**Lesson 4 – Recursion II**

***Lesson Objectives:***

* Describe the difference between head recursion and tail recursion.
* Write a function that employs head recursion.
* Write a function that employs tail recursion.
* Explain the utility of using helper methods with recursion.
* Describe how to conduct a recursive binary search.

***Readings:***

* Read Chapter 3, Sections 2-5.

***Lesson:***

* ***Recursively processing an attribute array:***
  + We can use private helper methods to provide a cleaner signature for client code.
    - We do this when we want to use recursion, but there is a difference between the parameters needed for the recursion and the parameters needed for the client code.
  + Though we can recursively process an array that is passed as an argument, we can also recursively process an array that is a class attribute:

// An array attribute of the class

private int[] intArray;

// Constructor creating the array attribute

public ConstructorName() {

intArray = { 23, 34, 45, 56, 67, 78 };

}

// calculate the sum of the elements of an array using head recursion

private static int sumByHead(int index)

{

if (index == intArray.length - 1)

return intArray[index];

else

return intArray[index] + sumByHead(index + 1);

}

public static int sumByHead()

{

return sumByHead(0);

}

// calculate the sum of the elements of an array using tail recursion

private static int sumByTail(int index, int s)

{

if (index == intArray.length - 1)

return s + intArray[index];

else

return sumByTail(index + 1, s + intArray[index]);

}

public static int sumByTail()

{

return sumByTail(0, 0);

}

* + In this example, we are assuming that all locations in the array contain relevant integers. Also, the array to be processed is passed as an argument to the method.
* ***In-class Exercise:***
  + Write a method that uses head recursion to find the maximum value in an array.
  + Write a method that uses tail recursion to find the maximum value in an array.

private int findMaxHR(int index)

{

if (index == intArray.length - 1)

return intArray[index];

else {

int maxFromRest = findMaxHR(index + 1);

if (maxFromRest > intArray[index])

return maxFromRest;

else

return intArray[index];

}

}

public static double findMaxHR()

{

return findMaxHR(values, 0);

}

private static double findMaxTR(int index, int maxSoFar)

{

if (index == intArray.length)

return maxSoFar;

else {

if (intArray[index] > maxSoFar)

return findMaxTR(index + 1, values[index]);

else

return findMaxTR(values, index + 1, maxSoFar);

}

}

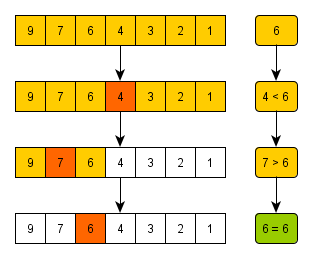
public static double findMaxTR(double[] values)

{

return findMaxTR(values, 0, 0);

}

* ***Recursively Searching a Sorted Array using Binary Search:***
  + Suppose that you have a sorted array lsnArray of integers and you want to find a specific integer. You could construct an iterative solution without too much difficulty, but instead consider a recursive formulation that is actually more efficient than examining every item:



// A recursive binary search helper function. It returns

// location of x in given array arr[l..r] is present,

// otherwise -1

private static int binarySearch(int[] arr, int l, int r, int x)

{

if (r >= l) {

int mid = l + (r - l) / 2;

// If the element is present at the middle

// itself

if (arr[mid] == x)

return mid;

// If element is smaller than mid, then

// it can only be present in left subarray

if (arr[mid] > x)

return binarySearch(arr, l, mid - 1, x);

// Else the element can only be present

// in right subarray

return binarySearch(arr, mid + 1, r, x);

}

// We reach here when element is not

// present in array

return -1;

}

// The public method for conducting a binary

// search of the array arr.

public static int binarySearch(int[] arr, int x)

{

return binarySearch(arr, x);

}