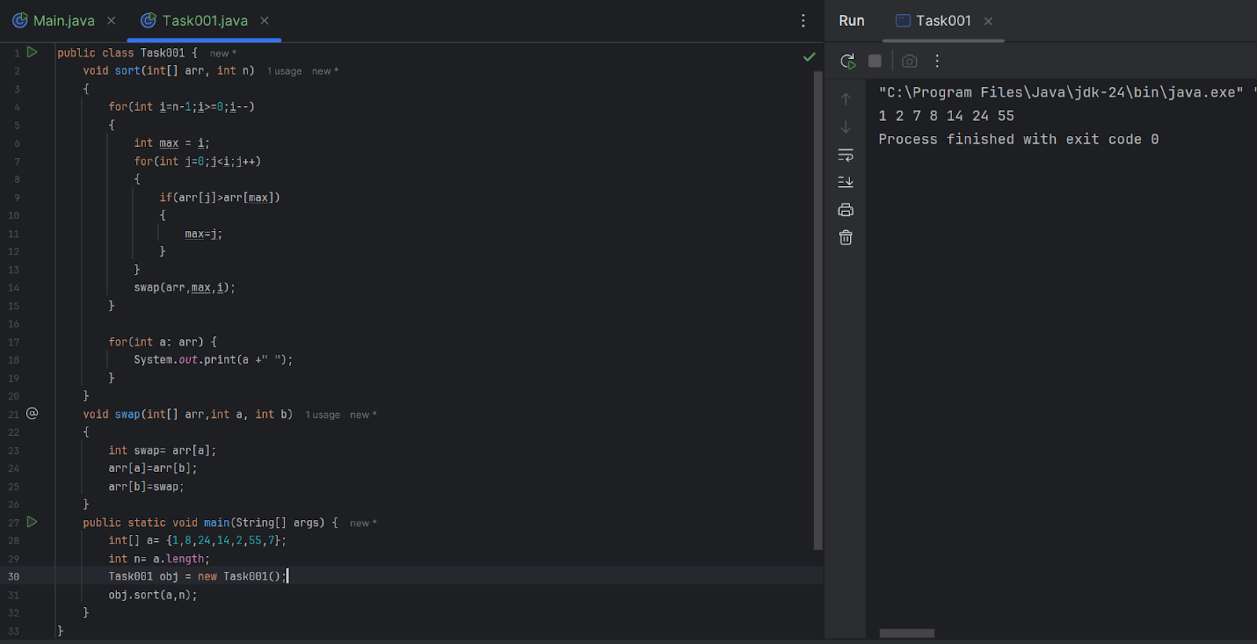
     If arr[j] < arr[minIndex]

       minIndex = j

   Swap arr[i] and arr[minIndex]

Print the arr

**Task 3**: Wap to make sure your list is sorted using selection sort.



Bubble Sort:

**Task 4**: Write algorithm for the Bubble sort.

Step 1: Compare the pair of adjacent elements

Step 2: Swap them if they are not in order

Step 3: Continue this until the end of the array

Step 4: Reduce the value of n by 1 and go to step 1.

**Task 5**: Write pseudo code for the bubble sort

Sequential-Bubble-Sort (A)

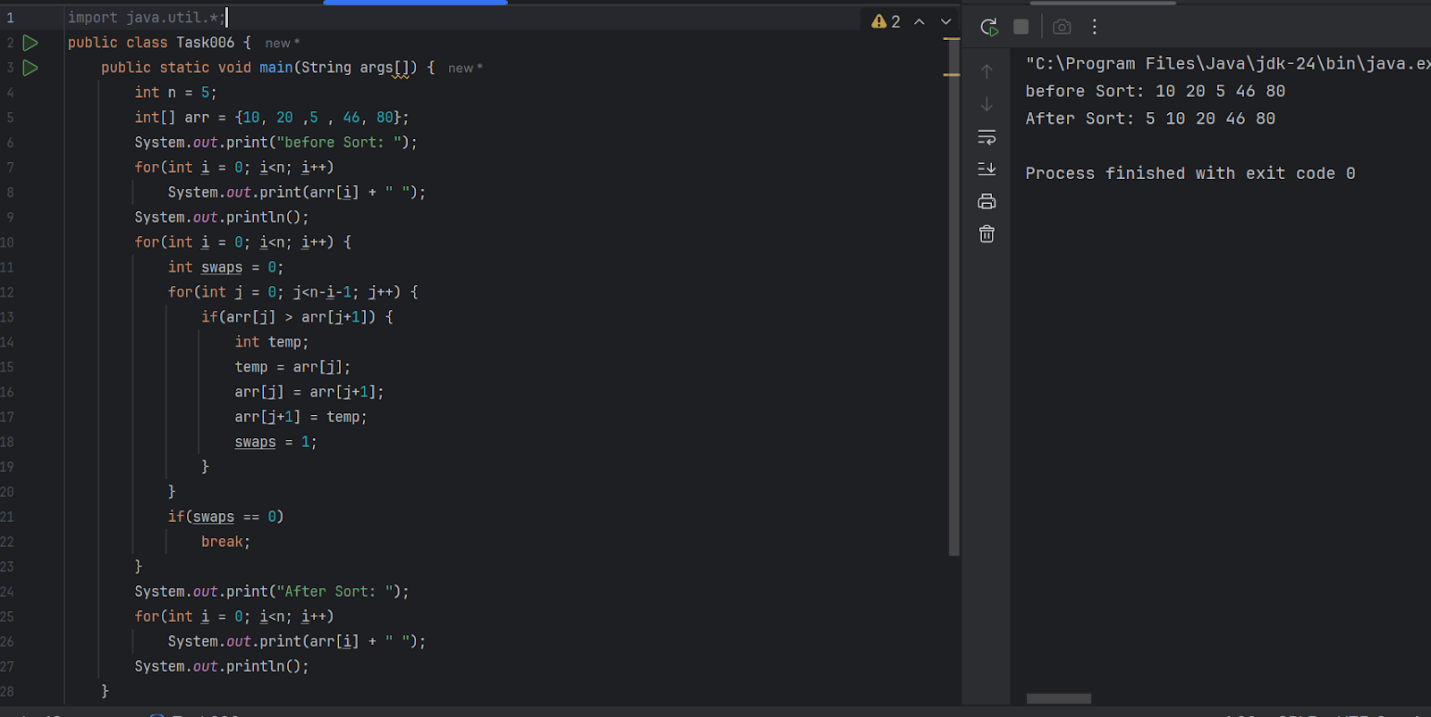
for i 1 to n

for j n to i +1

   if A[A] < A[j-1] then

      Exchange A[j] ⟷ A[j-1]

**Task 6**: Wap to make sure your list is sorted using Bubble sort.



**Task 7**: Write an algorithm for the Insertion sort.

Step 1: Start with the second element of the list (since the first element is considered sorted).

Compare this element with the elements before it.

Step 2: Insert this element into its correct position among the sorted elements.

Step 3: Move to the next element and repeat the comparison and insertion process.

Step 4: Continue this process until you reach the end of the list. By the end of each pass, the list of sorted elements grows by one.

Repeat until the entire list is sorted.

**Task 8**: Write pseudocode for the Insertion sort

Algorithm InsertionSort

Input: A list of elements

Output: A sorted list of elements

1. For i from 1 to length of list - 1

    a. Set key to list[i]

    b. Set j to i - 1

    c. While j >= 0 and list[j] > key

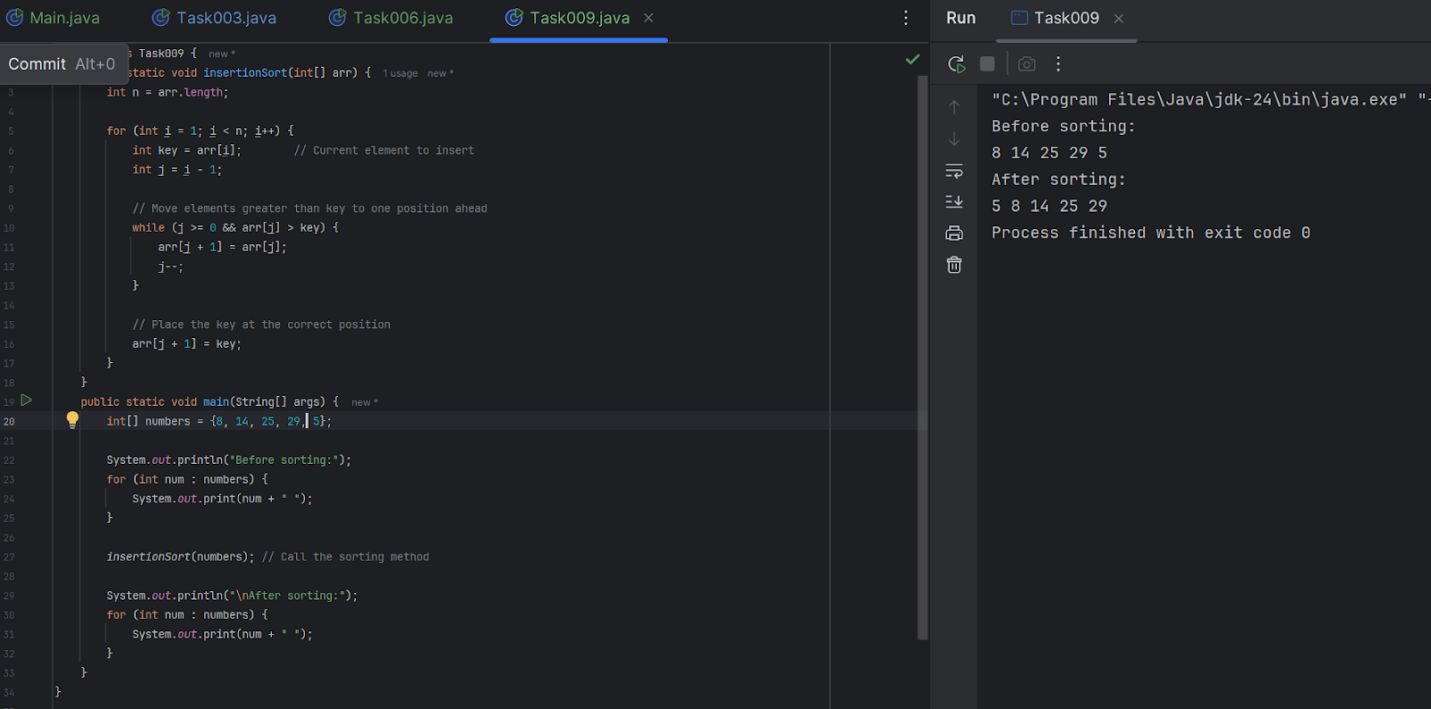
        i. Set list[j + 1] to list[j]

        ii. Decrement j by 1

    d. Set list[j + 1] to key

2. End For

**Task 9**: Wap to make sure your list is sorted using Insertion sort.



**Task 10**: What are the advantages and disadvantages of Bubble sort Algo?

List them

note:

Poor performance - limitations of bubble sort

**Advantages**: 1. Simplicity and ease of Implementation

2. In place sorting

3. It performs well on nearly sorted data

**Disadvantages:** 1. It is inefficient when working with large data

2. Not suitable for performance critical applications

3. Performs a large number of swaps than other sorting techniques

**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..

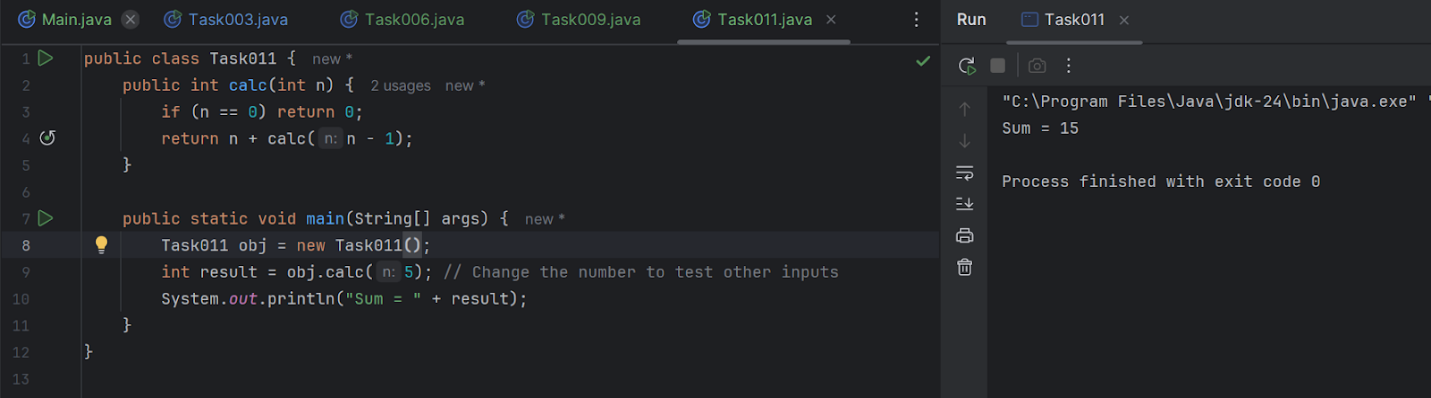
public class RecLoop {

     public int calc(int n) {

        if (n == 0) return 0;

        return n + calc(n);

    }



**Task 12**: Algo for merge sort,

**Divide:** Split the list into two halves until each sublist contains a single element.

**Conquer:** Recursively sort each sublist.

**Combine:** Merge the sorted sublists to produce new sorted sublists until there is only one sorted list.

**Task 13**: Pseudo code for merge sort,

Algorithm MergeSort(list)

Input: A list of elements

Output: A sorted list of elements

1. If length of list <= 1

  a. Return list

2. Divide the list into two halves

a. left\_half = list[0:mid]

  b. right\_half = list[mid:end]

3. Recursively sort each half

  a. sorted\_left = MergeSort(left\_half)

  b. sorted\_right = MergeSort(right\_half)

4. Merge the sorted halves

  a. Return Merge(sorted\_left, sorted\_right)

Algorithm Merge(left, right)

Input: Two sorted lists

Output: A single sorted list

1. Initialize an empty list to hold the merged result

2. While both left and right are not empty

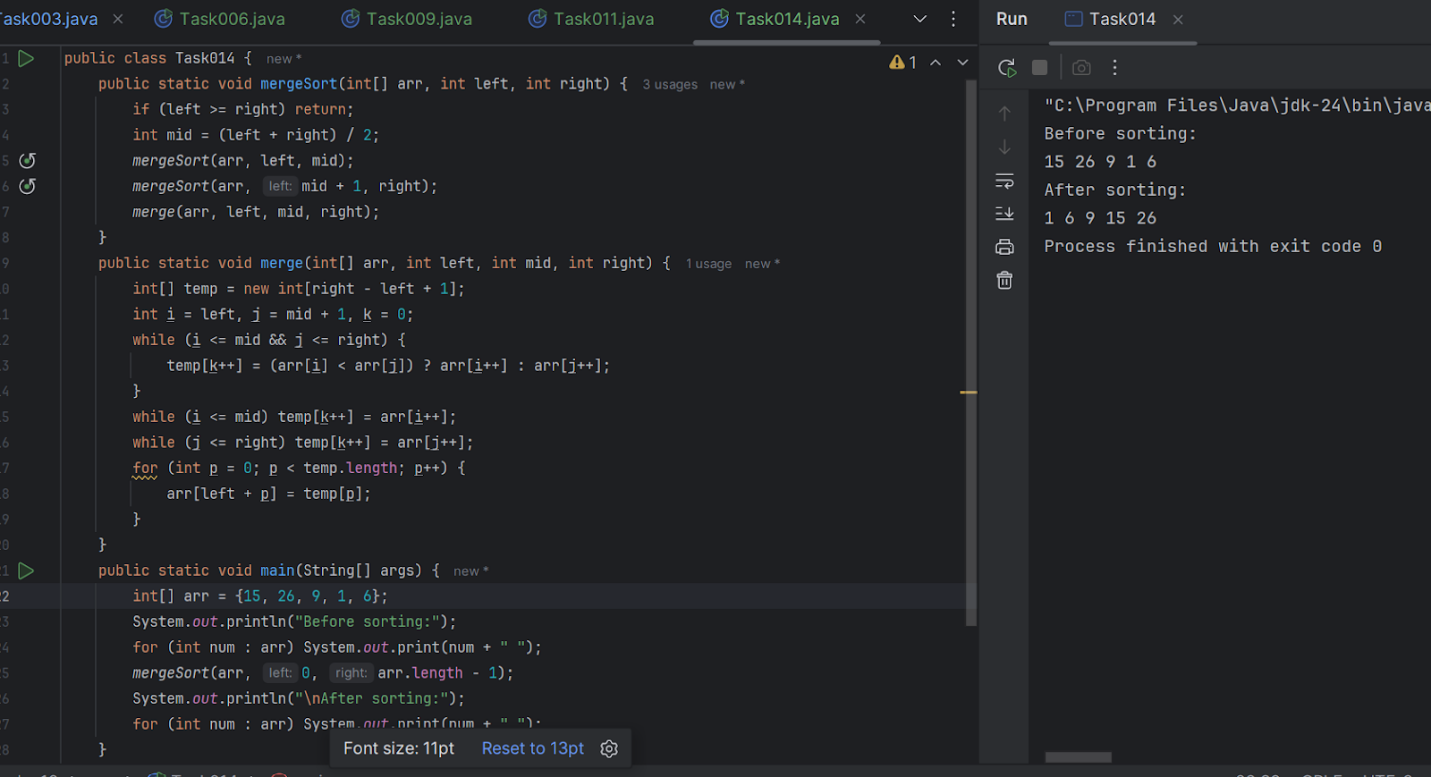
  a. Compare the first elements of both lists

  b. Remove the smaller element and append it to the merged list

3. If any elements remain in either list, append them to the merged list

4. Return the merged list

**Task 14**: code for Merge sort



**Task 15**: Algo for quick sort

**Choose a Pivot:** Select a pivot element from the list.

**Partition**: Initialize two pointers: one at the start and one at the end of the list.

Move the start pointer to the right until an element greater than the pivot is found.

Move the end pointer to the left until an element less than the pivot is found.

Swap these elements.

Repeat until the pointers cross.

Place the pivot in its correct position.

**Recursively Sort**: Apply the quick sort algorithm to the sublists of elements less than and greater than the pivot.

**Task 16**: Pseudo code for quick sort

Algorithm QuickSort(list, low, high)

Input: A list of elements, starting index (low), and ending index (high)

Output: A sorted list of elements

1. If low < high

    a. pi = Partition(list, low, high)

    b. QuickSort(list, low, pi - 1)

    c. QuickSort(list, pi + 1, high)

Algorithm Partition(list, low, high)

Input: A list of elements, starting index (low), and ending index (high)

Output: The index of the pivot element

1. Set pivot to list[high]

2. Set i to low - 1

3. For j from low to high - 1

    a. If list[j] < pivot

        i. Increment i

        ii. Swap list[i] with list[j]

4. Swap list[i + 1] with list[high]

5. Return i + 1

**Task 17**: Code for Quick sort

