Section Worksheet #6

1. Sort 3, 1, 4, 1, 5, 9, 2, 6, 5 using insertion sort.

Input	3	1	4	1	5	9	2	6	5
i=0	*3	1	4	1	5	9	2	6	5
i=1	3	1*	4	1	5	9	2	6	5
Insert	1	3	4	1	5	9	2	6	5
i=2	1	3	*4	1	5	9	2	6	5
i=3	1	3	4	*1	5	9	2	6	5
Insert	1	1	3	4	5	9	2	6	5
i=4	1	1	3	4	*5	9	2	6	5
i=5	1	1	3	4	5	*9	2	6	5
i=6	1	1	3	4	5	9	*2	6	5
Insert	1	1	2	3	4	5	9	6	5
i=7	1	1	2	3	4	5	9	*6	5
Insert	1	1	2	3	4	5	6	9	5
i=8	1	1	2	3	4	5	6	9	*5
	1	1	2	3	4	5	5	6	9

2. Sort 3, 1, 4, 1, 5, 9, 2, 6, 5 using merge sort.

Continually cut input into half.

1. (3, 1, 4, 1, 5, 9, 2, 6, 5)
2. (3, 1, 4, 1) (5, 9, 2, 6, 5)
3. (3, 1) (4, 1) (5, 9) (2, 6, 5)
4. (3) (1) (4) (1) (5) (9) (2,6) (5)
5. (3) (1) (4) (1) (5) (9) (2) (6) (5) (on this step, only need to divide the last chunk undivided chunk

Now merge back together, sorting each pair by going through each pair that was divided, adding the lesser values of one to a new array until the other array in the pair has lesser values, add all those, etc.

6. (1,3)(1,4) (5,9) (2,6) (5)
7. (1, 1, 3, 4) (5,9) (2,5,6)
8. (1, 1, 3, 4) (2, 5, 5, 6, 9)
9. (1, 1, 2, 3, 4, 5, 5, 6, 9)

3. Sort 3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5 using quick sort with median-of-three pivot, with insertion sort cutoff at 3.

At first step, take median of items at index 0, index 10, and index 5. Use this as your pivot. So for the first example, pivot is median of 3, 9, and 5, answer is 5. Place pivot at beginning of the array. Now your array is 5 1 4 1 5 9 2 6 5 3 3.

Set i=1 and j=10. Then go inward with your j pointer until you find an item that is less than the pivot, and stop. Go through inward with your i pointer until you find an item that is greater than the pivot. That point is when i=5 (a[5]=9) and j=10 (a[10]=3). Swap these items, then continue moving j and i inward until you find another item that needs to be swapped, at i=7 and j=9. Swap again, and continue moving inward. At some point both

i=8 and j=8, so they have "crossed". Stop, and swap the value at a[i] (which is 5) with the pivot, which was at a[0]=5.

Your array should now look like: 5 1 4 1 5 3 2 3 5 6 9.

Your median is at a[8], and so your two partitions are (5 1 4 1 5 3 2 3) and (6 9). The right partition has only 2 items, so that's less than the cutoff, so you can just insertion-sort it (which here is pretty trivial, no swaps needed).

For the left partition, find a pivot (median(5, 1, 3) so pivot is 3, which is at a[7]), and move to the beginning of this partition (which is a[0]). Set i=1 and j=7 (the edges of this partition, except for the pivot), and move inward again as before, swapping whenever i and j are stopped at items that don't belong there (i.e. are greater than for i, or less than for j, the pivot). Here you will end up swapping at i=2 and j = 6. Continue moving i and j, there will be no more swaps, and i=3 and j=3, at which point they have crossed, and so you stop. The partition now looks like (3 1 2 1 5 3 4 5) and the whole array now looks like 3 1 2 1 5 3 5 3 4 5 6 9. Swap the pivot at a[0] with the a[i], which is a[3]=1, and now your partition is (1 1 2 *3* 5 3 4), with the middle 3 as the just swapped pivot. Partition again into: (1 1 2) 2 (5 3 4). Both sides are at the cutoff, so we can insertion sort them: (1 1 2) and (3 4 5). Going up, those combine to be (1 1 2 2 3 4 5), then that combines all the way back up to (1 1 2 2 3 4 5 5 6 9).

4. Sort 25, 36, 85, 93, 21, 74, 22, 12 using radix sort with radix=10.

Table shows the two passes below. After first pass, go through the buckets and output in order: 21, 22, 12, 93, 74,25,85,36. Then take that input, and use it to second pass, and output results from that pass similarly (sorted order).

	0	1	2	3	4	5	6	7	8	9
Pass 1		21	22	93	74	25	36			
			12			85				
Pass 2		12	21	36				74	85	93
			22							
			25							

5. What would be the runtimes of the following algorithms if your data were all identical (only one unique item, e.g 7,7,7,7,7), sorted, or reverse sorted?

	Identical	Sorted	Reverse-sorted
Insertion Sort	O(n)	O(n)	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Heapsort	O(n)	O(n log n)	O(n log n)
Mergesort	O(n log n)	O(n log n)	O(n log n)
Quicksort	$O(n^2)$	O(n log n)	O(n log n)
Bucket Sort	O(n+k)	O(n+k)	O(n+k)
Radix Sort	O(P(B+n))	O(P(B+n))	O(P(B+n))