After Lecture 03 & 04 – Answer any questions on HW1
Practice Problems (all taken from previous exams)

1. Which of the following is time complexity of fun()?

```
int fun(int n){
        int count = 0;
        for(int i = 0; i < n; i++){
                for(int j = i; j > 0; j--){
                        count = count + 1;
                }
        }
        return count;
}
```

   a) $O(n)$
   b) $O(n^2)$
   c) $O(n \log n)$
   d) $O(n \log(n) \log(n))$

2. Consider the following function. What is the returned value of the above function?

```
int unknown(int n){
        int i, j, k=0;
        for(i = n/2; i <= n; i++){
                for(j=2; j <= n; j=j*2){
                        k = k + n/2;
                }
        }
        return k;
}
```

   a) $\Theta(n^2)$
   b) $\Theta(n^2 \log n)$
   c) $\Theta(n^3)$
   d) $\Theta(n^3 \log n)$

3. What is the worst-case auxiliary space complexity (including stack space for recursion) of merge sort?

   a) $O(1)$
   b) $O(\log n)$
   c) $O(n)$
   d) $O(n \log n)$

4. Choose the incorrect statement about merge sort from the following:

a) It is a comparison-based sort.

b) It's runtime is dependent on input order.

c) It is not an in-place algorithm (all the operations are on the original array).

d) It is a stable algorithm.

5. Use the definition of big-O to prove or disprove.

5a) is $2^{n+1}? =?O\left(2^n\right)$True:

$$2^{n+1} = C2^n$$
$$2 = C \text{ if c } \geq 2$$

5b) is $2^{2n}? =?O\left(2^n\right)$False:

$$2^{2n} = C2^n$$
$$2^n = C$$

However 2^n has an infinite range so it cannot be upperbounded

6. Although merge sort runs in $\Theta(n \lg n)$ worst-case time and insertion sort runs in $\Theta(n^2)$ worst-case time, the constant factors in insertion sort make it faster for small n. Thus, it makes sense to use insertion sort within merge sort when sub-problems become sufficiently small. Consider a modification to merge sort in which n/k sub-lists of length k are sorted using insertion sort and then merged using the standard merging mechanism, where k is a value to be determined.

6a) Show the n/k sub-lists; each of length k, can be sorted by insertion sort in $\Theta(nk)$ worst-case time.

$$k \to \Theta\left(k^2\right)$$
$$\frac{n}{k} \to \Theta\left(\frac{n}{k} \times k^2\right) = \Theta(nk)$$

6b) Show that the sub-lists can be merged in $\Theta(n \lg\left(\frac{n}{k}\right))$ worst-case time.

$$\frac{n}{k} \to \Theta\left(\frac{n}{2k}\right) \text{ for merging}$$
$$\Theta(2k) \text{ m?? for } 2k \text{ elements}$$
$$\Theta\left(\frac{n}{2k} \times 2k\right) = \Theta\left(n\right)\Theta\left(\frac{n}{4k} \times 4k\right) = \Theta\left(n\right)$$

6c) Given that the modified algorithm runs in $\Theta(nk + n \lg\left(\frac{n}{k}\right))$ worst-case time, what is the largest asymptotic (Θ-notation) value of k as a function of n for which the modified algorithm has the same asymptotic running time as standard merge sort?

6d) How should k be chosen in practice?

7. The Fibonacci sequence 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ... is defined recursively as

Algorithm 1 The Fibonacci sequence

```
1: function FIB(n)      ▷ This mathematical definition leads naturally to a recursive algorithm
2:     if n ≤ 1 then
3:         return n
4:     else
5:         return FIB(n-1) + FIB(n-2)
6:     end if
7: end function
```

7a) Write the recurrence relation, $T(n)$, for the asymptotic runtime for procedure FIB(n) shown above, and solve the recurrence relation to show that $T(n) = O\left(2^{n-2}\right)$.

7b) Another recursive procedure which computes the nth Fibonacci number is below.

Algorithm 2

```
1: function F1(n)
2:     if n < 2 then
3:         return n
4:     else
5:         return F2(2, n, 1, 1)
6:     end if
7: end function
8:
9: function F2(i, n, x, y)
10:    if i ≤ n then
11:        F2(i + 1, n, y, x + y)
12:    end if
13:    return x
14: end function
```

Trace out the algorithm as it computes F1(1), F1(2), F1(3), F1(4), explain how the algorithm works, and then compare its asymptotic runtime to the time for procedure FIB(n).

8. Use mathematical induction to show that when n is an exact power 2, the solution of the recurrence

$$T(n) = \begin{cases} 2 & \text{if } n = 2 \\ 2T(n/2) + n & \text{if } n = 2^k, \text{ for } k > 1 \end{cases}$$

is $T(n) = n \lg n$