CMP 338 (Fall 2012)

Exam 2, 11/15/12

Name (sign)

Name (print)

email

Question	Score
1	
2	
3	
4	
5	
6	
7	
8	
9	
Extra Credit	

Question 1

For each of the sorting methods below give a) its asymptotic worst-case cost (in comparisons or array accesses, as appropriate) as a function of the size of its input n, b) its average-case cost, c) the amount of extra space it requires, and d) whether or not it is stable.

SelectionSort: **`sort(Sequence<Item> seq)`**

a) **c)**

b) **d)**

InsertionSort: **`sort(Item[] a)`**

a) **c)**

b) **d)**

MergeSort: **sort(Item[] a)**

a) **c)**

b) **d)**

QuickSort: **`sort(Item[] a)`**

a) **c)**

b) **d)**

HeapSort: **`sort(Sequence<Item> seq)`**

a) **c)**

b) **d)**

Question 2

Complete the method below to perform an ***insertion sort*** of and array of items.

```
@Override public void sort (Item[] a) {
```

```
}
```

Question 3

Complete the methods below to perform a **merge sort** of an array of items.

```
/**
 * Sort a given region of an array.
 * Divide the region into two sub region.
 * Sort each sub-region recursively.
 * Merge the two sorted sub-regions.
 *
 * @param a  is the array containing the region.
 * @param lo is the index of the first element of the region.
 * @param hi is the index of the last element of the region.
 */
private void sort (Item[] a, int lo, int hi) {









}
/**
 * Merge two sorted (adjacent) sub-regions of a region.
 * If items are equal, give preference to items from the first sub-region
 *
 * @param a   is the array containing the region.
 * @param lo  is the index of the first element of the first sub-region.
 * @param mid is the index of the last element of the first sub-region.
 * @param hi  is the index of the last element of the second sub-region.
 */
private void merge (Item[] a, int lo, int mid, int hi){













}
```

Question 4

What result would be returned by a call to the **`QuickSort`** method

```
partition(a, 0, 15);
```

on the array **`a`** given below?

5	2	9	8	4	1	6	8	7	4	6	3	2	9	8	1

What would **`a`** look like after the call?

Question 5

Complete the method below to implement **quick sort**. (Your code may call any of the helper methods that we covered in class.)

```
@Override public void sort (Item[] a) {
    sort(a, 0, a.length-1);
}

/**
 * Sort a given region of an array.
 * Pick a random element of the region to use as a pivot.
 * Call partition to divide the region into sub-regions.
 * Find p such that
 *    a[lo..p) <= pivot = a[p] <= a(p..hi]
 * Sort the regions ( a[lo..p) and a(p..hi] )recursively.
 *
 * @param a  is the array containing the region.
 * @param lo is the index of the first element of the region.
 * @param hi is the index of the last element of the region.
 */
private void sort (Item[] a, int lo, int hi) {

}
```

Question 7

Complete the method below to sort an array of small integers using **counting sort.**

```
 @Override public void sort (Item[] a) {
    Item[] aux = (Item[]) new Object[a.length];
    sort(a, 0, a.length-1, aux); }

/**
 * Sort an array of Item's with small non-negative Integer keys
 * (0 <= key(items[i]) < radix).
 * @param items the array to be sorted.
 * @param radix an upper bound on the keys.
 * @return an array telling where the buckets of each size end. */
public int[] sort (Item[] items, int lo, int hi, Item[] aux) {
    // tabulate the histogram of keys
    int[] count = new int[radix+1];
    for




    // count[i] is the number of Item's with key = i-1;
    // integrate the histogram
    count[0] = lo;
    for




    // count[i] is the number of items with key < i
    // move the items to their sorted position in a new array
    // count[i] is the position of the first item with key == i
    for




    // count[i] is the number of items with key <= i
    // copy the items back to the input array
    for



    return count;
}
```

Question 8

Complete the following helper methods of **`TreeHeapPriorityQueue`**.

```
/** Reestablish the heap property
 * by comparing a given child with its parent.
 * @param n the given child. */
private void swim (Node n) {

}

/** Reestablish the heap property
 * by comparing a given parent with the lesser of its children.
 * @param p the given parent. */
private void sink (Node p) {

}
```

Question 9

Complete the following methods of **`MSDRadixSort`**.

```
@Override public void sort (String[] a) {
    String[] aux = new String[a.length];
    sort(a, 0, a.length-1, 0, aux);
}
/** Sort a given region of Strings that share a common prefix.
 * @param a   is the array containing the given region.
 * @param lo  is the index of the first String in the region.
 * @param hi  is the index of the last String in the region.
 * @param d   is the length of the common prefix.
 * @param aux is a scratch array.
 */
protected void sort(String[] a, int lo, int hi, int d, String[] aux){






























}
```

Extra Credit

Describe, in a few, short, legible, English sentences, how to efficiently sort a million thirty-two bit integers.

Question 6

Given the initial heap structure depicted below. What would be the result of executing the following priority queue operations? Draw the resulting heap.

add(17);
add(34);
removeMin();
add(23);