After Lecture 01 & 02 – Answer any questions on HW1
Practice Problems (all taken from previous exams)

1. Which of the following is not true of improved bubble sort (keep track of last swap position on the inner loop and use that to reduce outer loop iterations) on the case on input elements sorted?

   a) It is stable

   b) Consumes less memory

   c) Detects whether the input is already sorted

   d) Consumes less time

2.

Statement 1: In insertion sort, after m passes through the array, the first m elements are in sorted order.

Statement 2: And these elements are the m smallest elements in the array.

   a) Both of the statements are true.

   b) Statement 1 is true but statement 2 is false

   c) Statement 1 is false but statement 2 is true

   d) Both of the statements are false

3. Consider the following program that attempts to locate an element x in a sorted array $a[]$ using binary search. The program is erroneous. Under what conditions does the program fail?

Algorithm 1 Erroneous Binary Search

```
1:  function BS
2:      int i ← 1, j ← 100, k, x          ▷ assume x is assigned a value to search for
3:      int[] a ← new int[100];            ▷ assume values loaded in sorted order
4:      repeat
5:          k ← (i+j)/2
6:          if a[k] < x then
7:              i ← k;
8:          else
9:              j ← k;
10:         end if
11:     until ((a[k] == x) || (i ≥ j))
12:     if a[k] == x then
13:         System.out.println("x is in the array")
14:     else
15:         System.out.println("x is not in the array")
16:     end if
17: end function
```

a) x is the last element of the array $a[]$

b) x is greater than all elements of the array $a[]$

c) Both of the Above

d) x is less than the last element of the array $a[]$

4. What's the worst case of insertion sort if the correct position for inserting element is calculated using binary search?

a) $O(\log n)$

b) $O(n)$

c) $O(n \log n)$

d) $O(n^2)$

5. The following routine takes as input a list of n numbers, and returns the first value of i for which $L[i] < L[i-1]$, or n if no such number exists.

```
int firstDecrease(int* L, int n){
        for(int i=2; i <= n && L[i] >= L[i-1]; i++){}
        return i;
}
```

5a) What is the big-O runtime for the routine, measured as a function of its return value i?

5b) If the numbers are chosen independently at random, then the probability that firstDecrease(L) returns i is $\frac{i-1}{i!}$, except for the special case of $i = n+1$ for which the probability is $\frac{1}{n!}$ Use this fact to write an expression for the expected value returned by the algorithm. (Your answer can be expressed as a sum, it does not have to be solved in closed form. Do not use O-notation.) Use expectation

5c) What is the big-O average case running time of the routine? Hint: Simplify the previous summation until you see a common taylor series.

6. Some sorting algorithms are NOT stable. However if every key in $A[i]$ is changed to $A[i] * n + i - 1$ (assume $1 \leq i \leq n$) then all the new elements are distinct (and therefore stability is no longer a concern). After sorting, what transformation will restore the keys back to their original values? What is the effect on the runtime of any of the sorting algorithm if you add this transformation before executing the sort and un-transformation after the sort?

$$A[i] \rightarrow A[i] * n + i - 1$$

7. a) Use pseudocode to specify a brute-force algorithm that determines when a sequence of n positive integers is given as input, whether there are two distinct terms of the sequence that have as sum a third term. The algorithm should loop through all triples of the sequence, checking whether the sum of the first two terms equals the third.

b) Give a big-O estimate for the complexity of the brute-force algorithm from part (a).

c) Devise a more efficient algorithm for solving the problem that first sorts the input sequence and then checks for each pair of terms whether their sum is in the sequence.

d) Give a big-O estimate for the complexity of this algorithm. Is it more efficient than the brute-force algorithm?