CS 430 – SPRING 2023
INTRODUCTION TO ALGORITHMS
HOMEWORK #2
(20pts)

1. (3 pts) Show in detail that the running time to sort in increasing order using QUICKSORT is T(n^2) when the array A contains distinct elements and is initially sorted in decreasing order.

2. (4pts) Stack depth for QUICKSORT - The QUICKSORT algorithm of Section 7.1 contains two recursive calls to itself. After the call to PARTITION, the left sub array is recursively sorted and then the right subarray is recursively sorted. The second recursive call in QUICKSORT is not really necessary; it can be avoided by using an iterative control structure. Good compilers provide this technique, called tail recursion.. Consider the following version of quick sort, which simulates tail recursion.

```
QUICKSORT'(A, p, r)
  while p < r
        do     // Partition and sort left sub array.
             q = PARTITION (A, p, r)
             QUICKSORT'(A, p, q - 1)
             p = q + 1
```

2a) Argue that QUICKSORT '(A, 1, length [A]) correctly sorts the array A.(1pt)

2b) Compilers usually execute recursive procedures by using a stack that contains pertinent information, including the parameter values, for each recursive call. The information for the most recent call is at the top of the stack, and the information for the initial call is at the bottom. When a procedure is invoked, its information is pushed onto the stack; when it terminates, its information is popped. Since we assume that array parameters are represented by pointers, the information for each procedure call on the stack requires O(1) stack space. The stack depth is the maximum amount of stack space used at any time during a computation. Describe a scenario in which the stack depth of QUICKSORT' is $\Theta(n)$ on an n-element input array.(1pt)

2c) Modify the code for QUICKSORT' so that the worst-case stack depth is $\Theta(\lg n)$. Maintain the $O(n \lg n)$ expected running time of the algorithm.(2pts)

3. (6 pts) Hoare Partition Correctness - The version of PARTITION given in chapter 7 is not the original partitioning algorithm. Here is the original partition algorithm, which is due to T. Hoare:

```
HOARE-PARTITION(A, p, r)
 1  x ← A[p]
 2  i ← p - 1
 3  j ← r + 1
 4  while TRUE
 5      do repeat j ← j - 1
 6            until A[j] ≤ x
 7         repeat i ← i + 1
 8           until A[i] ≥ x
 9         if i < j
10            then exchange A[i] ↔ A[j]
11            else return j
```

3a) Demonstrate the operation of HOARE-PARTITION on the array A = {13, 19, 9, 5, 12, 8, 7, 4, 11, 2, 6, 21}, showing the values of the array and auxiliary values after each iteration of the for loop in lines 4–11. (1pt)

The next three questions ask you to give a careful argument that the procedure HOARE-PARTITION is correct. Prove the following:

3b) The indices i and j are such that we never access an element of A outside the subarray A[p . . r].(2pts)

3c) When HOARE-PARTITION terminates, it returns a value j such that $p \le j < r$.(1pt)

3d) Every element of A[p . . j] is less than or equal to every element of A[j +1 . . r] when HOARE-PARTITION terminates.(2pt)

4. (2 pts) What are the minimum and maximum numbers of elements in a heap of height h?

5. (3 pts) The code for MAX-HEAPIFY is quite efficient in terms of constant factors, except possibly for the recursive call in line 10, which might cause some compilers to produce inefficient code. Write an efficient MAX-HEAPIFY that uses an iterative control construct (a loop) instead of recursion.

6. (2 pts) What is the smallest possible depth of a leaf in a decision tree for a comparison sort? Explain why in detail.