

CMSI 282 Problem Set #2
Due February 17, 2009

- 1) Compute each of the following... by hand! For each one, indicate your method and show all intermediate results:
- 5^{13}
 - 3^{14}
 - $1206^{1/2}$ (the square of your result must be within one thousandth of 1206)
 - $18^{1/3}$ (the cube of your result must be within one thousandth of 18)
 - Among the following numbers, which is the 31st (thirty-first) smallest?

41	161	145	59	159	67	212	225	155	82
286	71	98	18	173	136	271	210	270	73
48	178	202	27	139	86	46	195	174	77
52	158	94	144	100	88	23	254	223	179
121	201	214	14	215	56	17	272	165	74
104	156	199	197	157	62	153	256	55	193
31	36	57	66	244	172	42	122	191	110
135	221	252	177	219	231	264	283	22	281
164	147	137	81	141	227	108	143	45	114
163	262	268	250	248	258	260	118	273	275

- 2) Make *public static double nthRoot (int n, double x)*, which returns the nth root of x using your own generalization of Newton's method for finding square roots, as discussed in class. Explain your technique, and embed the method within a program called *RootFinder.java*, which I will be able to test.
- 3) Here's a maximum heap, illustrated as an array:

<u>index</u>	<u>value</u>
1	82
2	61
3	75
4	18
5	31
6	53

- Insert 80 and, after it has "settled," draw the heap (as an array).
- From there, remove the max value and redraw the heap.
- From there, remove the max value again and redraw the heap.
- Insert 43 and, after it has "settled," redraw the heap.

- e) The point of heaps is that they implement all of the priority queue operations in, worst-case, $\theta(\log n)$ time. Does this hold up if Java's *ArrayLists* are used as the underlying structure for the heap (rather than arrays)? Why (or why not)?
- 4) Make *Select*, a program that takes command-line argument, n , plus a file of arbitrary integers (redirected from standard input), and outputs the n th smallest among those integers, with duplications permitted. Your program should implement random partitioning, as described in lecture. A typical invocation of your program might look like this:

```
java Select 53 < SomeFileFullOfIntegers
```

- 5) Make *BucketSort*, a program that takes an arbitrary file of doubles from the standard input, then outputs them in ascending order, using the bucket sort discussed in class. Read the data into an *ArrayList*; use small *ArrayLists* (rather than linked lists) for the buckets; and merge the buckets back into the original *ArrayList* before outputting the results. A typical invocation of your program might look like this:

```
java BucketSort < SomeFileFullOfDoubles
```