CMP 338 (Fall 2011)

Exam 1, 10/6/11

Name (sign) _____

Name (print) _____

email _____

Question	Score
1	13
2	16
3	13
4	13
5	13
6	11
7	2
8	6
9	4
10	9
TOTAL	100

1 a) The signature of a Java method to sort an array of **doubles** into ascending order is provided below. Complete the method to implement **Insertion Sort**. Do ***not*** call any library methods. You *may* use your own helper methods, provided you also write them.

```
static void insertionSort(double[] a) {

   for (int i=0; i<a.length; i++) {
      for (int j=i; 0<j && a[j-1] < a[j]; j--)
         exch(a, i, j);

}

static void exch(double[] a, int i, int j) {

   double tmp = a[i];
   a[i] = a[j];
   a[j] = tmp;

}
```

1 b) In the worst case, how many comparisons does insertion sort require to sort N items? (Use tilda notation. If you know the leading constant, provide it. Otherwise, use "c".) $\sim N^2/2$

1 c) Briefly characterize the worst case input for insertion sort?

a is already sorted in descending order.

1 d) Under what circumstances does insertion sort provide acceptable performance?

1) **a** is small.
2) **a** is almost sorted.

2 a) The signature of a Java method to sort an array of **`doubles`** into ascending order is provided below. Complete the method to implement **Merge Sort**. Do ***not*** call any library methods. You *may* use your own helper methods, provided you also write them. You may also call the method **`merge`** assuming that it will do what it is supposed to do.

```
static void mergeSort(double[] a) {

   mergeSort(a, 0, a.length-1);

}

static void mergeSort(double[] a, int lo, int hi) {

   if (hi <= lo) return;
   int mid = (hi+lo)/2;
   mergeSort(a, lo, mid);
   mergeSort(a, mid+1, hi);
   merge(a, lo, mid, hi);

}
```

2 b) In the worst case, how many comparisons does merge sort require to sort N items? (Use tilda notation. If you know the leading constant, provide it. Otherwise, use "c".) **~ N lg N**

2 c) How much extra space does mergesort require? **~ N**

2 d) Write the signature for the **`merge`** method.

```
static void merge(double[] a, int lo, int mid, int hi)
```

2 e) Briefly, what is the **`merge`** method supposed to do?

Combine two sorted sub arrays into a sorted array.

3 a) The signature of a Java method to sort an array of **doubles** into ascending order is provided below. Complete the method to implement **Quicksort**. You may use methods of the **Random** library, but no other library methods. You *may* use your own helper methods, provided you also write them. You may also assume that the **partition** method does what it is supposed to do.

```
static void quicksort(double[] a) {

   quicksort(double[] a, 0, a.length-1);

}

static void quicksort(double[] a, int lo, int hi) {

   if (hi <= lo) return;
   exch(a, lo, Random.uniform(lo, hi));
   int j = partition(a, lo, hi);
   quicksort(a, lo, j-1);
   quicksort(a, j+1, hi);

}
```

3 b) In the worst case, how many comparisons does quicksort require to sort N items? (Use tilda notation. If you know the leading constant, provide it. Otherwise, use "c".) $\sim N^2/2$

3 c) Give three reasons why you might ***not*** want to use quicksort to sort a list of Items.

1) Quicksort is unstabile (initial ordering of equal objects is lost).
2) If the input array is known to be nearly sorted.
3) If the sort is called very often on very small arrays.
4) If the worst-case behavior could be catastrophic

4 a) The signature of a Java method to sort an array of **`doubles`** into ascending order is provided below. Complete the method to implement **Heap Sort**. You *may* call the methods (and constructor) of a **MinPQ** priority queue library. You *may* use your own helper methods, provided you also write them.

```
static void heapSort(double[] a) {

   MinPQ pq = new MinPQ();

   for (int i=0; i<a.length; i++)
      pq.insert(a[i]);

   for (int i=0; i<a.length; i++)
      a[i] = pq.deleteMin();

}
```

4 b) In the worst case, how many comparisons does heap sort require to sort N items? (Use tilda notation. If you know the leading constant, provide it. Otherwise, use "c".) **~ cN lg N**

4 c) How much extra space is required by (the array version of) heap sort?

None (~ c)

4 d) In the worst case, how many comparisons are required to remove the smallest item from a **`MinPQ`** (and restore its heap) containing N items?

~ 2 lg N

5 a) The signature of a Java method to return the **k**-th smallest entry in an array of N **doubles** is provided below. Complete the method to return its result using (on average) ~ c N comparisons. Do ***not*** call any library methods. You *may* use your own helper methods, provided you also write them. You may also assume that the **partition** method does what it is supposed to do.

```
static ~~void~~ int select(double[] a, int k) {

   return select(a, 0, a.length, k);

}

static int select(double[] a, int lo, int hi, int k){

   if (hi == lo) return k;
   int j = partition(a, lo, hi);
   if (j < k) return select(a, j+1, hi, k);
   if (k < j) return select(a, lo, j-1, k);
   return k;

}
```

5 b) In the worst case, how many comparisons does your **select** method require to ~~sort~~ select the k-th of N items? (Use tilda notation. If you know the leading constant, provide it. Otherwise, use "c".) **~ $N^2/2$**

5 c) Describe a worst case input for your **select** method.

k=0 and a is already sorted in *descending* order.

5 d) In the worst case, how many comparisons are required by the best possible implementation of the **select** method? **~ cN**

6) The signature of a Java method is provided below. It takes two arrays and an `int` as parameters. The first array is sorted. Implement the method to look up the third parameter in the first array and return the corresponding value from the second array. If the arrays have N elements, your method may use ~ c lg N comparisons. Do ***not*** call any library methods. You *may* use your own helper methods, provided you also write them.

```
double getWeight(int[] sn, double[] weight, int n) {

   int j = binarySearch(sn, 0, sn.length-1, n);
   assert sn[j] == n;
   return weight[j];

}

int binarySearch(int[] a, int lo, int hi, int v) {

   if (hi < lo) return lo;
   int mid = (hi+lo)/2;
   if (a[mid] < v)
      return binarySearch(a, mid+1, hi);
   if (v < a[mid])
      return binarySearch(a, lo, mid-1);
   return mid;

}
```

6 a) How much extra space is required by shell sort? None (~ c).

6 b) Under what circumstances would one ***not*** want to use shell sort?

1. A stable sort is needed.
2. The input array is really, really big.
3. The sort method is going to be called many, many times.

7) Under what circumstances is bubblesort acceptable? Never.

8 a) What does it mean for a sort to be ***stable***?

Items that are equal (wrt the sorting criterion) are not reordered by the sort.

8 b) Which of the sorts we have studied are stable? insertion and merge sorts

8 c) Under what circumstances would one want to use a stable sort?

When the input array is already sorted wrt a different criterion.

9 a) What does Java's **`Comparable`** interface enable?

Allows instances of a class to be sorted wrt some "natural" order.

9 b) What does Java's **`Comparator`** interface enable?

Allows instances of a class to be sorted wrt multiple orderings.

10 a) Complete the following method to determine the number of distinct integers in an array. (You may call library methods.)

```
static int unique(int[] a) {

   if (0 == a.length) return 0;
   Arrays.sort(a);
   int count = 1;
   for (int i=1; i<a.length; i++)
      if (a[i-1] <= a[i])
         count++;
   return count;

}
```

10 b) What is your method's expected asymptotic running time? ~ cN lg N