CS3364 – Design and Analysis of Algorithms
Key for Review sheet for test 1

1. Given the flowing piece of code, in the worst case it is big-Θ of what (where the input is of size n)?

```
Foo(A)
1   k = 0
2   for i = 1 to A.length
3      A[i] = A[i] + 1
4      for j = 1 to A. length - i
5         A[j] = A[j] * A[i]
6         if A[j] > 20
7            k = k + 1
8            A[i] = k
9   for i = 1 to ⌊A.length / 2⌋
10     exchange A[i] with A[A.length + 1 - i]
```

Answer: n^2

2. Know and understand the ordering on different rates of growth on different standard functions (i.e. lg n grows slower than n, n^5 grows slower that 2^n, etc.)

Answer: in general something of order 1 grows slower than lg n which is slower than order n, which is slower than n ln n, which is slower than n^2, which is slower than 2^n, which is slower than n!. Note: for any numbers j >1 and k > 1, if j < k then n^j grows slower than n^k, and j^n grows slower than k^n, however, in general, for any constant j, n^j grows slower than 2^n although as j gets large n has to be large to overcome the size of j.

3. Under what situation might bubble sort run faster than quicksort on a given data set?

Answer: There are several possible correct answers here.

1) For small inputs (n is small).
2) If the input gives the worst case for quicksort (quicksort is n^2 in the worst case. So is bubble sort but the constant is smaller)
3) If bubble sort is implemented such that it stops if no data is swapped during a run through the list then the best case complexity is Θ(n) which is faster than quicksort.

4. Given the following data stored in an array in the order given, what would the contents of the array look like after the first application of the partition routine from quicksort used in the book?

17, 42, 12, 8, 34, 16, 9, 4, 22, 43, 13, 19

Answer (showing work):
Pivot is last element of array, 19.
The item we are currently looking at is underlined.

^ will mark index position just after last thing we have seen less than or equal to the pivot
Initially we have
17, 42, 12, 8, 34, 16, 9, 4, 22, 43, 13, 19
^
17 is ≤ the pivot so we swap with the item at position ^ (itself in this case) and move ^ index one to the right and go to the next item.
17, 42, 12, 8, 34, 16, 9, 4, 22, 43, 13, 19
    ^
42 is greater than the pivot so we just move go to the next item.
17, 42, 12, 8, 34, 16, 9, 4, 22, 43, 13, 19
    ^
12 is ≤ the pivot so we swap it with the item at position ^, 42, and move ^ index one to the right and go to the next item.
17, 12, 42, 8, 34, 16, 9, 4, 22, 43, 13, 19
       ^
8 is ≤ the pivot so we swap it with the item at position ^, 42, and move ^ index one to the right and go to the next item.
17, 12, 8, 42, 34, 16, 9, 4, 22, 43, 13, 19
          ^
34 is greater than the pivot so we just move go to the next item.
17, 12, 8, 42, 34, 16, 9, 4, 22, 43, 13, 19
          ^
16 is ≤ the pivot so we swap it with the item at position ^, 42, and move ^ index one to the right and go to the next item.
17, 12, 8, 16, 34, 42, 9, 4, 22, 43, 13, 19
             ^
9 is ≤ the pivot so we swap it with the item at position ^, 34, and move ^ index one to the right and go to the next item.
17, 12, 8, 16, 9, 42, 34, 4, 22, 43, 13, 19
                ^
4 is ≤ the pivot so we swap it with the item at position ^, 42, and move ^ index one to the right and go to the next item.
17, 12, 8, 16, 9, 4, 34, 42, 22, 43, 13, 19
                   ^
22 is greater than the pivot so we just move go to the next item.
17, 12, 8, 16, 9, 4, 34, 42, 22, 43, 13, 19
                   ^
43 is greater than the pivot so we just move go to the next item.
17, 12, 8, 16, 9, 4, 34, 42, 22, 43, 13, 19
                   ^
13 is ≤ the pivot so we swap it with the item at position ^, 34, and move ^ index one to the right and go to the next item.
17, 12, 8, 16, 9, 4, 13, 42, 22, 43, 34, 19
                      ^
19 is ≤ the pivot so we swap it with the item at position ^, 42, and move ^ index one to the right and go to the next item.
17, 12, 8, 16, 9, 4, 13, 19, 22, 43, 34, 42
                         ^

We have reached the end. The partition is complete and we return the index of the pivot (which will be one position before the ^).

5. Given the initial data in the order below, show the data after each of the 3 iteration of the radix sort routine:

234, 526, 278, 129, 312, 221, 193, 426

Answer:

initially		sort on rightmost digit		sort on middle digit		sort on leftmost digit
234		221		312		129
526		312		221		193
278		193		526		221
129	=>	234	=>	426	=>	234
312		526		129		278
221		426		234		312
193		278		278		426
426		129		193		526

Note: must be a stable sort used at each step, i.e. if the numbers are equal on a position they must remain in the same order they were in the previous step, for example, there were 4 numbers with 2 as the middle digit, so when sorting on the middle digit they stayed in the same order they occurred in the previous step.

6. The Master Method has the following 3 cases:

1. if $f(n) = O(n^{(\log_b a)-e})$ for some constant $e > 0$, then $T(n) = \Theta(n^{\log_b a})$
2. if $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$
3. if $f(n) = \Omega(n^{(\log_b a)+e})$ for some constant $e > 0$ and if $af(n/b) \leq cf(n)$ for some constant $c > 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.

What is the asymptotic bounds of $T(n) = 8T(n/2) + n$?

Answer: $T(n)$ is $aT(n/b) + f(n)$, so a is 8, b is 2, and $f(n)$ is n. $\log_2 8 = 3$
So notice $f(n) = n$ which is $O(n^{3-e})$ if $0 < e \leq 2$ so case 1 applies and $T(n) = \Theta(n^3)$.

7. Describe how the max-heapify routine from heapsort works (make sure to state what assumptions are made on the input).

Answer: max-heapify assumes we have a binary tree where the left and right subtrees of the root are both max-heaps but the root may not be in the right position for the whole tree to be a max heap. The algorithm works as follows:

1) start with the current node being the root of the whole tree
2) if the current node is a leaf or its data is bigger than the data of both of its children then stop,

3) otherwise swap the data in the current node with the data in the bigger of the two children and go back to step 2) with the current node being the child you swapped with.

8. Describe counting sort. When is counting applicable?

Answer: Counting sort is applicable when all the values in the array being sorted are between the value of 0 and some K (inclusive). The algorithm then works as follows:

1) Create and array, A, from 0 to K which is initially all 0s.
2) Loop through input and for each item i add 1 to A[i].
3) (without going into details of books implementation) Create a new output array and loop through array A, putting A[i] i's into the output.

9. Other than searching, sorting, and the max sub-array problem given in class, give another problem for which a divide and conquer algorithm would be applicable.

A possible answer: given a continuous mathematical function, f(X), and two values a and b where f(a) > 0 and f(b) < 0 (or vice versa), the problem of finding a value, c, such that f(c) = 0 can be found by a divide and conquer algorithm. While the question didn't as how, the way to do so is as follows:

```
While True (repeat until a value is returned)
    Let c be a/b.
    if f(c) = 0 return c.
            else
            if f(c) is > 0 then
                        if f(a) > 0 the let a = c else b = c
                else
                        if f(a) < 0 the let a = c else b = c
```

10. Write the code for a sort routine that is $O(n^2)$. You may use pseudocode so long as the control flow is understandable.

One Possible Answer: Selection sort: Given a input list of length n, first create an output list which is initially empty, next go through the input list n times doing the following 2 steps: 1) use a linear search to find the smallest item in the input list 2) remove the smallest item from the input list and append it to the output list

11. What properties should a good hash function have?

Answer: The function should run quickly given the input data, and should hash to all available bins, with a roughly even distribution. It must be deterministic (calling the function on a specific item should return the same bin each time it is called with that item), therefore the function cannot use any randomization and should not be based on a part of the data item that could change over time.