Extra review problems for exam 1

1: Let T(n) be the time for weirdsort to sort n elements. Find a recurrence relation for the order of T and find the theta order of the time for weirdsort(A, 1, n). (It is a separate logic question, *skipped* here, to see that weirdsort will in fact sort A[0..n-1].)

```
weirdsort(A, i, n)
// n >= 1, sorts n elements A[i] …. A[i+n-1]
  if n == 2) and A[i] > A[i+1]
     swap A[i] and A[i+1]
  if n <= 2
    return
  k = ⌊n/3⌋  // round down 1/3 of n
  weirdsort(A, i, n – k) // first 2/3
  weirdsort(A, i+k, n-k) // second 2/3
  weirdsort(A, i, n – k) // first 2/3 AGAIN
```

2: Suppose you are given an array A of n elements, A[0]... A[n-1], and a number r, 0 <= r < 1. Give an efficient (on average) algorithm that will find the number in A that would be at position ⌊rn⌋ in the array after sorting (but obviously you cannot do a full sort in O(n)).
Example, If array A, after sorting would yield

i	0	1	2	3	4	5	6	7
A[i]	2	4	5	8	9	12	14	20

and r is .3, then the result is A[⌊.3(8)⌋] = A[2] = 5.

3. Young Tableau
An mxn Young tableau Y is an m x n matrix where the elements in each row are sorted in increasing order and columns are also sorted in increasing order, so if i < k and j < p,
Y[i][j] <= Y[k][j] and Y[i][j] <= Y[i][p].
A value of infinity means an element is missing.

a: Draw a 4x3 Young tableau containing 2, 7, 9, 14, 3, 8, 1, 11, 5, 6.

b: Give an ExtractMin algorithm on a Young tableau that is O(n+m). Hint: use a recursive subroutine that reduces an m x n problem to either an (m-1) x n or a m x (n-1) problem. This function returns the smallest element after it eliminates it from the tableau, replacing it by infinity in the tableau, and fixing the tableau to make it legal again.

c: Give an Insert algorithm that is O(m+n)

d: Using no other sorting method as a subroutine, show how to use an n x n Young tableau to sort n^2 numbers in $O(n^3)$.

Hints: page 2
Solutions or solution outlines: page 3

Hints:

1. Straightforward recurrence relation solved with Master Theorem
2. In class and in the book finding the median was discussed (r = .5). This is a direct generalization.
3. a. Do it – help visualize for later general algorithms
   bc. A lot like heap operations to add or remove a number.
   d. Use b, c.

Solutions or outline

1: $T(n) = 3T(\lceil 2n/3 \rceil) + \Theta(1)$

$E = (\lg 3)/\lg (3/2))$, $\Theta(n^E)$

2: Follow median idea. Partition. If location is where the pivot goes – done. Otherwise recurse in the part containing to index you want.

3:a Lots of solutions. Will always work if you entered sorted numbers diagonally. I use "-" for infinity.

```
1, 2, 5, 8
3  6  9  14
7 11 -    -
```

b.

I will assume indices 1..n for simplicity

```
deleteMin(A, m, n)
    retval = A[1,1]
    inc(A, 1, 1, m, n)
    return retval

inc(A, rLow, cLow, rHigh, cHigh)
    // assume all in place except A[rLow][cLow];
   // like bubble down in heap
    // in place of children are elements one row down or one column over (if not out of range)
    if val is not the smallest of the 1-3 elements to consider:
        swap it with the smallest element.
        Call inc with rLow, cLow replaced by the location where val was swapped
```

Analysis: nonrecursive part O(1) and the sum of rows + column left to consider decreases by 1, so order O(n+m)

c. insert: Assume that if an element can be added, A[m][n] was infinity before. Replace that with the new value and exactly mimic inc, except decreasing indices from m, n, rather than increasing. Same time analysis.

d. combine b,c as in heapsort

```
Start with n xn tableau of infinity
for each of n*n elements to sort:
    insert element into tableau
n*n times:
   output value returned from deleteMin acting on the tableau
```

Time analysis: Since m=n, inner function call O(n+n) = O(n), repeated n*n times in each part, giving $O(2n^3) = O(n^3)$