Algorithm Design and Analysis
Review for Exam 1

Friday, February 17, 2017
Material Covered: Lectures 1-6

Brief Summary of Material:

Lecture 1:
- Counting iteration examples, know formulas for sum of first n integers and sum of harmonic numbers (1+1/2+…+1/n). Be able to analyze similar examples.

Lecture 2:
- Big O, Big Omega, and Big Theta formal definitions.
- Know the basic complexity classes and be able to rank in order.
- Be able to justify the complexity class of a given function: basic calculus can be helpful here, such as derivatives of polynomials, derivatives of logarithmic and exponential functions, and L’Hopital’s rule
- Know the time and space complexity of Insertion sort
- Understand and be able execute an insertion sort on a simple array

Lecture 3:
- Understand and be able to execute a Merge Sort on a simple array
- Know the best and worst case running time of Merge Sort in addition to the space complexity.
- Understand and be able to solve the recurrence relation for Merge Sort using either the substitution or recurrence tree method.

Lecture 4:
- Be able to derive and solve the recurrence equation for binary search.
- Be able to solve a recurrence that is similar to the recurrence that arises through the recursive integer multiplication algorithm
- Be able give examples off algorithms that correspond to common recurrence relations (last page of lecture) and be able to solve such recurrences.
- The master method will not be tested directly on this exam.

Lecture 5:
- Understand how to both partition an array and how this partition plays into the quicksort algorithm
- Be able to produce a tree of the sequence of function calls required to perform a quicksort on a given array

Lecture 6:
- Understand the best and worse case analysis of Quicksort, and be able to justify why the average running time tends to the best case running time.
- Know the alternative ways to chose pivots.
Sample questions: (not exhaustive)

1. For the following program, what is the order of the running time? For an input value of N, what value does func1 return?

```c
int func1(int n)
{
    int a, i, j;
    a = 0;
    for (i = 0 ; i < n ; i++)
    {
        for (j = i ; j >= 0 ; j--)
        {
            a++;
        }
    }
    return a;
}
```

2. What is the worst-case running time of each of the following sorting algorithms? What is the best-case running time? What is the space-complexity of each algorithm?

- **Insertion Sort** __________________
- **Bubble Sort** __________________
- **Merge Sort** __________________
- **Quick Sort** __________________

Useful Recurrences and their running time (all of which you should be able to justify). What is an example of an algorithm that corresponds to each of the recurrences?

- $T(n) = T(n/2) + O(1)$
  - $O(\log n)$
- $T(n) = T(n-1) + O(1)$
  - $O(n)$
- $T(n) = 2 T(n/2) + O(1)$
  - $O(n)$
- $T(n) = T(n-1) + O(n)$
  - $O(n^2)$
- $T(n) = 2 T(n/2) + O(n)$
  - $O(n \log n)$

4. State whether the following statements are true or false:
   a. $$\text{___} \ lg \ n = O(n^2)$$
   b. $$\text{___} 3n^2 + 2n = \Omega(n)$$
   c. $$\text{___} f(n) = \Omega(g(n)) \text{ implies } f(n) = \Theta(g(n))$$
5. In Quicksort, all comparisons are done in the PARTITION procedure. If you make the PARTITION procedure twice as fast, what happens to the running time of Quicksort?

6. Here’s an algorithm:

```c
int mystery(int n)
{
    int s = 0;
    if (n = 0)
        return s;
    else
        return n + mystery(n-1);
}
```

a) What does it compute?

b) Write the recurrence that describes the running time of this algorithm, and solve it.

c) Write an algorithm that accomplishes the same task as the one above, but does not use recursion. Make sure the running time of your algorithm is at least as good or better.

7. Let A[1..n] be an array of n distinct numbers. If i < j and A[i] > A[j] then the pair (i, j) is called an inversion. For example, the array <2, 3, 8, 6, 1> contains five inversions. Write an algorithm INVERSIONS(A, n) that determines the number of inversions in A[1..n]. Give the running time of your algorithm.

8. Professor Lupin is applying for tenure. In his application he sites the following algorithm as one of his greatest achievements. A is an array, p is the index of the first element, and n is the number of elements in the array.

```c
LUPIN( A, p, n )
if (n <= 2)
    then return
    then exchange A[1] with A[n]
LUPIN(A, 2, n-2)
LUPIN(A, 2, n-1)
```

(a) Write down a recurrence describing the number of times two members of array A are compared, measured as a function of the array length n.
(b) Find the running time of algorithm LUPIN by solving the recurrence in (a)

(c) Which problem is algorithm LUPIN solving?

(d) Can you solve the same problem more efficiently? How?

(e) Would you recommend Prof. LUPIN for tenure?

9. For a sorted array A[1..n], give and justify expressions for worst-case complexity (O) for:

   (a) Inserting an element into A

   (b) Deleting an element from A

   (c) Determining A[i] when i is given

   (d) Determining i when A[i] is given