**Searching Algorithms: Linear Search**

A Searching Algorithm will allow you to identify if a specific value exists within a given array of information. The Linear Sort is the simpler of the 2 we will be looking at. Either build your own array for these searches, consisting of 20 elements or use the following.**int** OriginalArray[] = {4,10,6,1,8,10,9,12,14,6,15,6,7,10,8,2,7,3,9,1};

1. Please develop a Linear Search Method in your eclipse file, and then copy and paste it into the box below. Identify the amount of instances of the selected value which exist within the array.

|  |
| --- |
| static void binarySearch(int[] a) {  /// sorting  int itemToInsert, j;  boolean reset;    for(int k = 1; k < a.length; k++) {  itemToInsert = a[k];  j = k-1;  reset = false;    while((j >= 0) && !reset) {  if (itemToInsert < a[j] ) {  a[j + 1] = a[j];  j--;  if(j == -1)  a[0] = itemToInsert;  }  else {  reset = true;  a[j + 1] = itemToInsert;  }  }  }    /// binary search with sorted array    System.***out***.println("\n\n(Binary) Input a number to search: ");  int numIn = *input*.nextInt();    int low = 0;  int high = a.length;    while(low+1 < high) {  int temp = (low + high)/2; // move index to middle of array    if(a[temp] > numIn) { // if the currently selected element is greater than the input number then set the index to the high variable, otherwise set it to the low variable  high = temp;  }  else {  low = temp;  }  }    if(a[low] == numIn) { // if a exact match is found, print its index  System.***out***.printf("The first instance of the number is at index %d or position %d\n", low, low+1);  }  else {  System.***out***.println("\nThat number is not in the array");  }  }      public static void main(String[] args) {  int[] newOriginalArray = {4,10,6,1,8,10,9,12,14,6,15,6,7,10,8,2,7,3,9,1};    *binarySearch*(newOriginalArray);  } |

1. Please add developer comments to the various elements of your search. These comments should briefly outline what each line of code is for, and what it accomplishes.
2. In the box below, please provide a written description of how this search moves through an array identifying the selected values. This Description should highlight the process, it does not need to outline every single adjustment that is made.

|  |
| --- |
| An integer value is inputted and then compared with every element of the array procedurally, if a match is found then it prints the index and adds one to the identified indexes counter. At the end it prints the amount of identified indexes |

1. Can you think of a situation when this searching method would be the most logical method to use, please explain why?

|  |
| --- |
| When used with strings, this can be helpful for identifying a name in a list of names to verify if that name is on the list. An example of this would be verifying login credentials |

**Searching Algorithms: Binary Search**

The Binary Search is slightly more complicated, as it will not work unless it has a fully sorted array. Either build your own array for this search, consisting of 20 elements or use the following.**int** OriginalArray[] = {4,10,6,1,8,10,9,12,14,6,15,6,7,10,8,2,7,3,9,1};

1. Please develop a Binary Search Method in your eclipse file, and then copy and paste it into the box below. Identify the amount of instances of the selected value which exist within the array.

|  |
| --- |
| static void binarySearch(int[] a) {  /// sorting  int itemToInsert, j;  boolean reset;    for(int k = 1; k < a.length; k++) {  itemToInsert = a[k];  j = k-1;  reset = false;    while((j >= 0) && !reset) {  if (itemToInsert < a[j] ) {  a[j + 1] = a[j];  j--;  if(j == -1)  a[0] = itemToInsert;  }  else {  reset = true;  a[j + 1] = itemToInsert;  }  }  }    /// binary search with sorted array    System.***out***.println("\n\n(Binary) Input a number to search: ");  int numIn = *input*.nextInt();    int low = 0;  int high = a.length;  int m = 0;  int n = 0;    do {  m = 0;  int temp=0;  while(low+1 < high) {    temp = (low + high)/2;// move index to middle of array    if(a[temp] > numIn) { // if the currently selected element is greater than the input number then set the index to the high variable, otherwise set it to the low variable  high = temp;  }  else {  low = temp;  }  }    if(a[low] == numIn) { // if a exact match is found, print its index  System.***out***.printf("The number is at index %d or position %d\n", low, low+1);    //System.out.printf("The first instance of the number is at index %d or position %d\n", low, low+1);  m++;  n++;  if(numIn != 0) { // need some way to add in an element to replace and eliminate a found element when identifying multiple instances of the element  a[low] = 0;  }  else {  a[low] = 1;  }  for(int k = 1; k < a.length; k++) { // re-sort  itemToInsert = a[k];  j = k-1;  reset = false;    while((j >= 0) && !reset) {  if (itemToInsert < a[j] ) {  a[j + 1] = a[j];  j--;  if(j == -1)  a[0] = itemToInsert;  }  else {  reset = true;  a[j + 1] = itemToInsert;  }  }  }  }  else {  System.***out***.println("\nThat number is not in the array");  }  }while(m == 1);    System.***out***.printf("There is %d instances of this number in the array", n);  }      public static void main(String[] args) {  int[] newOriginalArray = {4,10,6,1,8,10,9,12,14,6,15,6,7,10,8,2,7,3,9,1};    *binarySearch*(newOriginalArray);  } |

1. Please add developer comments to the various elements of your search. These comments should briefly outline what each line of code is for, and what it accomplishes.
2. In the box below, please provide a written description of how this search moves through an array sorting it.   
     
   In order to identify multiple instances of the selected value, what did you need to add to the binary search?

|  |
| --- |
| I decided to remove the first instance of the element after it is identified and reassign it to an element that isn’t being searched for and re-sort it to be linear and re-search with the binary search |

1. Can you think of a situation when this searching method would be the most logical method to use, please explain why?

|  |
| --- |
| Binary search works by sorting an array linearly and then comparing the number being search to half of the array and eliminating one half of it and then doing the same process with the other half until the number is identified. In most cases this can be very efficient because It eliminates a large range of numbers in the first search and the proceeding searches which allows for few searches to identify the number. |