CS 430 – FALL 2023
INTRODUCTION TO ALGORITHMS
HOMEWORK #1

1. **(5 points) Let $A[1 \ldots n]$ 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.**

   1a) List the five inversions of the array $< 2, 3, 8, 6, 1 >$.

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

   1c) What is the relationship between the running time of insertion sort and the number of inversions in the input array? Justify your answer.

2. **(5 points) Communication security is extremely important in computer networks, and one way many network protocols achieve security is to encrypt messages. Typical cryptographic schemes for the secure transmission of messages over such networks are based on the fact that no efficient algorithms are known for factoring large integers. Hence, if we can represent a secret message by a large prime number p, we can transmit over the network the number $r = p * q$, where $q > p$ is another large prime number that acts as the encryption key. An eavesdropper who obtains the transmitted number r on the network would have to factor r in order to figure out the secret message p.**
   Using factoring to figure out a message is very difficult without knowing the encryption key q. To understand why, consider the following naive factoring algorithm:

   For every integer p such that $1 < p < r$, check if p divides r. If so, print "The secret message is p!" and stop; if not, continue.

   2a) Suppose that the eavesdropper uses the above algorithm and has a computer that can carry out in 1 microsecond (1 millionth of a second) a division between two integers of up to 100 bits each. Give an estimate of the time that it will take in the worst case to decipher the secret message if r has 100 bits.

   2b) What is the worst-case time complexity of the above algorithm? Since the input to the algorithm is just one large number r, assume that the input size n is the number of bytes needed to store r, that is, $n = (\log_2 r)/8$, and that each division takes time $O(n)$.

3. **(5 points) Bubblesort can be described as follows: Consider sorting (in increasing order) n numbers by comparing the 1st and 2nd number and swapping them if they are out of order, then comparing the 2nd and 3rd number and swapping them if necessary, ..., comparing the $(n-1)$ st and nth number and swapping them if necessary. Then do this again for $n-1$ numbers, then $n-2$ numbers, etc. When no exchanges are required on some pass, the file is sorted.**
   A different version of bubble sort keeps track of where the last swap occurred, and on the next pass, it will not go past this point. If the last change was made in the swap of locations i and $i+1$, the next pass will not look at any elements past location i.

3a) Write iterative pseudocode for this new version of Bubblesort.

3b) Write a short paragraph that explains exactly why this new version of bubble sort will work.

3c) How does this new version of bubble sort does change the worst-case runtime analysis? Can you state what particular input will give worst-case runtime? Give a detailed explanation of what is involved in calculating the worst-case runtime analysis.

4. (5 points) Give an algorithm that determines the number of inversions (see problem #1) in any permutation on n elements in $\Theta(n \lg n)$ worst-case time. (Hint: Modify merge sort.)

5. (5 points) Prove, by induction on k, that level k of a binary tree has less than or equal to $2k$ nodes (root level has $k = 0$).

6. (5 points)

6a) Describe the final returned value of the recursive algorithm below on a general input N:

```
1: function X(int N)                                   ▷ Return type: int
2:     if N ≤ 1 then
3:         return 1
4:     else
5:         return X(N-1) + X(N-1)
6:     end if
7: end function
```

6b) Perform asymptotic analysis, find the recurrence relation, and give big-O bounds on the running time of X.

6c) The following algorithm "Y" generates the same result (verify this). Perform asymptotic analysis, find the recurrence relation, and give big-O bounds on the running time of Y.

```
1: function Y(int N)                                   ▷ Return Type: int
2:     if N ≤ 1 then return 1
3:     else
4:         temp ← Y(N-1)
5:         return (temp + temp)
6:     end if
7: end function
```

7. (5 points)

```
1: function FOO(int A[], int start, int end)        ▷ Initial call FOO(array, 0, array.length)
2:     if end ≤ 1 then return
3:     end if
4:     if A[start] > A[end-1] then
5:         swap A[start] and A[end-1]
6:     end if
7:     FOO(A, start+1, end-2)
8:     if A[start] > A[start+1] then
9:         swap A[start] and A[start+1]
10:    end if
11:    FOO(A, start+1, end-1)
12: end function
```

Write a recurrence describing the number of times the algorithm compares two members of array A, measured as a function of the array length n. You do not need to solve the recurrence relation exactly, but state if the solution is linear growth, polynomial growth or exponential growth.

8. (5 points) Give big-O bounds for $T(n)$ in each of the following recurrences. Use induction, iteration or Master Theorem.

8a) $T(n) = 2T(n/8) + n^{\frac{1}{3}}$

8b) $T(n) = T(n-2) + n$

8c) $T(n) = 7T(n/3) + n^2$