# DSC 40B - Discussion 03

**Problem 1.**

Solve the following recurrence relations.

**a)** T(n) = T(n-1) + n
T(0)=0

**Solution:** Discussion timestamp $\to XX : XX$

$$\begin{aligned}
T(n) &= T(n-1) + n \\
&= [T(n-2) + n - 1] + n \\
&= T(n-2) + 2n - 1 \\
&= [T(n-3) + n - 2] + 2n - 1 \\
&= T(n-3) + 3n - (1 + 2) \\
&= [T(n-4) + n - 3] + 3n - (1 + 2) \\
&= T(n-4) + 4n - (1 + 2 + 3)
\end{aligned}$$

We can infer that $T(n) = T(n-k) + kn - \sum_{i=1}^{k-1} i$ in the k-th step.
T(0) is the base case. $n - k = 0$ when $n = k$.

$$\sum_{i=1}^{n-1} i = \frac{(n-1)n}{2} = \frac{n^2 - n}{2}$$

$$\begin{aligned}
T(n) &= T(n-n) + n \cdot n - \sum_{i=1}^{n-1} i \\
&= T(0) + n^2 - \frac{n^2 - n}{2} \\
&= 0 + n^2 - \frac{n^2 - n}{2} \\
&= \theta(n^2)
\end{aligned}$$

**b)** T(n)=4T(n/4) + n
T(1)=1

**Solution:** Discussion timestamp $\to XX : XX$

$$\begin{aligned}
T(n) &= 4 \cdot T(n/4) + n \\
&= 4\,[4 \cdot T(n/16) + n/4] + n \\
&= 16 \cdot T(n/16) + 2n \\
&= 16\,[4 \cdot T(n/64) + n/16] + 2n \\
&= 64 \cdot T(n/64) + 3n
\end{aligned}$$

We can infer that in the k-th step, we have:

$$= 4^k \cdot T(n/4^k) + k \cdot n$$

The base case will be reached when $n/4^k = 1$, that is, when $k = \log_4 n$. Substituting this value of $k$ into the general expression:

$$\begin{aligned} T(n) &= 4^{\log_4 n} \cdot T(n/4^{\log_4 n}) + n \cdot \log_4 n \\ &= n \cdot T(n/n) + n \cdot \log_4 n \\ &= n \cdot T(1) + n \cdot \log_4 n \end{aligned}$$

Since $T(1) = 1$, we have:

$$\begin{aligned} &= n + n \cdot \log_4 n \\ &= \Theta(n \log_4 n) \end{aligned}$$

Since logarithms of different bases differ only by a constant factor, we typically omit the base when using asymptotic notation:

$$= \Theta(n \log n)$$

**Problem 2.**

Determine the recurrence relation describing the time complexity of each of the recursive algorithms below.

**a)**
```
def fact(n):
    if(n <= 1)
        return 1
    else
        return n*fact(n-1)
```

**Solution:** Discussion timestamp $\rightarrow XX : XX$
T(1) = 1
T(n) = T(n-1) + 1

**b)**
```
def max_arr(arr):
    if(len(arr) == 1):
        return arr[0]
    mid = len(arr)//2
    left_max = max_arr(arr[:mid])
    right_max= max_arr(arr[mid:])
    if(left_max>right_max):
        return left_max
    else:
        return right_max
```

**Solution:** Discussion timestamp $\rightarrow XX : XX$
T(1)=1
T(n)=2T(n/2)+n

**Problem 3.**

Determine whether each piece of code is correct or incorrect.

**a)**
```
def max_arr(arr):
    max1 = arr[0]
    max2 = max_arr(arr[1:])
    if(max1 > max2):
        return max1
    else:
        return max2
```

**Solution:** Discussion timestamp $\rightarrow XX : XX$
The code does not have a base case. Hence, the arr[1:] will result in an error when the size of the array is 1.

**b)**
```
def fib(n):
    if (n==1):
        return 1
    return fib(n-1)+fib(n-2)
```

**Solution:** Discussion timestamp $\rightarrow XX : XX$
The base case of n==2 is not handled. Therefore, the code will run into an infinite recursion.

**Problem 4.**

We'll consider an array of `True`s and `False`s to be sorted if all of the `False`s come before any of the `True`s, like in `[False, False, True, True, True]`.

The function `find_first_true(arr, start, stop)` below is a generalization of the binary search we saw in lecture. It accepts a sorted array of `True`s and `False`s and returns the index of the first `True` (if there is one) within `arr[start:stop]`; if the array contains all `False`s, it returns `stop`.

```
def find_first_true(arr, start, stop):
    if stop - start == 1:
        if arr[start]:
            return start
        else:
            return stop

    middle = math.floor((start + stop) / 2)
    if arr[middle]:
        return find_first_true(arr, start, middle)
    else:
        return find_first_true(arr, middle, stop)
```

Modify this function so that it takes in a sorted array of floats and a number $a$ and returns the index of the first element that is $\geq a$.

**Solution:**

```
def find_first_geq(arr, x, *, start=0, stop=None):
    """Find first index that is >= x in arr[start:stop], where arr is sorted"""
    if stop is None:
        stop = len(arr)

    if stop - start == 1:
        if arr[start] >= x:
            return start
        else:
            return stop

    middle = math.floor((start + stop) / 2)
    if arr[middle] >= x:
        return find_first_geq(arr, x, start=start, stop=middle)
    else:
        return find_first_geq(arr, x, start=middle, stop=stop)
```

**Problem 5.**

Given a sorted array of distinct integers A[1 . . . n], give an algorithm to find out whether there is an index i for which A[i] = i. Analyze the time complexity of the algorithm.
(Hint : There is a solution that runs in better than linear $\Theta(n)$ time!)

**Solution:** Discussion timestamp $\rightarrow XX : XX$

```
def fixedPoint(arr, start, stop):
    if (start >= stop):
        return False
```

```
    mid = int((start + stop)/2)
    if(arr[mid] == mid):
        return True
    if(arr[mid] > mid):
        return fixedPoint(arr, start, mid)
    else:
        return fixedPoint(arr, mid+1, stop)
```

The time complexity of the algorithm is $\theta(logn)$.