**Task001**

**Write an algorithm / steps for selection sort.**

Create the array

Find the largest number x in the range 0 to n-1  
Swap this largest element with the last element of the unsorted that is n-1

Reduce n by 1

Back to step 1

**Example**

Array a ={5,3,7,2,6,1}

Here n=6

Max is 7 at index 2

Swap a[5] and a[2]

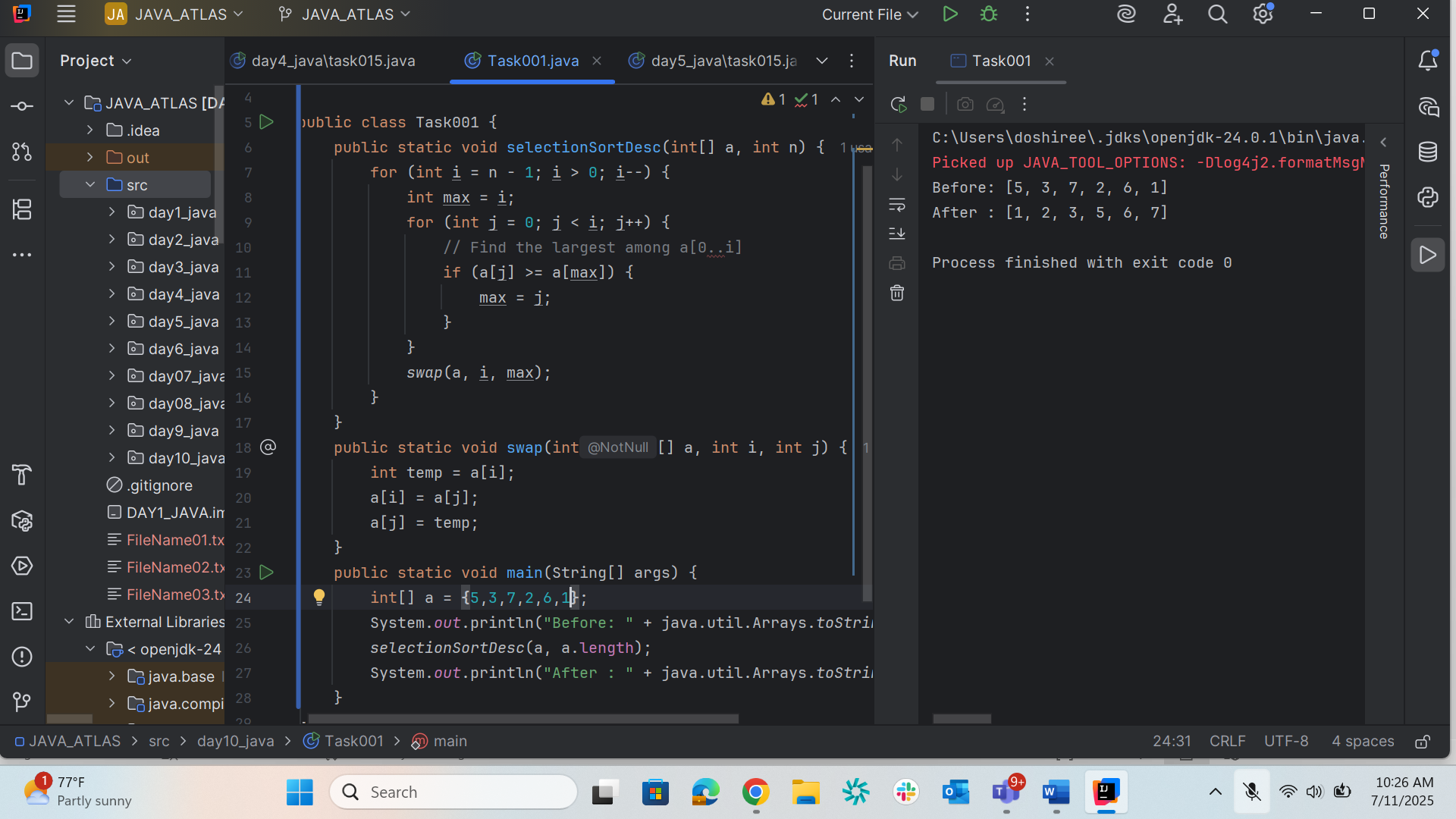
**Task002**

Pseudo code

void selectionSort(int[] a, int n) {  
 for (int i = n - 1; i > 0; i--) {  
 int max = i;  
 for (int j = 0; j < i; j++) {  
 if (a[j] >= a[max]) {  
 max = j;  
 }  
 }  
 *swap*(a, i, max);  
 }  
}

task003

package day10\_java;  
  
import static java.util.Collections.*swap*;  
  
public class Task001 {  
 public static void selectionSort(int[] a, int n) {  
 for (int i = n - 1; i > 0; i--) {  
 int max = i;  
 for (int j = 0; j < i; j++) {  
 // Find the largest among a[0..i]  
 if (a[j] >= a[max]) {  
 max = j;  
 }  
 }  
 *swap*(a, i, max);  
 }  
 }  
 public static void swap(int[] a, int i, int j) {  
 int temp = a[i];  
 a[i] = a[j];  
 a[j] = temp;  
 }  
 public static void main(String[] args) {  
 int[] a = {5,3,7,2,6,1};  
 System.*out*.println("Before: " + java.util.Arrays.*toString*(a));  
 *selectionSort*(a, a.length);  
 System.*out*.println("After : " + java.util.Arrays.*toString*(a));  
 }  
}



Bubble sort

Task004

Step 1: Take an array which is unsorted

Step 2: Compare pair of adjacent items

Step 3: Swap if the items are not in order

Step 4: Repeat the end of array

Step 5: The largest item will be at the last position

Step 6: Reduce n by 1 and go to Step 1

Task005

Pseudo code

A = {20,40,80,10,60}

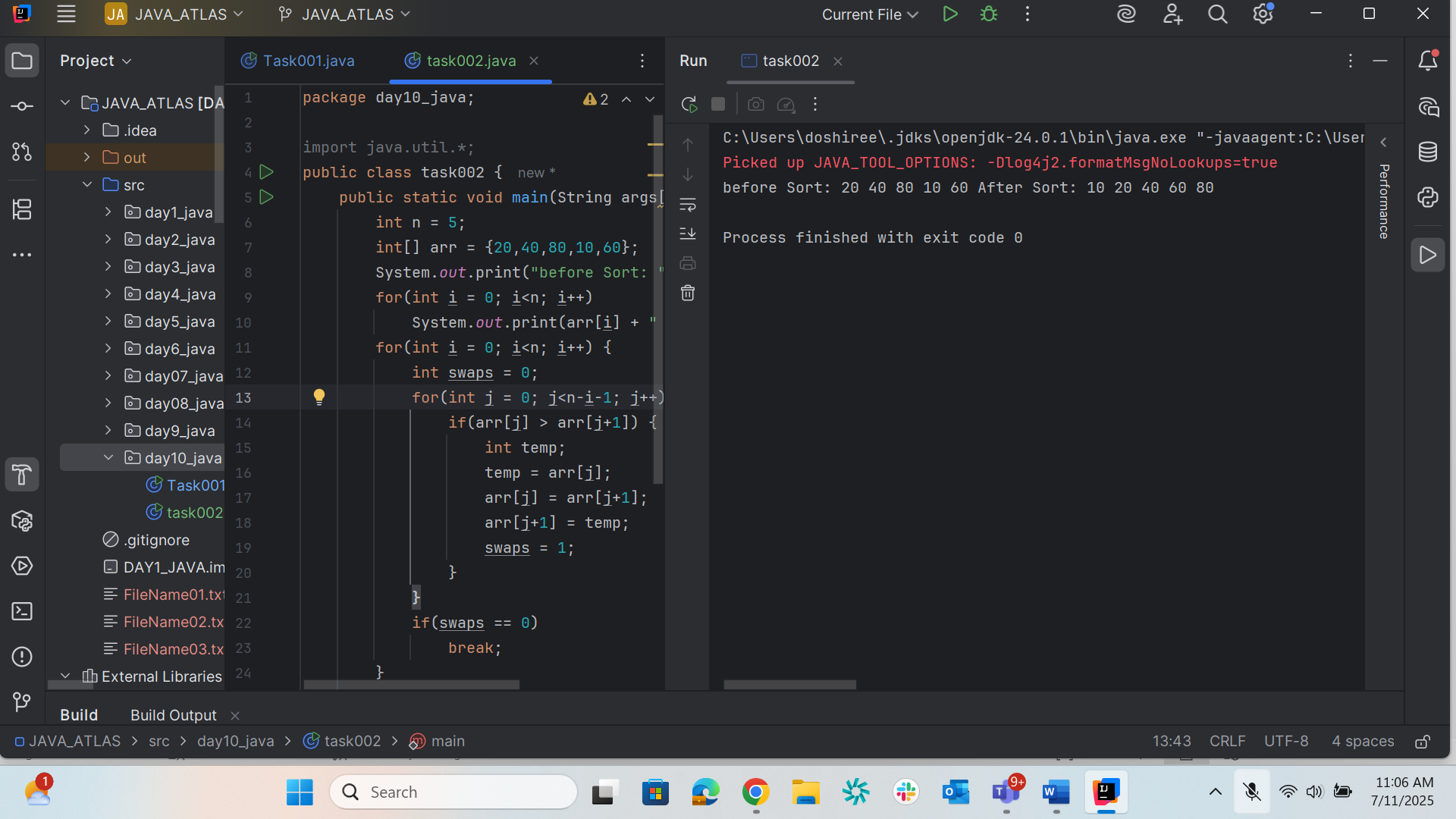
for(int i = 0; i<n; i++)

for(int j = 0; j<n-i-1; j++)   
 if(A[j] > A[j+1])

exchange a[j] and a[j+1]

task006

package day10\_java;  
  
import java.util.\*;  
public class task002 {  
 public static void main(String args[]) {  
 int n = 5;  
 int[] arr = {20,40,80,10,60};  
 System.*out*.print("before Sort: ");  
 for(int i = 0; i<n; i++)  
 System.*out*.print(arr[i] + " ");  
 for(int i = 0; i<n; i++) {  
 int swaps = 0;  
 for(int j = 0; j<n-i-1; j++) {  
 if(arr[j] > arr[j+1]) {  
 int temp;  
 temp = arr[j];  
 arr[j] = arr[j+1];  
 arr[j+1] = temp;  
 swaps = 1;  
 }  
 }  
 if(swaps == 0)  
 break;  
 }  
 System.*out*.print("After Sort: ");  
 for(int i = 0; i<n; i++)  
 System.*out*.print(arr[i] + " ");  
 System.*out*.println();  
 }  
}



Task007

1. create an array and If the first element already sorted

2. return 1

3. move to next element

4. Compare with all elements in the sorted list

5. Shift all the elements in the sorted list which is greater than the value to be sorted

6. Insert the value

7. Repeat until list is sorted

Task008

Insertion Sort(A)

for j = 2 to A.length

   key = A[j]

   i = j  1

   while i > 0 and A[i] > key

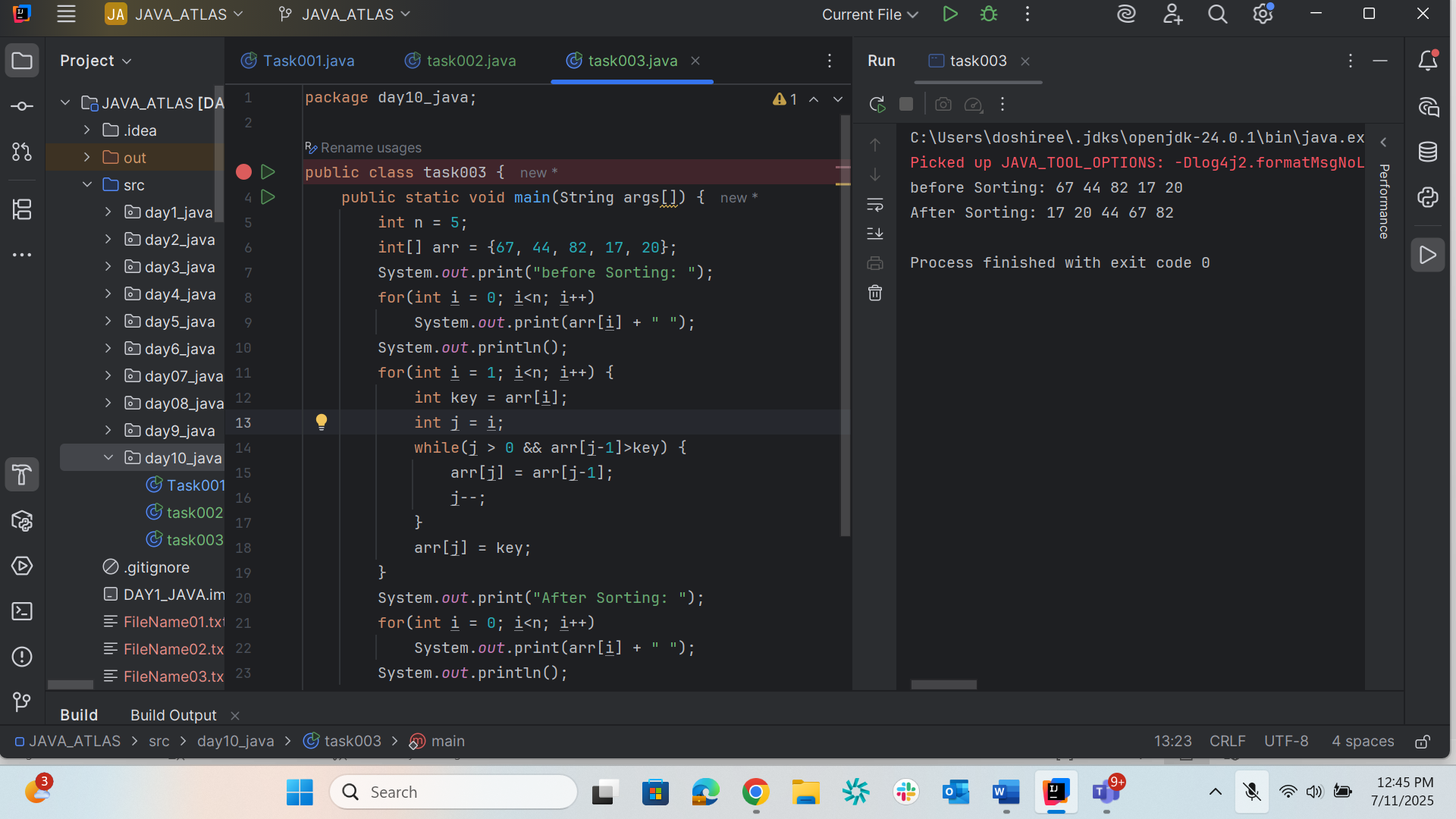
      A[i + 1] = A[i]

      i = i -1

   A[i + 1] = key

Task009

package day10\_java;  
  
public class task003 {  
 public static void main(String args[]) {  
 int n = 5;  
 int[] arr = {67, 44, 82, 17, 20};  
 System.*out*.print("before Sorting: ");  
 for(int i = 0; i<n; i++)  
 System.*out*.print(arr[i] + " ");  
 System.*out*.println();  
 for(int i = 1; i<n; i++) {  
 int key = arr[i];  
 int j = i;  
 while(j > 0 && arr[j-1]>key) {  
 arr[j] = arr[j-1];  
 j--;  
 }  
 arr[j] = key;  
 }  
 System.*out*.print("After Sorting: ");  
 for(int i = 0; i<n; i++)  
 System.*out*.print(arr[i] + " ");  
 System.*out*.println();  
 }  
}



Task10

Advantages and disadvantages of bubble sort

Advantages

1. Easy to implement – straight forward logic
2. Sorts using only the input array and a few variables
3. Stable sort
4. **O(n) best-case time**-If the array is already sorted, it can finish after one pass

Disadvantages

1. Very slow on large datasets
2. Rarely used in real applications
3. Unless optimized with an early-exit flag, it keeps scanning even when sorted

**task010**

1. just only one element in the list, consider it already sorted, so return.

2. Divide the list recursively into two halves until it can not be divided further.

3. Merge all the smaller lists into new list in sorted order.

**Task011**

   if length of array > 1 then

        mid ← length of array / 2

        leftHalf ← array[0 ... mid - 1]

        rightHalf ← array[mid ... end]

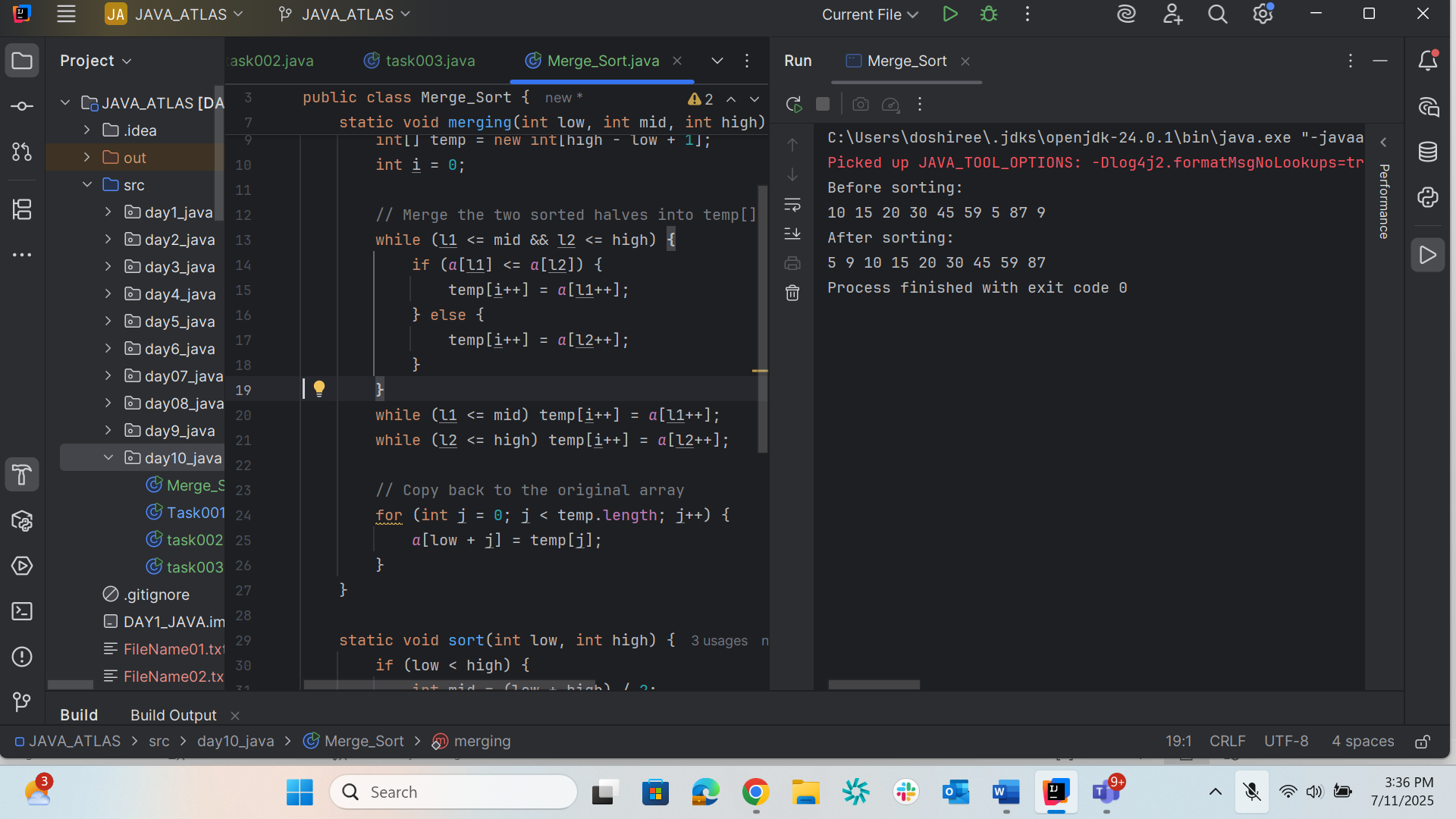
        mergeSort(leftHalf)

        mergeSort(rightHalf)

        merge(leftHalf, rightHalf, array)

**Task 12**

package day10\_java;  
  
public class Merge\_Sort {  
 static int[] *a* = {10, 15, 20, 30, 45, 59, 5, 87, 9};  
 static int[] *b* = {19, 12, 21, 30, 48, 56, 7, 86, 8};  
  
 static void merging(int low, int mid, int high) {  
 int l1 = low, l2 = mid + 1;  
 int[] temp = new int[high - low + 1];  
 int i = 0;  
  
 // Merge the two sorted halves into temp[]  
 while (l1 <= mid && l2 <= high) {  
 if (*a*[l1] <= *a*[l2]) {  
 temp[i++] = *a*[l1++];  
 } else {  
 temp[i++] = *a*[l2++];  
 }  
 }  
 while (l1 <= mid) temp[i++] = *a*[l1++];  
 while (l2 <= high) temp[i++] = *a*[l2++];  
  
 // Copy back to the original array  
 for (int j = 0; j < temp.length; j++) {  
 *a*[low + j] = temp[j];  
 }  
 }  
  
 static void sort(int low, int high) {  
 if (low < high) {  
 int mid = (low + high) / 2;  
 *sort*(low, mid);  
 *sort*(mid + 1, high);  
 *merging*(low, mid, high);  
 }  
 }  
  
 public static void main(String[] args) {  
 System.*out*.println("Before sorting:");  
 for (int num : *a*) System.*out*.print(num + " ");  
 *sort*(0, *a*.length - 1);  
 System.*out*.println("\nAfter sorting:");  
 for (int num : *a*) System.*out*.print(num + " ");  
 }  
}



Task013

Algo for quick sort

1. **Choose a pivot** element (e.g., last, first, random, median).
2. **Partition** the array so elements ≤ pivot are on its left and ≥ pivot on its right.
3. **Recursively sort** the left and right subarrays.
4. Stop when a subarray has size ≤ 1.

Task014

Quicksort : Pseudo code

partitionFunc(left, right, pivot)

   leftPointer = left

   rightPointer = right - 1

   while True do

      while A[++leftPointer] < pivot do

      end while

      while rightPointer > 0 && A[--rightPointer] > pivot do

      end while

      if leftPointer >= rightPointer

         break

      else

         swap leftPointer,rightPointer

      end if

   end while

   swap leftPointer,right

   return leftPointer

end

Task015

package day10\_java;

import java.util.Arrays;

public class QuickSort {

int[] intArray = {4,6,3,2,1,9,7};

void swap(int num1, int num2) {

int temp = intArray[num1];

intArray[num1] = intArray[num2];

intArray[num2] = temp;

}

int partition(int left, int right, int pivot) {

int leftPointer = left - 1;

int rightPointer = right;

while (true) {

while (intArray[++leftPointer] < pivot) {

// do nothing

}

while (rightPointer > 0 && intArray[--rightPointer] > pivot) {

// do nothing

}

if (leftPointer >= rightPointer) {

break;

} else {

swap(leftPointer, rightPointer);

}

}

swap(leftPointer, right);

// System.out.println("Updated Array: ");

return leftPointer;

}

void quickSort(int left, int right) {

if (right - left <= 0) {

return;

} else {

int pivot = intArray[right];

int partitionPoint = partition(left, right, pivot);

quickSort(left, partitionPoint - 1);

quickSort(partitionPoint + 1, right);

}

}

public static void main(String[] args) {

QuickSort sort = new QuickSort();

int max = sort.intArray.length;

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

System.out.println(Arrays.toString(sort.intArray));

sort.quickSort(0, max - 1);

System.out.println("Contents of the array after sorting :");

System.out.println(Arrays.toString(sort.intArray));

}

}

