CS 240: Data Structures and Data Management **Spring 2023**

Tutorial 05: June 5

1. **Expected Runtime**
   Give the best-case and expected running time for the following function. You can assume that the Shuffle operation requires $\mathcal{O}(n)$ time and the array A contains no duplicates.
   Note: the *Shuffle*() function produces each permutation equally likely.

```
MonkeySort(A)    // A is an array oof n elements
    shuffle(A)
    if A is sorted
        return A
    else
        return MonkeySort(A)
```

2. **Recursion Tree Height Analysis**
   Suppose we have an algorithm as follows, analyze its behaviour and run time. We suppose that array A is sorted. Additionally, Array splitting and printing is done in $O(1)$ time.

```
Partition_and_Find(Array A, size n):
    if (size <= 1) return;
    left = A[0 ... n/2-1];
    right = A[n/2 ... n-1];
    middle = A[n/4 ... 3n/4-1];

    Partition_and_Find(left, n/2);
    Partition_and_Find(middle, n/2);
    Partition_and_Find(right, n/2);


    // bool BinarySearch(Array A, int value);
    result = BinarySearch(A, (A[0]+A[n-1])/2);
    print(result);
    return 0;
```

3. **Efficient In-Place Partition (Hoare Partition)**
   With this question, we will take a look at Hoare Partition and go through an example. Apply following pseudo-code with `A = [8,17,10,1,6,20,9,2,13,7]` and `p=2`

```
partition(A, p)
A: array of size n,   p: integer s.t. 0 ≤ p < n
1.    swap(A[n−1], A[p])
2.    i ← −1,   j ← n−1,   v ← A[n−1]
3.    loop
4.        do i ← i+1 while A[i] < v
5.        do j ← j−1 while j ≥ i and A[j] > v
6.        if i ≥ j then break   (goto 9)
7.        else swap(A[i], A[j])
8.    end loop
9.    swap(A[n−1], A[i])
10.   return i
```