Task 1 - Algorithm for selection sort

1. Start with an array of numbers.

2. Look through the array to find the largest number.

3. Move that largest number to the end of the array.

4. Repeat the process for the remaining unsorted part of the array.

5. Keep doing this until all numbers are arranged in order.

6. The array is now sorted in ascending order.

7. End.

Task 2 - Write a pseudo code for selection sort

Algorithm SelectionSort(a, n)

Input: Array a of size n

Output: Sorted array a in ascending order

For i ← n - 1 down to 1 do

maxIndex ← i

For j ← 0 to i - 1 do

If a[j] > a[maxIndex] then

maxIndex ← j

End If

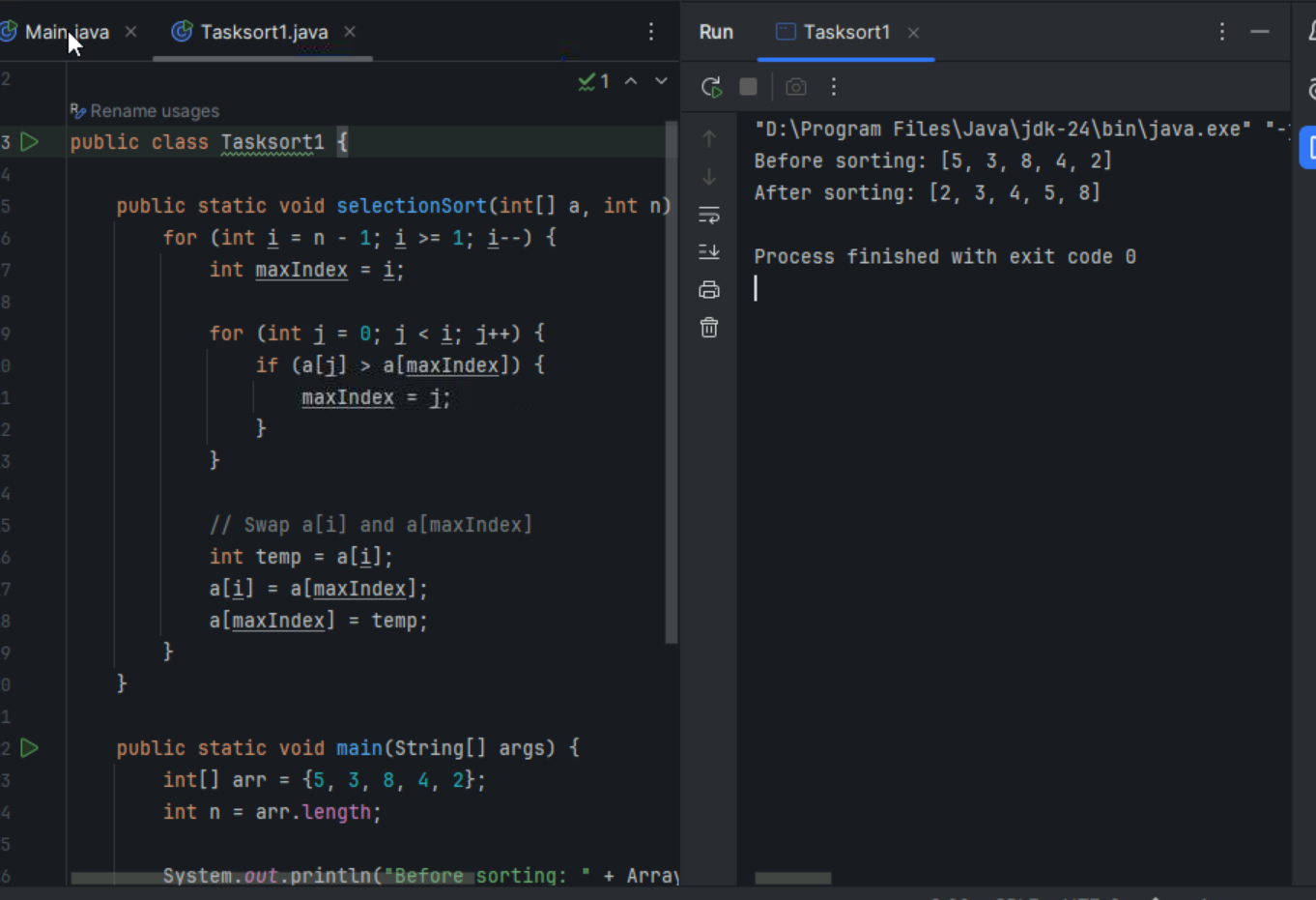
End For

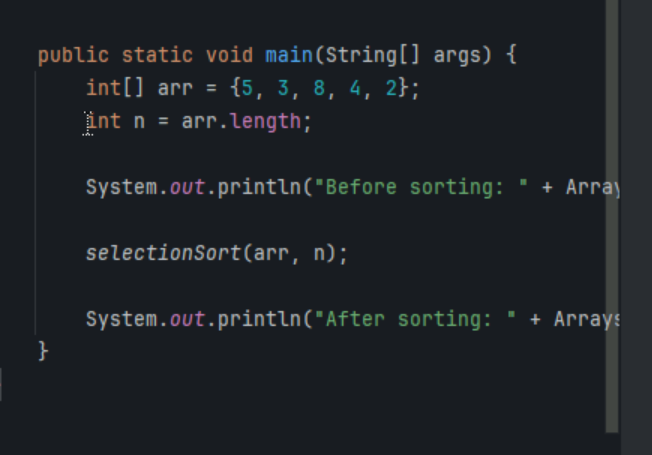
Swap a[i] with a[maxIndex]

End For

End Algorithm

Task3





Task 4 - Algorithm for bubble sort

1.Start with the first element of the list.

2.Compare this element with the next element.

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

4.Move to the next pair of elements and repeat the comparison and swap if necessary.

5.Continue this process until you reach the end of the list. By the end of this pass, the

6. largest element will have "bubbled up" to its correct position at the end of the list.

7. Repeat the process for the remaining elements, ignoring the last sorted elements.

8. Continue until the entire list is sorted.

Task5 - pseudo code for bubble sort

procedure bubbleSort(list)

n ← length of list

repeat (n - 1) times

for i ← 0 to n - 2

if list[i] > list[i + 1] then

swap list[i] and list[i + 1]

end if

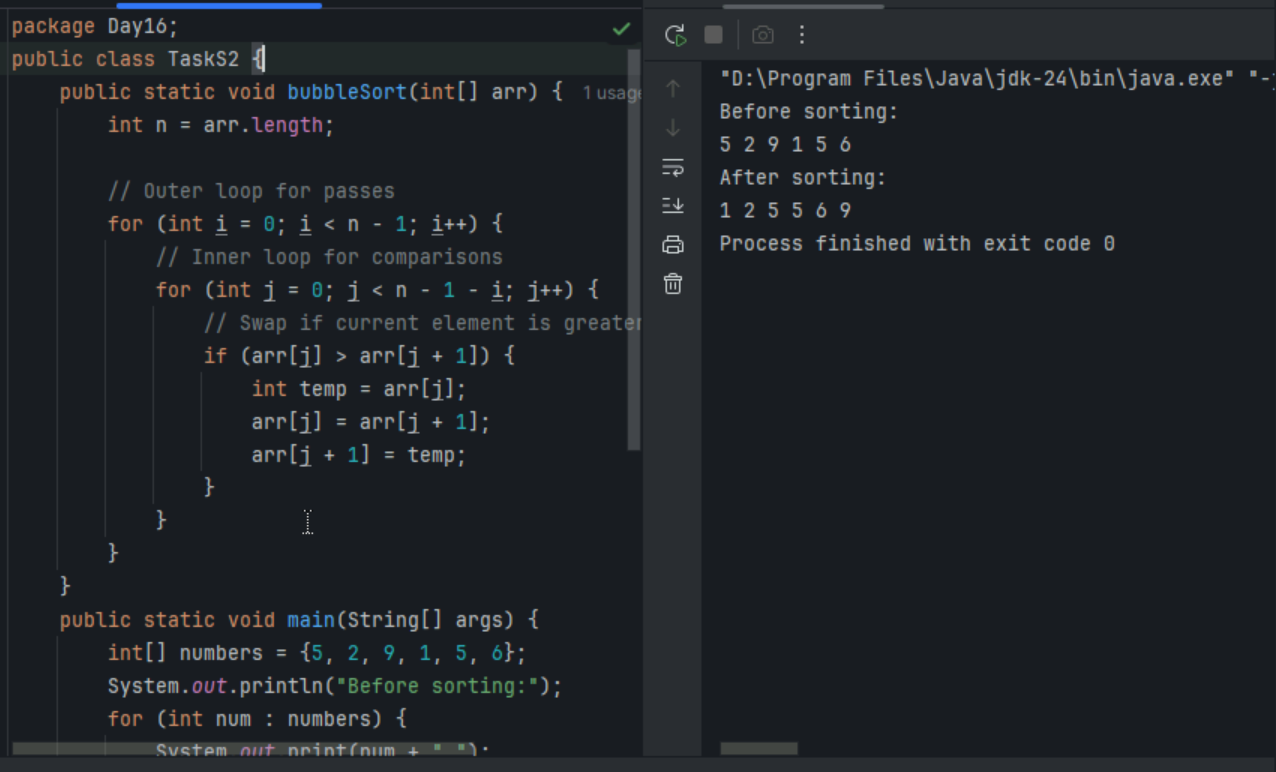
end for

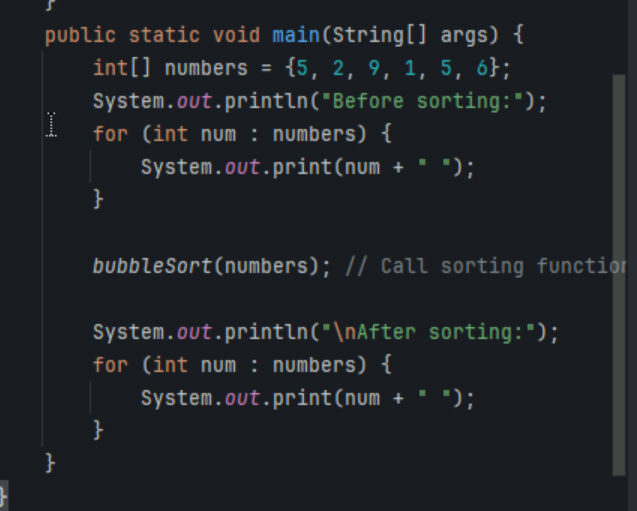
// After each pass, the largest element is at the end

end repeat

end procedure

Task 6





Task7 - Write an algorithm for insertion sort

1. **Start with the second element** of the list (since the first element is considered sorted).
2. **Compare** this element with the elements before it.
3. **Insert** this element into its correct position among the sorted elements.
4. Move to the next element and **repeat** the comparison and insertion process.
5. 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.
6. **Repeat** until the entire list is sorted.

Task8 - Write pseudo code for 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

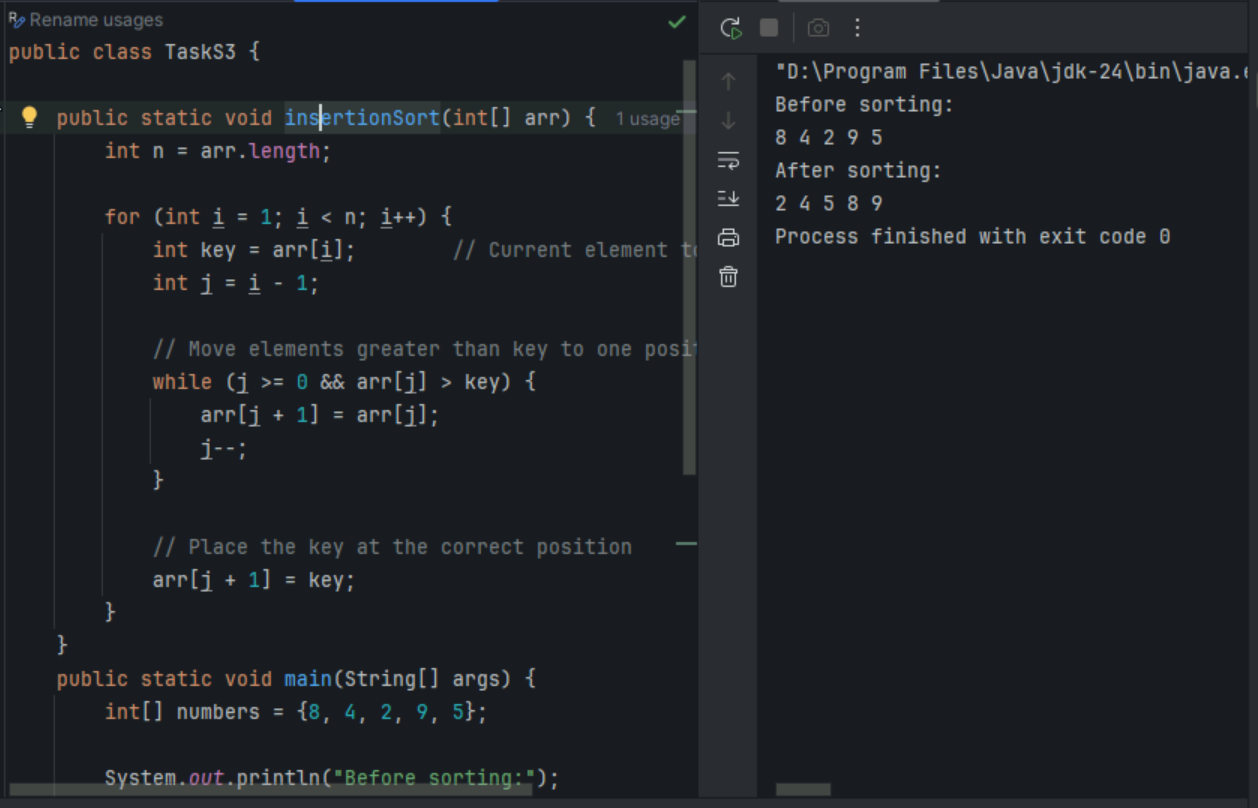
c. While j >= 0 and list[j] > key

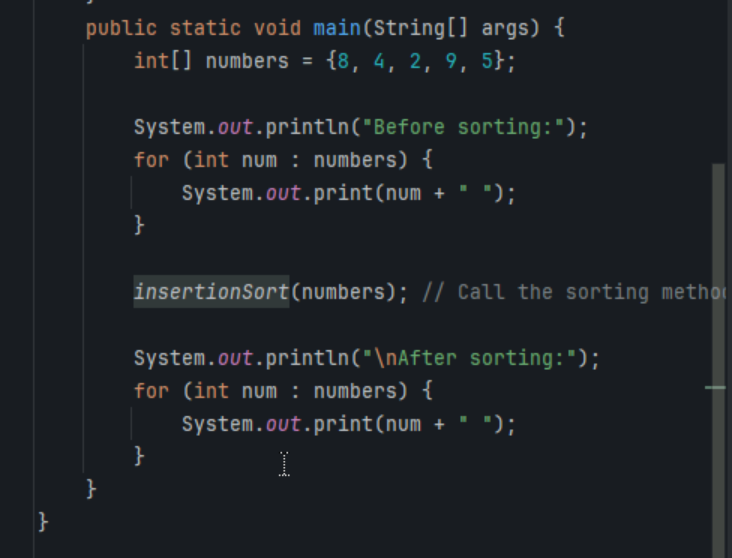
i. Set list[j + 1] to list[j]

ii. Decrement j by 1

d. Set list[j + 1] to key

2. End For

Task9 - 



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

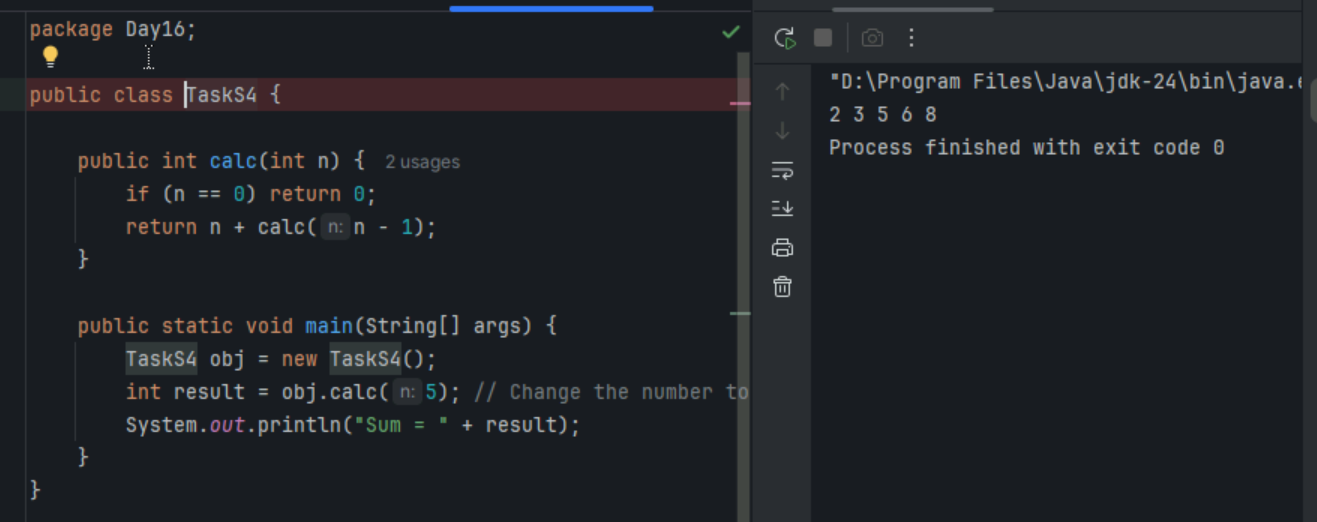
### **Advantages of Bubble Sort Algorithm:**

1. **Simplicity**: Bubble sort is easy to understand and implement.
2. **No Extra Space**: It sorts the list in place, requiring only a constant amount of additional memory space.
3. **Stable Sort**: It maintains the relative order of equal elements.
4. **Best Case Performance**: When the list is already sorted, bubble sort performs very well with a time complexity of (O(n)).

### **Disadvantages of Bubble Sort Algorithm:**

1. **Poor Performance**: Bubble sort has a worst-case and average-case time complexity of (O(n^2)), making it inefficient for large lists.
2. **Unnecessary Comparisons**: It continues to compare elements even if the list is already sorted, leading to unnecessary operations.
3. **Not Suitable for Large Data Sets**: Due to its quadratic time complexity, it is not practical for sorting large lists.
4. **High Number of Swaps**: It performs a large number of swaps, which can be costly in terms of time and wear on memory.

Task11



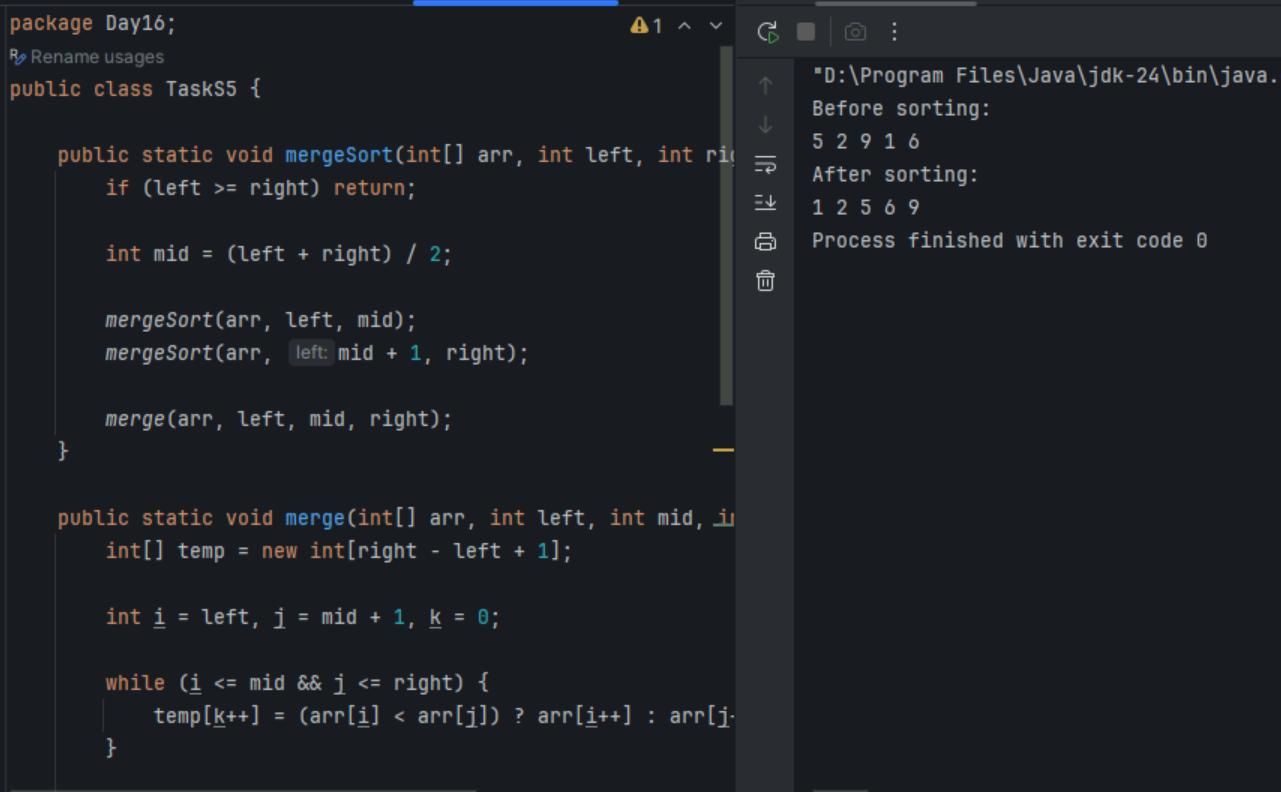
Task12 - Algorithm for merge sort

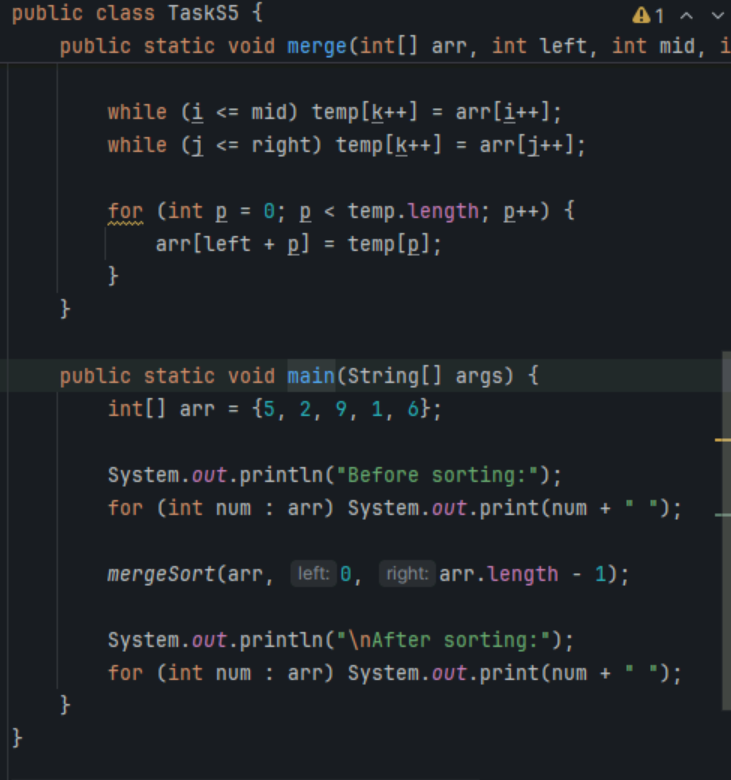
Divide: Split the list into two halves until each sublist contains a single element.  
Conquer: Recursively sort each sublist.  
Combine: Merge the sorted sublists to produce new sorted sublists until there is only one sorted list.

Task13 - 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

Task14





Task15 - Algorithm 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.

Task16 - 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

Task17

