**Binary Search**

1. Suppose that an array contains the following elements.

| 23 | 27 | 30 | 34 | 41 | 49 | 51 | 55 | 57 | 60 | 67 | 72 | 78 | 83 | 96 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

Trace the execution of the binary search algorithm as it searches for the following values:

array.length = 15

Uses the code:

int binarySearch(int serachValue)

{

int bIndex = 0;

int tIndex = array.length-1;

int mIndex;

index = -1;

while (bIndex <= tIndex && index == -1)

{

mIndex = (bIndex+tIndex)/2;

if (array[mIndex] == searchValue )

{

return mIndex;

}

else if (array[mIndex] > searchValue)

{

bIndex = mIndex + 1;

}

else

{

tIndex = mIndex - 1;

}

}

return index; //index will be -1 at this point

}

| Variables  bIndex (int) = 0 8  tIndex (int) = 14  mIndex (int) = 7 11  searchValue (int) = 72  index (int) = -1 | **Calculations**  tIndex = array.length - 1 = 15 - 1 = 14  mIndex = (tIndex + bIndex)/2 = (14+0)/2 = 7  because array[7] != 72 and 72 > array[7],  therefore bIndex = 8  mIndex = (8+14)/2 = 22/2=11  array[11] == searchValue(int) ->  72 == 72 (true), index = 11, return index;  \*end of method |
| --- | --- |

| Variables  bIndex (int) = 0 4  tIndex (int) = 14 6 4  mIndex (int) = 7 3 5  searchValue (int) = 41  index (int) = -1 | **Calculations**  tIndex = array.length - 1 = 15 - 1 = 14  mIndex = (tIndex + bIndex)/2 = (14+0)/2 = 7  because array[7] != 41 and 41 < array[7],  therefore tIndex = 6  mIndex = (6+0)/2 = 3  array[3] != 41  41 > array[3]  bIndex = 3+ 1 = 4  mIndex = (4+6)/2= 5  array[5] != 41  41 < array[5]  tIndex = 5-1 =4  array[4] == 41,  41 = 41. index = 4,  return 4  \*end of method |
| --- | --- |

| Variables  bIndex (int) = 0 8 10  tIndex (int) = 14 10  mIndex (int) = 7 11 9  searchValue (int) = 62  index (int) = -1 | **Calculations**  tIndex = array.length - 1 = 15 - 1 = 14  mIndex = (tIndex + bIndex)/2 = (14+0)/2 = 7  because array[7] != 62 and 62 > array[7] (55),  therefore bIndex = 8  mIndex = (14+8)/2 = 11  array[11] != 62  62 < array[11] (72)  tIndex = 11-1 = 10  mIndex = (10+8)/2= 9  array[9] != 41  62 > array[9] (60)  bIndex = 9+1=10  mIndex = (10+10)/2 = 10  array[10] != 62  62 < array[10] (67)  bIndex = 10+1 = 11  Because bIndex =11, tIndex = 10, bIndex > tIndex, while loop closes  return (index = -1)  \*end of method |
| --- | --- |

1. Write two methods to search, using the binary search algorithm, for the given number, on the given array. The method should return the location (index) of the item on the array.
   1. Implement the binary search algorithm using simple iteration (loops)
   2. Implement the binary search algorithm using recursion
   3. Test the two methods: write a program that creates an array of 20 integers that sorted in ascending order. Then, it prompts user to enter a number, then searches for the number and outputs its position or indicates if the number is not present in the array.

| //importing scanner  import java.util.\*;  public class BinarySearchAlgorithm {  /\*  \* int binarySearchNonRecursive(int searchValue, int[] inputArray)  \*  \* returns int - the index of the searchValue (-1 if not found)  \*  \* searchValue -> the target value  \*  \* inputArray -> the array to serach  \*  \*This method uses binary search in a looping manner instead of recursion  \*/    public static int binarySearchNonRecursive(int searchValue, int[] inputArray)  {  int top;  int bottom;  int middle;    top = inputArray.length-1;  bottom = 0;  int searchIndex = -1;  while (bottom <= top && searchIndex == -1)  {  middle = (top+bottom)/2;    if (inputArray[middle] == searchValue)  {  searchIndex = middle;  return searchIndex;  }  else if (searchValue > inputArray[middle])  {  bottom = middle+1;  }  else  {  top = middle-1;  }  }    return searchIndex;  }      /\*  \* int binarySearchRecursion(int bottom, int top, int searchValue, int[] inputArray)  \*  \* returns int - the index of the searchValue (-1 if not found)  \*  \* searchValue -> the target value  \*  \* inputArray -> the array to serach  \*  \* bottom -> bottom bound of the binary serach  \*  \* top -> top bound of the binary search  \*  \* This method uses binary search in a looping manner instead of recursion  \*/    public static int binarySearchRecursion(int bottom, int top, int searchValue, int[] inputArray)  {  int middle;  if (bottom > top)  {  return -1;  }  middle = (top+bottom)/2;  if (inputArray[middle] == searchValue)  {  return middle;  }  else  {  if (searchValue > inputArray[middle])  {  return binarySearchRecursion(middle+1, top, searchValue, inputArray);  }  else  {  return binarySearchRecursion(bottom, middle-1, searchValue, inputArray);  }  }  }    /\*  \* void bubbleSort(int[] input Array)  \*  \* returns nothing (Sorts the array, a pass-by object)  \*  \* int[] inputArray -> the array to be sorted  \*  \* This method sorts the array using bubble, o(n^2) comparing following neighbouring elements and corresponding swaps  \*/  public static void bubbleSort(int[] inputArray)  {  int swapValue;  boolean isSorted;  int upperBound;  isSorted = false;  upperBound = inputArray.length-1;    while (upperBound > 0 && !isSorted)  {  isSorted = true;  for (int i = 0; i < upperBound; i++)  {  if (inputArray[i+1] < inputArray[i])  {  isSorted = false;  swapValue = inputArray[i];  inputArray[i] = inputArray[i+1];  inputArray[i+1] = swapValue;  }  }  upperBound--;  }      }  public static void main(String[] args)  {  //declaring scanner  Scanner sc = new Scanner(System.in);  int intInput;  //test array  final int TEST\_ARRAY\_LENGTH = 20;  int[] testArray = new int[TEST\_ARRAY\_LENGTH];        //filling the array with random integers 1-100  for (int i = 0; i < TEST\_ARRAY\_LENGTH; i++)  {  testArray[i] = (int)(Math.random()\*100+1);    }    //sorting with bubble sort  bubbleSort(testArray);      //printing the array  System.out.print("{");  for (int i = 0; i < TEST\_ARRAY\_LENGTH; i++)  {  System.out.print(""+testArray[i]+",");  }  System.out.println("}");    //prompting user input  System.out.print("\nEnter a value to check if it is in the array: ");  intInput = sc.nextInt();    //output  if (binarySearchNonRecursive(intInput, testArray) == -1)  {  System.out.println("\nUsing Binary search with loops, "+intInput+" does not appear");  }  if (binarySearchRecursion(0,testArray.length-1, intInput, testArray) == -1)  {  System.out.println("\nUsing Binary search with recursion, "+intInput+" does not appear");  }  if (binarySearchNonRecursive(intInput,testArray) != -1)  {  System.out.println("\nUsing Binary search with loops, "+intInput+" is at index "+binarySearchNonRecursive(intInput,testArray));  }  if (binarySearchRecursion(0, testArray.length-1, intInput, testArray) != -1)  {  System.out.println("\nUsing Binary search with recursion, "+intInput+" is at index "+binarySearchRecursion(0,testArray.length-1,intInput,testArray));  }    sc.close();    }  } |
| --- |

1. What change would have to be made so that it will search an array that is sorted in descending order?

Simply by reversing the comparison brackets (array[i] > serachValue) ->

(array[i] < searchValue) will imply the data set is sorted in descending

order

1. Rewrite the binary search algorithm so that, if a search is unsuccessful, the method will return the index of the value nearest to item, instead of returning -1. (If there is a tie, return the smaller index.)

| /\*  \* int binarySearchNonRounded(int searchValue, int[] inputArray)  \*  \* returns int - the index of the searchValue (or closest if not found, favoring the lower index)  \*  \* searchValue -> the target value  \*  \* inputArray -> the array to search  \*  \*This method uses binary search in a looping manner instead of recursion, not returning -1 and instead the closest lower index  \*/  public static int binarySearchRounded(int searchValue, int[] inputArray)  {  int top;  int bottom;  int middle;    top = inputArray.length-1;  bottom = 0;  int searchIndex = -1;  while (bottom <= top && searchIndex == -1)  {  middle = (top+bottom)/2;    if (inputArray[middle] == searchValue)  {  searchIndex = middle;  return searchIndex;  }  else if (searchValue > inputArray[middle])  {  bottom = middle+1;  }  else  {  top = middle-1;  }  }    //no value found  if (searchValue > inputArray[inputArray.length-1])  {  return inputArray.length-1;  }  else if (searchValue < inputArray[0])  {  return 0;  }  else  {    if (Math.abs(inputArray[bottom]-searchValue) <= Math.abs(inputArray[top]-searchValue))  {  return bottom;  }  else  {  return top;  }  }    } |
| --- |

1. What is the maximum number of comparisons that might be necessary to perform a binary search on a list containing
   1. 7 items = 3
   2. 3 items = 2
   3. 15 items = 4
   4. 31 items = 5
   5. 63 items = 6
   6. 100 items = 7
   7. 500 items = 9
   8. 1000 items = 10
   9. 10000 items = 13