DSC 40B - Discussion 03

Problem 1.

a) State (but do not solve) the recurrence relation describing this function's run time.

```python
import random
def foo(n):
    if n <= 2:
        return
    for i in range(n):
        for j in range(i,n):
            print(i)
    return foo(n//2) + foo(n//2)
```

Solution: $T(n) = 2T(n/2) + \Theta(n^2)$

b) Suppose a binary search is performed on the following array using the implementation of *binary_search* from lecture. What is the worst case number of comparison's that would be made to search for an element in the array.

`[1, 4, 7, 8, 8, 10, 15, 51, 60, 65, 71, 72, 101]`

Solution: 3. Since we know that the time complexity of binary search is $\Theta(logn)$

Problem 2.

We're given two lists, `A` and `B` and a target `t`, and our goal is to find an element a of `A` and an element b of `B` such that $a + b = t$.

Solution:

```python
def target_sum(A,B,t):
    """
        A and B are list of sorted numbers
        t is the target to be found
    """

    # since the two arrays are sorted. we start with
    # initializing one pointer with the first index of any one array
    # and the other pointer with the last index of the other array
    A_i, B_i = 0, len(B) - 1

    while A_i < len(A) and B_i >= 0:

        current_sum = A[A_i] + B[B_i]

        # found target then return the values a and b
        if current_sum == t:
            return (A[A_i], B[B_i])
```

```
        elif current_sum < t:
            # current_sum was smaller! since we are incrementing the values in A
            # and decrementing the values in B
            # Increasing A_i will mean that the element in A that we get next will
            # increase our current sum, which is what is expected to find the target
            A_i += 1
        else:
            # current sum was larger!
            # Decreasing B_i will mean that the element in B that we get next will
            # decrease our current sum, which is what is expected to find the target
            B_i -= 1

    return None
```

Problem 3.

Solve the following recurrence relations.

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

Solution:

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

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:

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

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 4.

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:
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:
T(1)=1
T(n)=2T(n/2)+n