Day 16 – 11/07/2025

Q1. Write an algorithm/step for selection sort.  
Ans.

Selection Sort Algorithm

1. Start with the first element: Consider the first element of the array as the minimum value.
2. Compare with remaining elements: Compare the current minimum value with the remaining elements in the array.
3. Find the minimum value: If a smaller value is found, update the minimum value and its index.
4. Swap the minimum value: After comparing all elements, swap the minimum value with the current element.
5. Repeat the process: Repeat steps 1-4 for the remaining unsorted elements in the array.

Q2. Write pseudocode for selection sort.  
Ans. code –

FOR i FROM 0 TO n-2

minIndex = i

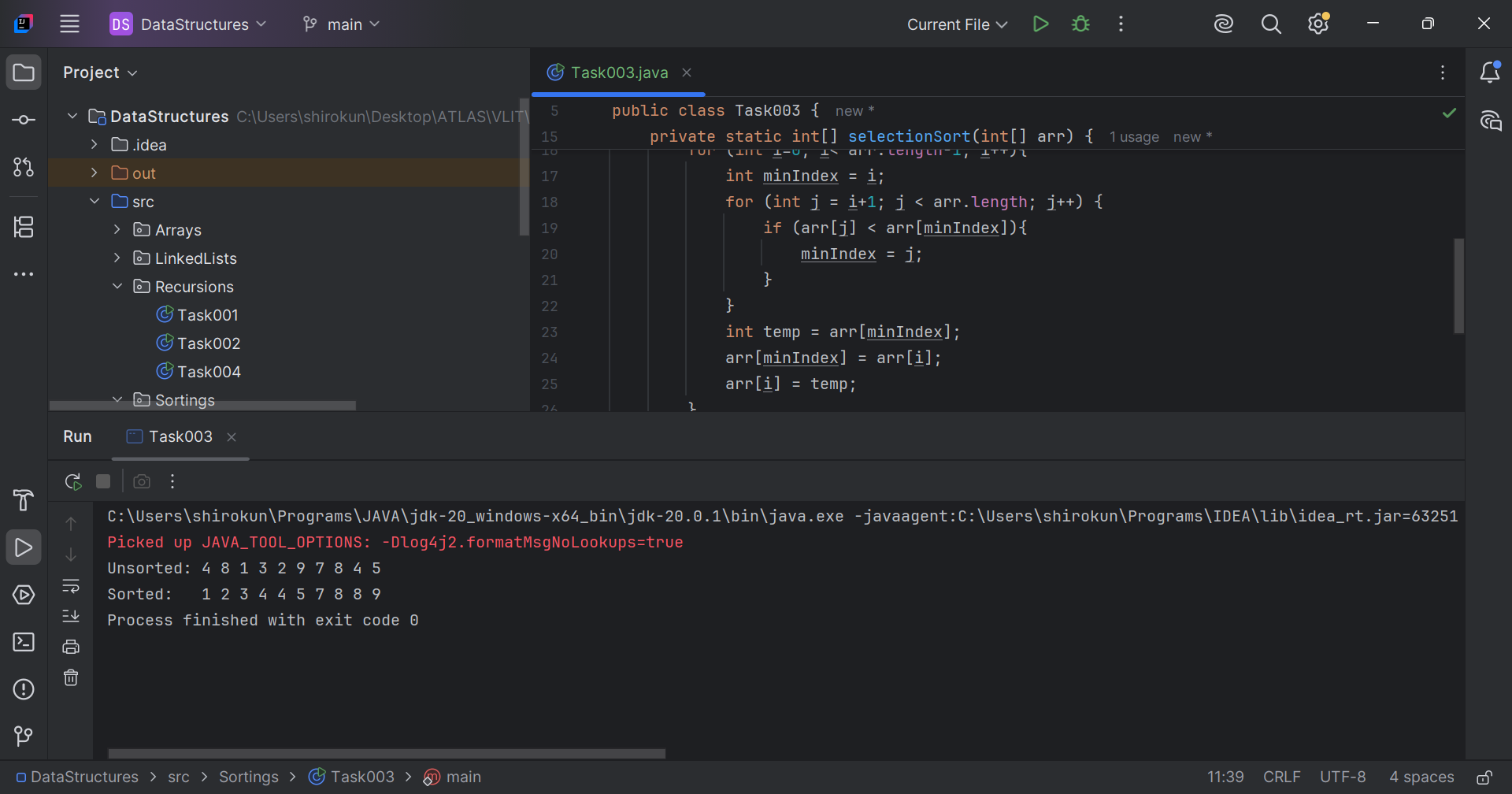
FOR j FROM i+1 TO n-1

IF arr[j] < arr[minIndex]

minIndex = j

SWAP arr[i] WITH arr[minIndex]

// Task003: Selection sort  
  
package Sortings;  
  
public class Task003 {  
 public static void main(String[] args) {  
 int[] sortArr, arr = {4, 8, 1, 3, 2, 9, 7, 8, 4, 5};  
 System.*out*.print("Unsorted: ");  
 for (int a: arr) System.*out*.print(a+" ");  
 sortArr = *selectionSort*(arr);  
 System.*out*.print("\nSorted: ");  
 for (int a: sortArr) System.*out*.print(a+" ");  
 }  
  
 private static int[] selectionSort(int[] arr) {  
 for (int i=0; i< arr.length-1; i++){  
 int minIndex = i;  
 for (int j = i+1; j < arr.length; j++) {  
 if (arr[j] < arr[minIndex]){  
 minIndex = j;  
 }  
 }  
 int temp = arr[minIndex];  
 arr[minIndex] = arr[i];  
 arr[i] = temp;  
 }  
 return arr;  
 }  
}



Q4. Write algorithm for Bubble sort.  
Ans. Bubble Sort Algorithm -

1. Start with the first element: Compare the first element with the next element.
2. Compare adjacent elements: If the current element is greater than the next element, swap them.
3. Repeat the process: Continue comparing and swapping adjacent elements until the end of the array is reached.
4. Repeat passes: Repeat the process until no more swaps are needed, indicating that the array is sorted.

Q5. Write pseudocode for Bubble sort.  
Ans.   
n = length of array

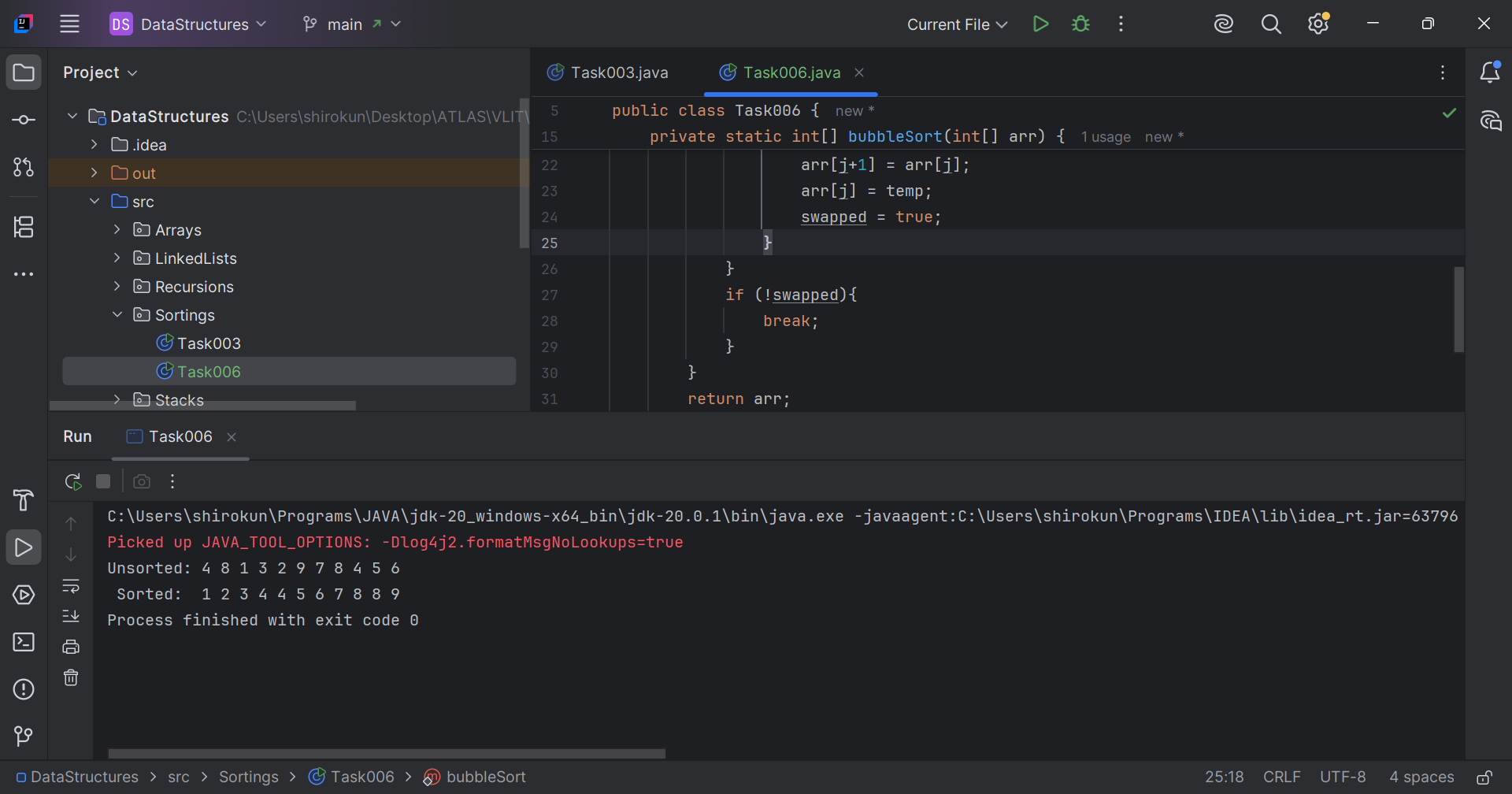
FOR i FROM 0 TO n-2

FOR j FROM 0 TO n-i-2

IF arr[j] > arr[j+1]

SWAP arr[j] WITH arr[j+1]

// Task006: Bubble sort  
  
package Sortings;  
  
public class Task006 {  
 public static void main(String[] args) {  
 int[] sortArr, arr = {4, 8, 1, 3, 2, 9, 7, 8, 4, 5, 6};  
 System.*out*.print("Unsorted: ");  
 for (int a: arr) System.*out*.print(a+" ");  
 System.*out*.print("\n Sorted: ");  
 sortArr = *bubbleSort*(arr);  
 for (int a: sortArr) System.*out*.print(a+" ");  
 }  
  
 private static int[] bubbleSort(int[] arr) {  
 int n = arr.length;  
 for (int i = 0; i < n-1; i++) {  
 for (int j = 0; j < n-i-1; j++) {  
 if (arr[j] > arr[j+1]){  
 int temp = arr[j+1];  
 arr[j+1] = arr[j];  
 arr[j] = temp;  
 }  
 }  
 }  
 return arr;  
 }  
}



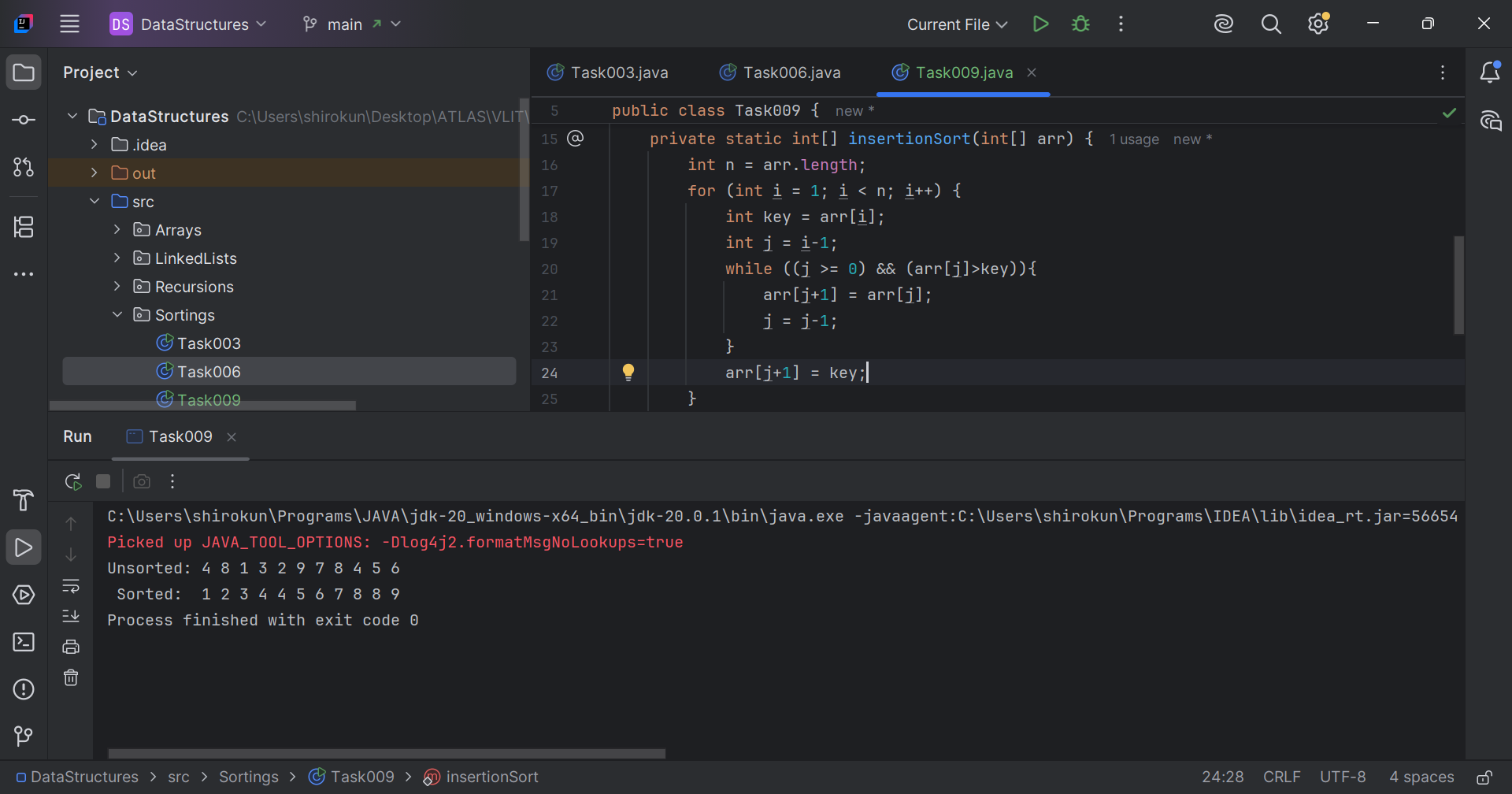
Q7. Write an algorithm for an insertion sort.   
Ans. Algorithm –

1. Start with the second element: Consider the second element as the key.
2. Compare with previous elements: Compare the key with the previous elements in the sorted portion.
3. Shift elements: If the key is smaller than a previous element, shift that element one position to the right.
4. Insert the key: Insert the key at the correct position in the sorted portion.
5. Repeat the process: Repeat steps 1-4 for the remaining elements.

Q8. Write pseudo code for insertion sort.  
Ans. Pseudo code –

* FOR i FROM 1 TO n-1
  + key = arr[i]
  + j = i - 1
  + WHILE j >= 0 AND arr[j] > key
    - arr[j + 1] = arr[j]
    - j = j - 1
  + arr[j + 1] = key

// Task009: Insertion sort.  
  
package Sortings;  
  
public class Task009 {  
 public static void main(String[] args) {  
 int[] sortArr, arr = {4, 8, 1, 3, 2, 9, 7, 8, 4, 5, 6};  
 System.*out*.print("Unsorted: ");  
 for (int a: arr) System.*out*.print(a+" ");  
 System.*out*.print("\n Sorted: ");  
 sortArr = *insertionSort*(arr);  
 for (int a: sortArr) System.*out*.print(a+" ");  
 }  
  
 private static int[] insertionSort(int[] arr) {  
 int n = arr.length;  
 for (int i = 1; i < n; i++) {  
 int key = arr[i];  
 int j = i-1;  
 while ((j >= 0) && (arr[j]>key)){  
 arr[j+1] = arr[j];  
 j = j-1;  
 }  
 arr[j+1] = key;  
 }  
 return arr;  
 }  
}



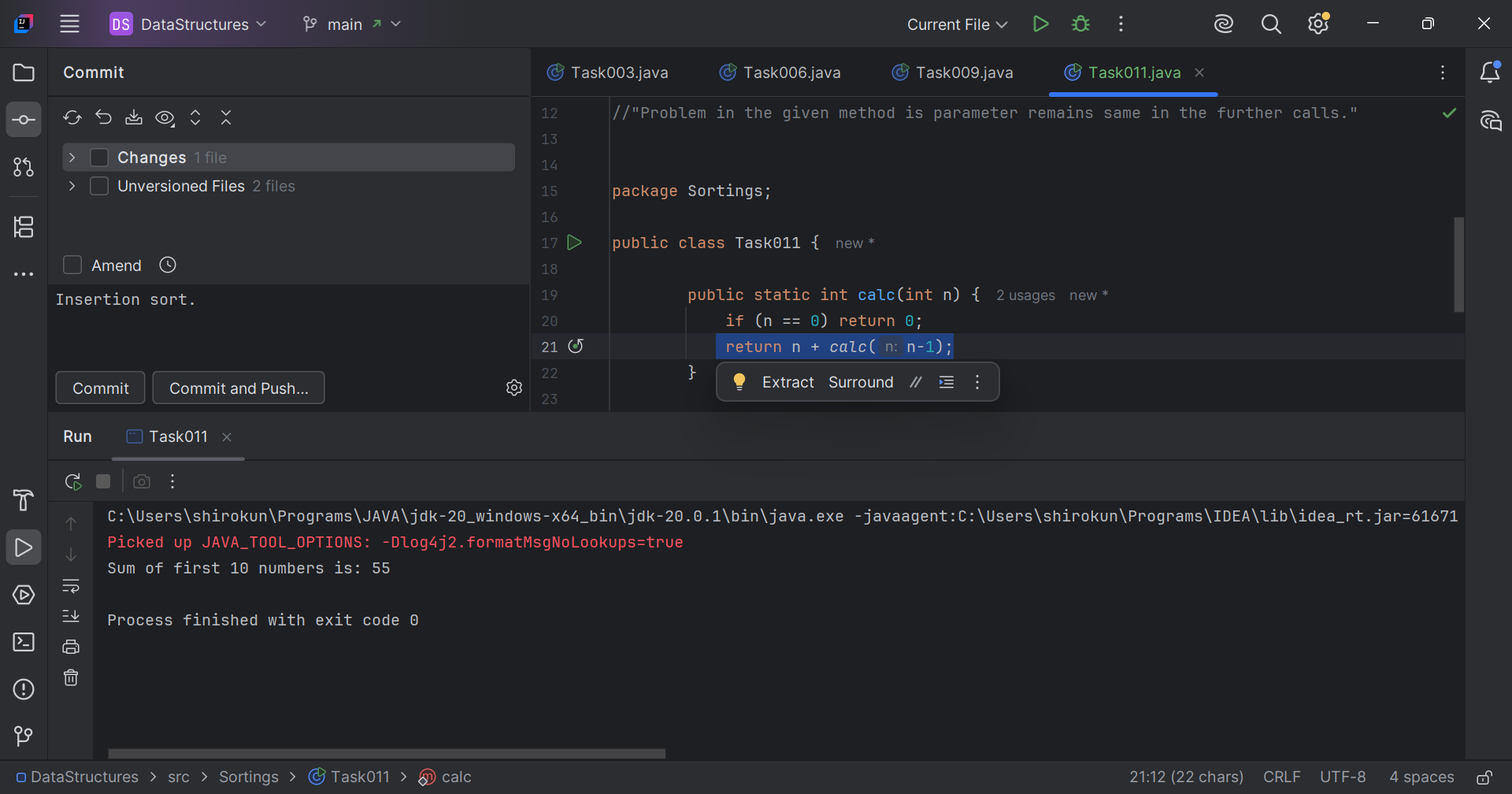
Q10. What are the advantages and disadvantages of Bubble sort?  
Ans. Advantages –

* Simple to implement: Bubble Sort is one of the simplest sorting algorithms to understand and implement.
* Space-efficient: Bubble Sort only requires a single additional memory space for temporary swapping, making it space-efficient.
* Stable sorting algorithm: Bubble Sort is a stable sorting algorithm, meaning that the order of equal elements is preserved.

Disadvantages –

* Slow performance: Bubble Sort has a worst-case and average time complexity of O(n^2), making it inefficient for large datasets.
* Not suitable for large datasets: Due to its slow performance, Bubble Sort is not suitable for sorting large datasets or applications that require fast sorting.
* Not efficient for reverse-sorted arrays: Bubble Sort performs poorly on arrays that are reverse-sorted, as it requires the maximum number of swaps.

/\* Task011:  
This code is going overflow of stack.  
Can you plz help me fix it guys  
  
public class RecLoop {  
 public int calc(int n) {  
 if (n == 0) return 0;  
 return n + calc(n);  
 } \*/  
  
"calc() method is used to calculate sum of first n numbers by recursion."  
"Problem in the given method is parameter remains same in the further calls."  
  
  
package Sortings;  
  
public class Task011 {  
  
 public static int calc(int n) {  
 if (n == 0) return 0;  
 return n + *calc*(n-1);  
 }  
  
 public static void main(String[] args) {  
 int n = 10;  
 int sum = *calc*(10);  
 System.*out*.println("Sum of first "+n+" numbers is: "+sum);  
 }  
}



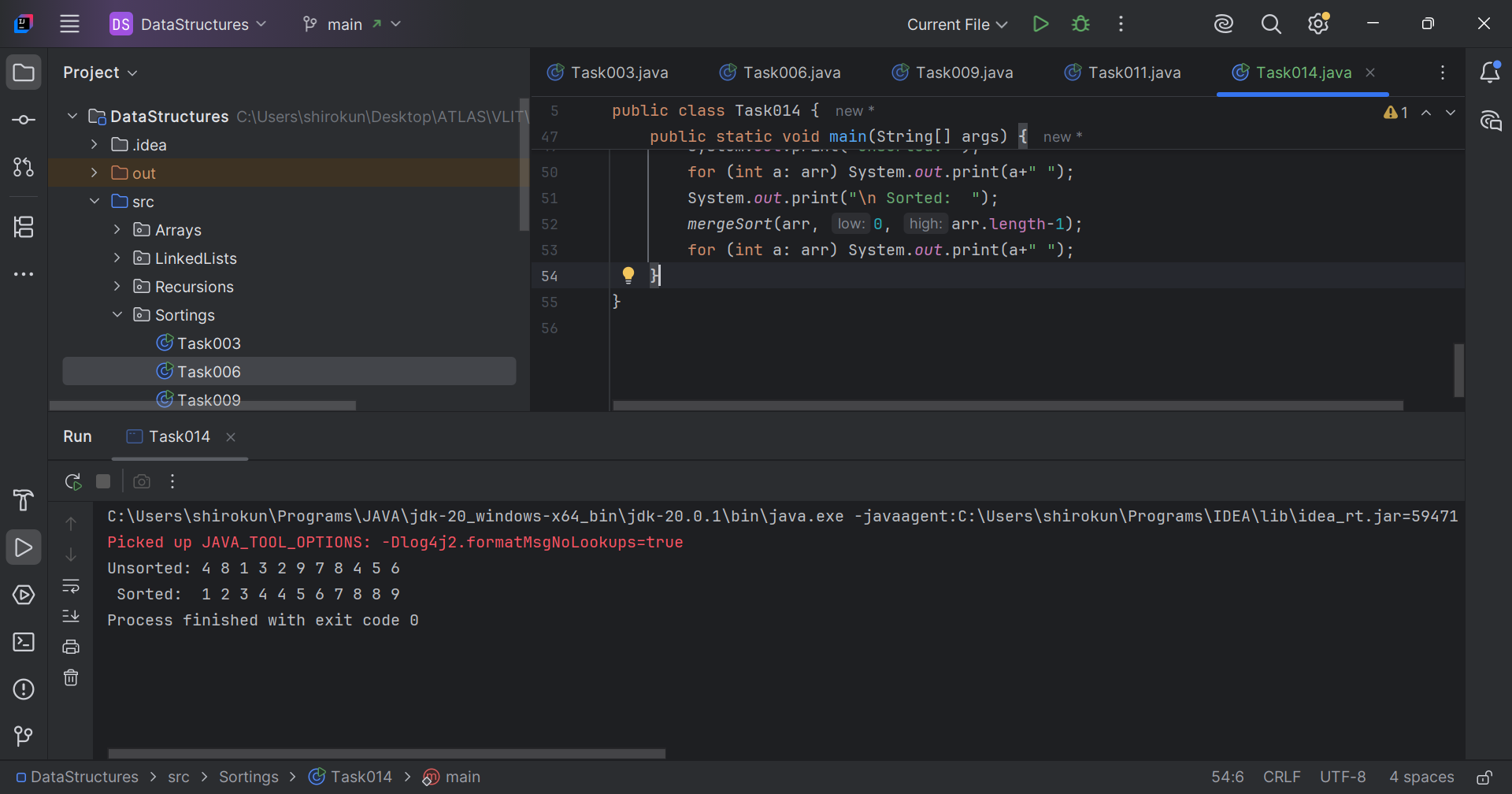
Q12. Write an algorithm for Merge sort.  
Ans. Algorithm –

1. Divide the array: Divide the array into two halves until each half has one element.
2. Merge the halves: Merge the halves in a sorted manner.
3. Repeat the process: Repeat the division and merging process until the entire array is sorted.

Q13. Write pseudocode of Merge sort.  
Ans. Pseudocode –

* MERGE-SORT(arr, low, high)
  + IF low < high
    - mid = (low + high) / 2
    - MERGE-SORT(arr, low, mid)
    - MERGE-SORT(arr, mid + 1, high)
    - MERGE(arr, low, mid, high)
* MERGE(arr, low, mid, high)
  + LEFT = arr[low..mid]
  + RIGHT = arr[mid + 1..high]
  + i = j = 0
  + k = low
  + WHILE i < length(LEFT) AND j < length(RIGHT)
    - IF LEFT[i] <= RIGHT[j]
      * arr[k] = LEFT[i]
      * i = i + 1
    - ELSE
      * arr[k] = RIGHT[j]
      * j = j + 1
    - k = k + 1
  + WHILE i < length(LEFT)
    - arr[k] = LEFT[i]
    - i = i + 1
    - k = k + 1
  + WHILE j < length(RIGHT)
    - arr[k] = RIGHT[j]
    - j = j + 1
    - k = k + 1

// Task014: Merge sort.  
  
package Sortings;  
  
public class Task014 {  
 public static void mergeSort(int[] arr, int low, int high) {  
 if (low < high) {  
 int mid = (low + high) / 2;  
 *mergeSort*(arr, low, mid);  
 *mergeSort*(arr, mid + 1, high);  
 *merge*(arr, low, mid, high);  
 }  
 }  
  
 private static void merge(int[] arr, int low, int mid, int high) {  
 int[] left = new int[mid - low + 1];  
 int[] right = new int[high - mid];  
  
 System.*arraycopy*(arr, low, left, 0, mid - low + 1);  
 System.*arraycopy*(arr, mid + 1, right, 0, high - mid);  
  
 int i = 0, j = 0, k = low;  
 while (i < left.length && j < right.length) {  
 if (left[i] <= right[j]) {  
 arr[k] = left[i];  
 i++;  
 } else {  
 arr[k] = right[j];  
 j++;  
 }  
 k++;  
 }  
  
 while (i < left.length) {  
 arr[k] = left[i];  
 i++;  
 k++;  
 }  
  
 while (j < right.length) {  
 arr[k] = right[j];  
 j++;  
 k++;  
 }  
 }  
  
 public static void main(String[] args) {  
 int[] sortArr, arr = {4, 8, 1, 3, 2, 9, 7, 8, 4, 5, 6};  
 System.*out*.print("Unsorted: ");  
 for (int a: arr) System.*out*.print(a+" ");  
 System.*out*.print("\n Sorted: ");  
 *mergeSort*(arr, 0, arr.length-1);  
 for (int a: arr) System.*out*.print(a+" ");  
 }  
}



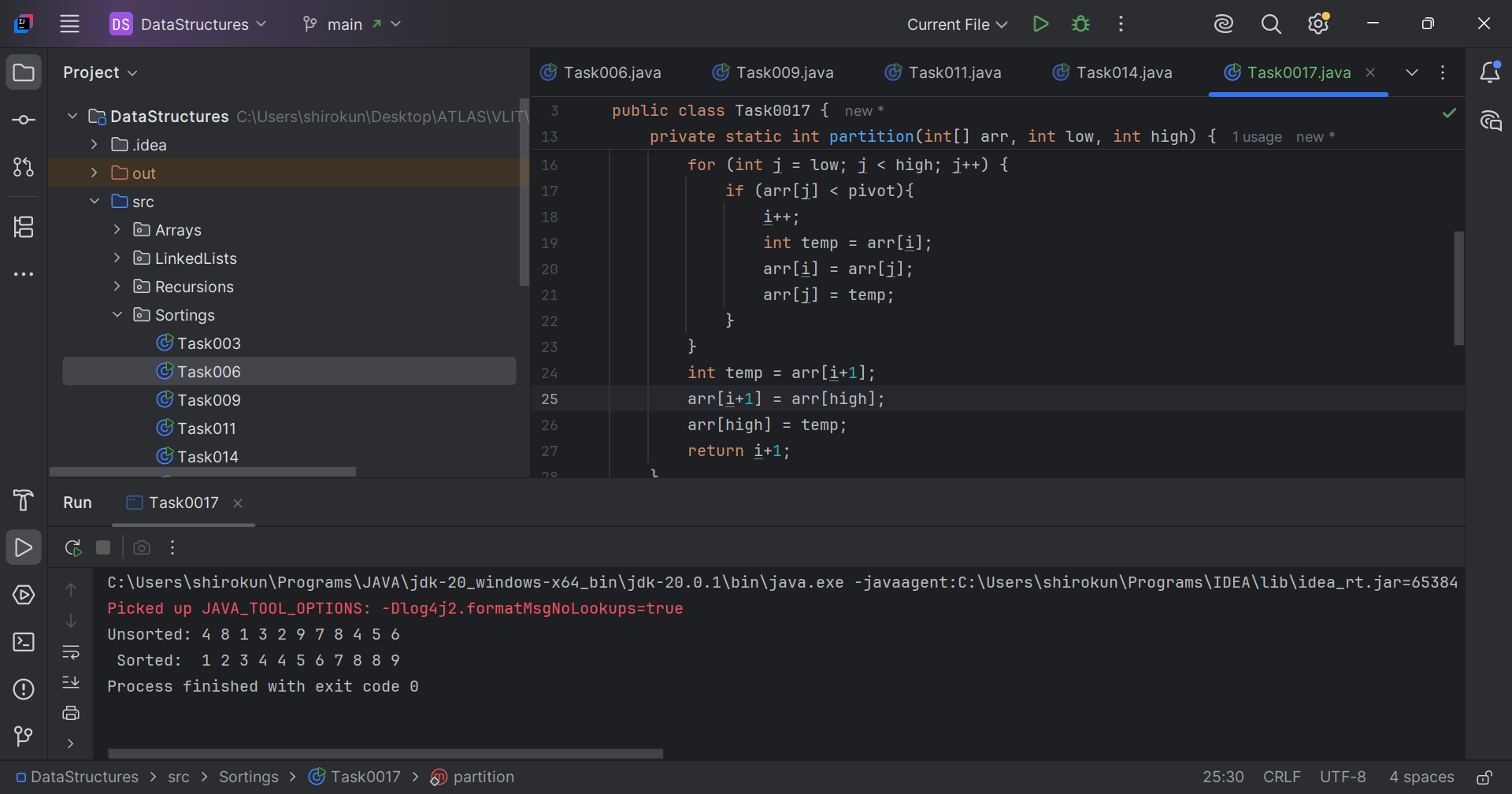
Q15. Write an algorithm for Quick sort.  
Ans. Algorithm –

1. Choose a pivot: Select a pivot element from the array.
2. Partition the array: Partition the array around the pivot element, such that all elements less than the pivot are on the left, and all elements greater than the pivot are on the right.
3. Recursively sort subarrays: Recursively apply the Quick Sort algorithm to the subarrays on the left and right of the pivot element.

Q16. Write pseudocode for Quick sort.  
Ans. Pseudocode –

* QUICKSORT(arr, low, high)
  + IF low < high
    - pivotIndex = PARTITION(arr, low, high)
    - QUICKSORT(arr, low, pivotIndex - 1)
    - QUICKSORT(arr, pivotIndex + 1, high)
* PARTITION(arr, low, high)
  + pivot = arr[high]
  + i = low - 1
  + FOR j FROM low TO high - 1
    - IF arr[j] < pivot
      * i = i + 1
      * SWAP arr[i] WITH arr[j]
  + SWAP arr[i + 1] WITH arr[high]
  + RETURN i + 1

// Task017: Quick sort.  
  
package Sortings;  
  
public class Task017 {  
  
 public static void quickSort(int[] arr, int low, int high){  
 if(low < high){  
 int pivot = *partition*(arr, low, high);  
 *quickSort*(arr, low, pivot-1);  
 *quickSort*(arr, pivot+1, high);  
 }  
 }  
  
 private static int partition(int[] arr, int low, int high) {  
 int pivot = arr[high];  
 int i = low-1;  
 for (int j = low; j < high; j++) {  
 if (arr[j] < pivot){  
 i++;  
 int temp = arr[i];  
 arr[i] = arr[j];  
 arr[j] = temp;  
 }  
 }  
 int temp = arr[i+1];  
 arr[i+1] = arr[high];  
 arr[high] = temp;  
 return i+1;  
 }  
  
 public static void main(String[] args) {  
 int[] arr = {4, 8, 1, 3, 2, 9, 7, 8, 4, 5, 6};  
 System.*out*.print("Unsorted: ");  
 for (int a: arr) System.*out*.print(a+" ");  
 System.*out*.print("\n Sorted: ");  
 *quickSort*(arr, 0, arr.length-1);  
 for (int a: arr) System.*out*.print(a+" ");  
 }  
}



Add-ons

Q1. What is the difference between binary tree and binary search tree?  
Ans.

Structure

* Binary Tree: A tree data structure where each node has at most two children (left and right).
* Binary Search Tree (BST): A Binary Tree with a specific ordering property: all values in the left subtree are less than the node's value, and all values in the right subtree are greater.

Operations

* Binary Tree: Supports basic operations like insertion, deletion, and traversal, but search operations are not efficient.
* Binary Search Tree (BST): Supports efficient search operations (O(log n) average time complexity) due to its ordering property. Also supports insertion, deletion, and traversal.

Q2. In sorted array why do you think binary search tree is best than linear search.  
Ans. Binary Search is better than Linear Search for searching a sorted array. Binary Search offers O(log n) time complexity without the overhead of a tree structure. Linear Search is inefficient with O(n) time complexity.

Q3. Difference between static and dynamic arrays.  
Ans.

Static Arrays

* Fixed size: The size of the array is determined at compile time and cannot be changed.
* Memory allocation: Memory is allocated at compile time.
* Faster access: Static arrays have faster access times since the memory location is known at compile time.

Dynamic Arrays

* Variable size: The size of the array can be determined and changed at runtime.
* Memory allocation: Memory is allocated at runtime.
* More flexible: Dynamic arrays can grow or shrink as needed, making them more flexible than static arrays.

Q4. In BFS, DFS which one is more preferred in terms of shortest path for the unweighted graphs.  
Ans. BFS (Breadth-First Search) is generally preferred for finding the shortest path. This is because BFS explores the graph level by level, starting from the source node, and guarantees that when it reaches a node, it has done so via the shortest path from the source.

Q5.