CS 430 – SPRING 2023
INTRODUCTION TO ALGORITHMS
HOMEWORK #1
DUE THURSDAY, Jan. 26

Ethics: Any behavior on any homework or exam that could be considered copying or cheating will result in an immediate zero on the assignment for all parties involved and will be reported to academichonesty@iit.edu See the IIT Code of Academic Honesty, https://web.iit.edu/student-affairs/handbook/fine-print/code-academic-honesty

1. (4 points)
1a) Use pseudocode to specify a brute-force algorithm that determines when given as input a sequence of *n* positive integers whether there are two distinct terms of the sequence that have as sum a third term. The algorithm should loop through all triples of terms of the sequence, checking whether the sum of the first two terms equals the third.

1b) Give a big-*O* estimate for the complexity of the brute-force algorithm from part (a).

1c) 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.

1d) Give a big-*O* estimate for the complexity of this algorithm. Is it more efficient than the brute-force algorithm?

2. (2 points) Prove, by induction on k, that level k of a binary tree has less than or equal to 2^k nodes (root level has k=0).

3. (2 points) Use definition of big O to prove or disprove.
3a) is 2^(n+1) ?=? O(2^n)

3b) is 2^(2n) ?=? O(2^n)

4. (3 points) 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;
}
```

4a) What is the big-O runtime for the routine, measured as a function of its return value i?

4b) If the numbers are chosen independently at random, then the probability that firstDecrease(L) returns i is (i-1)/i!, except for the special case of i = n+1 for which the probability is 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.)

4c) What is the big-O average case running time of the routine? Hint: Simplify the previous summation until you see a common taylor series.

5. (3 points) Consider the following program and recursive function.

```
#include <iostream.h>
void Z(int[], int, int);
void swap (int&, int&);

void main() {
        int A[3]={1,2,3};
        int n=3;
        Z(A, n, 0);
}

void swap(int &x, int &y) {
        int temp;
        temp = x;
        x = y;
        y = temp;
}
```

```
void Z(int A[], int n, int k) {
        if (k == n-1) {
                for (int i=0; i<n; i++)
                        cout << A[i] << " ";
                cout << endl;
        }
        else {
                for (int i=k; i<n; i++) {
                        swap(A[i], A[k]);
                        Z(A, n, k+1);
                        swap(A[i], A[k]);
                }
        }
}
```

5a) Demonstrate the execution, show the output, and explain what the program accomplishes.

5b) Give a recurrence equation describing the worst-case behavior of the program.

5c) Solve the recurrence equation.

6. (4 points) textbook: Problem 2-4: Inversions
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 of A.
6a) List the five inversions of the array <2, 3, 8, 6, 1>.

6b) What array with elements from the set {1, 2, . . . , n} has the most inversions? How many does it have?

6c) What is the relationship between the running time of insertion sort and the number of inversions in the input array? Justify your answer.

6d) Give an algorithm that determines the number of inversions in any permutation on n elements in $\Theta(n \lg n)$ worst-case time. (Hint: Modify merge sort.)

7. (2 points) Give big-O bounds for T(n) in each of the following recurrences. (Substitution or Recursion Tree)

7a) $T(n) = T(n-1) + n$

7 b) $T(n) = T(n/4) + T(n/2) + n^2$