Data Structures and Algorithms – Week 2 Code Samples

Lecturer: Arran Stewart

This zip file contains code samples we will refer to during week 2.

To understand the code samples well, you should make use of the textbook, which is *Data Structures and Problem Solving Using Java* (4th edition) by Mark Allen Weiss.

The outline for the Data Structures and Algorithms course contains suggested readings from the textbook that will help you understand the code here.

If there are sections of the textbook you particularly should read for some code listings, I will mention them here.

Four Sorting Algorithms

```java


/**
 * Utility class offering several sorting algorithms
 */

public class SortingAlgorithms {
    /**
     * Execute the insertion sort algorithm sorting the argument
     * array. There is no return since the parameter is mutated.
     *
     * @param arr the array of short integers to be sorted
     **/
    public static void insertionSort(short[] arr) {
        for (int j = 1; j < arr.length; j++) {
            short key = arr[j];
            int i = j - 1;
            while (i >= 0 && arr[i] > key) {
                arr[(i + 1)] = arr[i];
                i = i - 1;
            }
            arr[i + 1] = key;
        }
    }

    /**
     * Execute the selection sort algorithm on an argument array.
     * There is no return as the parameter is mutated.
     * @param arr   the array of short (numbers) to be sorted
     **/
    public static void selectionSort(short[] arr) {
        //build the sorted portion from 0 upwards
        for (int i = 0; i < arr.length - 1; i++) {
            //initialise the minimum value and its position
            short min = arr[i];
            int minpos = i;
            //search the unsorted part for its smallest element
            for (int j = i + 1; j < arr.length; j++) {
                if (arr[j] < min) {
                    min = arr[j];
                    minpos = j;
                }
            }
            if (i != minpos) {  // swap min into place if necessary
                short temp = arr[i]; //copy
                arr[i] = arr[minpos]; //transfer
                arr[minpos] = temp; //replace
            }
        }
```

```java
    }






    /**
     * Execute the merge sort algorithm sorting the argument array.
     * There is no return as the parameter is to be mutated.
     * @param arr the array of short integers to be sorted
     **/
    public static void mergeSort(short[] arr) {
        mergeSort(arr, 0, arr.length-1);
    }

    private static void mergeSort(short[] arr, int l, int r) {
        if (l < r) {
            int mid = (l + r) / 2;
            mergeSort(arr, l, mid);
            mergeSort(arr, mid + 1, r);
            merge(arr, l, mid, r);
        }
    }

    /**
     * Merge two parts of array arr from l to mid and mid+1 to r inclusive
     * @param arr the array containing the portions for merging
     * @param l left index, used for portion 1 from l to mid
     * @param mid mid index, user for portion 2 from mid+1 to r
     * @param r right index
     */
    private static void merge(short[] arr, int l, int mid, int r) {
        int lsize = mid - l + 1;
        int rsize = r - mid;
        short[] left = new short[lsize];
        short[] right = new short[rsize];

        for (int i = 0; i < lsize; i++) {
            left[i] = arr[l + i];
        }
        for (int j = 0; j < rsize; j++) {
            right[j] = arr[mid + 1 + j];
        }
        int i = 0;
        int j = 0;
        int k = l;
        while (i < lsize && j < rsize) {
            if (left[i] < right[j]) {
                arr[k++] = left[i++];
            } else {
```

```java
                arr[k++] = right[j++];
            }
        }
        while ( i < lsize ) { // Copy rest of first half
            arr[k++] = left[i++];
        }
        while( j < rsize ) { // Copy rest of second half
            arr[k++] = right[j++];
        }
    }

    /**
     * Execute the quicksort algorithm sorting the argument array.
     * There is no return as the parameter is to be mutated.
     * @param arr the array of short integers to be sorted
     **/
    public static void quickSort(short[] arr) {
        quickSort(arr, 0, arr.length - 1);
    }

    /**
     *Overloads the quickSort method with parameters to set the range to be sor
     **/
    private static void quickSort(short[] arr, int p, int r) {
        if (p < r) {
            int q = partition(arr, p, r);
            quickSort(arr, p, q - 1);
            quickSort(arr, q + 1, r);
        }
    }

    /**
     *A private method to partition the array arr,
     * between the indices start and finish inclusive
     * inclusive. The method selects the element arr[r] as the pivot.
     *
     * @param arr the array to be sorted, which is mutated by the method
     *@param p the lower index of the range to be partitioned
     *@param r the upper index of the range to be paritioned
     *@return the index of the point of partition
     **/
    public static int partition(short[] arr, int start, int finish) {
        short fence = arr[start]; //get the pivot value
        int left = start+1;
        int right = finish;
        while (right >= left) {
            while (left <= right && arr[left] <= fence)
                left++;
            while (right >= left && arr[right] >= fence)
                right--;
            if (right > left) {
```

```java
                    short swap = arr[left];
                    arr[left] = arr[right];
                    arr[right] = swap;
                }
            }
            arr[start] = arr[right];
            arr[right] = fence;

            return right; //return position of the fence
        }




}
```

Three Search Algorithms

```java
/**
 * Three search algorithms
 * Sequential and Step with
 * Binary search from Weiss Fig 5.11
 */
public class SearchAlgorithms {

  public static int NOT_FOUND = -1;
  public static boolean DEMO = true; //true to print debug statements

  /**
   * search for an item in an array this is the slowest search,
   * O(N) but also the simplest
   *
   * @param a array to be searched, assumed to be sorted
   * @param key item being searched for
   * @return index of key in a if key is found, and NOT_FOUND otherwise
   */
  public static int SequentialSearch(int[] a, int key) {
    for (int i = 0; i < a.length; i++) {
      if (DEMO) { System.out.println("Testing position "+i); }
      if (a[i] == key) { // found it!
        return i;
      }
    }
    return NOT_FOUND; // if we get here, the item was not found
  }

  /**
   * step search for an item in an array
   * this is slightly faster than sequential search but still O(N)
   *
   * @param a array to be searched, assumed to be sorted
```

```java
     * @param key item being searched for
     * @param step step size for moving through array
     * @return index of key in a if key is found, and NOT_FOUND otherwise
     */
    public static int StepSearch(int[] a, int key, int step) {
      int i = 0;
      while (i < a.length) {
        if (DEMO) { System.out.println("Testing position "+i); }
        if (a[i] == key) { // found it
          return i;
        }
        if (a[i] < key) { // flip forwards
          i = Math.min(i + step, a.length - 1);
        } else { // search back through the block
          for (int j = i - 1; j > i - step; j--) {
            if (DEMO) { System.out.println("Testing position "+j); }
            if (a[j] == key) {
              return j;
            }
          }
          return NOT_FOUND;
        }
      }
      return NOT_FOUND; // if we get here, the item was not found
    }

    /**
     * Binary search to find key in array a is O(log N)
     *
     * @param a array to be searched, assumed to be sorted
     * @param key item being searched for
     * @return index of key in a if key is found, and NOT_FOUND otherwise
     */
    public static int BinarySearch(int[] a, int key) {
      int low = 0;
      int high = a.length - 1;
      int mid;

      while (low <= high) {
        mid = (low + high) / 2;
        // more generally ( a[ mid ].compareTo( x ) < 0 )
        if (DEMO) { System.out.println("Testing position "+mid); }
        if (a[mid] < key) { // continue to search lower part
          low = mid + 1;
        } else if (a[mid] > key) { // continue to search upper part
          high = mid - 1;
        } else { // we've found it
          return mid;
        }
      }
      // if we get here, the item was not found
```

```
        return NOT_FOUND; // NOT_FOUND = -1
    }

    /**
     * test code
     */
    public static void main(String[] args) {
        //some test cases to see binary search at work
        int[] a = new int[] { 11, 102, 223, 254, 265, 306, 367, 388, 399, 1000 };
        int[] testkeys = new int[] { 265, 388, 1000, 200 };
        for (int key : testkeys) {
            if (DEMO) { System.out.println("Sequential Search for "+key); }
            int res1 = SequentialSearch(a, key);
            if (DEMO) { System.out.println("Step Search for "+key); }
            int res2 = StepSearch(a, key, 3);
            if (DEMO) { System.out.println("Binary Search for "+key); }
            int res3 = BinarySearch(a, key);
            System.out.println("Found key = " + key + " at position " +
                    res1 + "," + res2 + "," + res3);
        }
    }

}
```