**Insertion Sort**

1. Create a method insertionSort that uses an insertion sort to arrange an array of double values in ascending order. Create a main method in the class to test the method.
2. An insertion sort is to be used to put the following list of values in sorted ascending order. Show the values as they would appear after each pass of the sort.

| 6 | 2 | 8 | 3 | 1 | 7 | 4 |
| --- | --- | --- | --- | --- | --- | --- |

| Array before: {6,2,8,3,1,7,4,}  Array pass 1: {2,6,8,3,1,7,4,}  Array pass 2: {2,6,8,3,1,7,4,}  Array pass 3: {2,3,6,8,1,7,4,}  Array pass 4: {1,2,3,6,8,7,4,}  Array pass 5: {1,2,3,6,7,8,4,}  Array After: {1,2,3,4,6,7,8,} |
| --- |
| Code:  /\*  \* Program name: TransparentInsertionSort.java  \*  \* By: Lucas Chow (Last edited: 2022-)  \*  \* ICS4U1: Insertion Sort  \*  \* This program uses an insertion sort method to sort an array using insertion sort, except it shows the passes of the array  \*  \*  \* This program also tests the method in the main method  \*  \*  \*/  public class TransparentInsertionSort{  /\*  \* void transparentInsertionSort(int[] inputArr)  \*  \* int[] inputArr -> this is the inputed array  \*  \* This method uses insertion sort to sort the array (see above)  \*  \* void - returns nothing, as arrays are pass-by objects  \*/  public static void transparentInsertionSort(int[] inputArr)  {  int temp;  int pointerValue;  System.out.print("Array before: {");  for (int i = 0; i < inputArr.length; i++)  {  System.out.print(inputArr[i]+",");  }  System.out.println("}");      for (int i = 1; i < inputArr.length; i++)  {  //printing the array for each step  if (i != 1)  {  System.out.print("Array pass "+(i-1)+": {");  for (int x = 0; x < inputArr.length; x++)  {  System.out.print(inputArr[x]+",");  }  System.out.println("}");  }    pointerValue = i;  while (pointerValue > 0 && inputArr[pointerValue-1] > inputArr[pointerValue])  {  temp = inputArr[pointerValue];  inputArr[pointerValue] = inputArr[pointerValue-1];  inputArr[pointerValue-1]=temp;  pointerValue--;  }  }    System.out.print("Array After: {");  for (int i = 0; i < inputArr.length; i++)  {  System.out.print(inputArr[i]+",");  }  System.out.println("}");  }  public static void main(String[] args)  {  //creating test array, from document  int[] testArr = {6,2,8,3,1,7,4};        //performing insertion sort  transparentInsertionSort(testArr);  }  } |

1. Write a program that initializes an array with the names of the planets order by their distances from the sun (Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune, and Pluto) and prints them in that order on one line. The program should then use an insertion sort to arrange the names alphabetically. To trace the progress of the sort, have it print the list after each pass.

| /\*  \* Program name: InsertionSortPlanet.java  \*  \* By: Lucas Chow (Last edited: 2022-10-30)  \*  \* ICS4U1: Insertion Sort  \*  \* This program sorts an array of planets that are ordered by their distance to the sun to  \* alphabetical order using insertion sort  \*  \*  \*/  public class InsertionSortPlanet{  public static void alphabeticalInsertionSort(String[] stringArr)  {  int pointerString;  String temp;  for (int i = 1; i < stringArr.length; i++)  {  pointerString = i;  while (pointerString > 0 && stringArr[pointerString].compareToIgnoreCase(stringArr[pointerString-1])<0)  {  temp = stringArr[pointerString];  stringArr[pointerString]=stringArr[pointerString-1];  stringArr[pointerString-1]=temp;  pointerString--;  }  }  }    public static void main(String[] args)  {  //array of the names of the planets in the solar system (Technically Pluto doesn't qualify as a planet :/ ) in terms of their distance from the sun  String[] planets\_distance\_from\_sun = {"Mercury","Venus","Earth","Mars","Jupiter","Saturn","Neptune","Pluto"};  //precondition of program  System.out.print("Array before: {");  for (int i = 0; i < planets\_distance\_from\_sun.length; i++)  {  System.out.print(planets\_distance\_from\_sun[i]+",");  }  System.out.println("}");    //performing insertion sort  alphabeticalInsertionSort(planets\_distance\_from\_sun);  //post-condition of the program  System.out.print("Array After: {");  for (int i = 0; i < planets\_distance\_from\_sun.length; i++)  {  System.out.print(planets\_distance\_from\_sun[i]+",");  }  System.out.println("}");    }  } |
| --- |

1. The median of an ordered list of numerical values is defined in the following way. If the number of values is odd, the median is the middle value. If the number of values is even, the median is the average of the two middle values. Write a program that first prompts the user for the number of items to be processed, reads that many real values, and then finds their median.

| /\*  \* Program name: FindMedian.java  \*  \* By: Lucas Chow (Last edited: 2022-)  \*  \* ICS4U1: Insertion Sort  \*  \*  \*  \*  \*/  //importing Scanner  import java.util.Scanner;  public class FindMedian{  public static double median(int[] inputArr)  {  double median;  int temp;  int pointerValue;    for (int i = 1; i < inputArr.length; i++)  {  pointerValue = i;  while (pointerValue > 0 && inputArr[pointerValue-1] > inputArr[pointerValue])  {  temp = inputArr[pointerValue];  inputArr[pointerValue] = inputArr[pointerValue-1];  inputArr[pointerValue-1]=temp;  pointerValue--;  }  }  //even case  if (inputArr.length%2 ==0)  {  median = ((double)inputArr[(inputArr.length-1)/2+1]+(double)inputArr[(inputArr.length-1)/2])/2;  }  else  {  median = inputArr[(inputArr.length-1)/2];  }    return median;    }    public static void main(String[] args)  {  Scanner sc = new Scanner(System.in);    int array\_length;  int[] intArray;  System.out.print("Enter a value to be the length of the array: ");  array\_length = sc.nextInt();    //declaring intArray with length array\_length  intArray = new int[array\_length];    for (int i = 0; i < intArray.length; i++)  {  System.out.print("Enter value for index "+(i)+": ");  intArray[i] = sc.nextInt();  }    //outputs the array median  System.out.println("The median of the array is "+median(intArray));    //closing scanner  sc.close();  }  } |
| --- |

1. A sort is said to be *stable* if it always leaves values that are considered to be equal in the same order after the sort. How does the algorithm of a *stable* insertion sort differ from that of an non *stable* insertion sort?

| The key difference for a stable and non state insertion sort is if the insertion sort considers the case of two adjacent identical values, where if it chooses to swap them (while x> 0 && arr[x-1] **>=** arr[x]) if it is a **>=**, it will swap, becoming **un-stable,** if it is just a **>,** it will be a **stable** sort |
| --- |