CS200 - Recurrence Worksheet

1. Create a recurrence relation for the worst case runtime of the following algorithm for binary search when the array is initially size n. You may assume n is a power of 2. Use the iterative method to solve the recurrence relation.

Algorithm 1: `BinarySearch`(A, V)

Input : (1) Array (list) A containing integers, where there are no repeated integers and the integers are sorted from smallest to largest, (2) an element V

Output: True if there is an index g such that $A[g] = V$ and False otherwise.

```
1  l=length of A;
   // Base Case
2  if l = 1 and A[1] = V then
3  |  return True;
4  else
5  |  return False;
6  end
   // Recursive step
7  mid = ⌊l/2⌋;
8  if A[mid] < V then
9  |  return BinarySearch(A[mid + 1 : l], V);
10 else
11 |  return BinarySearch(A[1 : mid], V);
12 end
```

Solution Let $T(n)$ be the run time of the algorithm, when n is the size of the initial array. Then for $n > 1$,

$$T(n) = T(n/2) + O(1) \tag{1}$$

because each time we run the algorithm, we make one recursive call on an input of size $n/2$, and we additionally do some constant amount of work. Using the definition of T, we have that this one recursive call takes time $T(n/2)$.

The base case is:

$$T(1) = O(1). \tag{2}$$

So the recurrence relation is

$$T(1) = O(1) \qquad \text{and} \qquad T(n) = T(n/2) + O(1) \tag{3}$$

Then using the iterative approach, we have

$$\begin{aligned} T(n) &= T(n/2) + O(1) \\ T(n) &= T(n/4) + O(1) + O(1) \\ T(n) &= T(n/8) + O(1) + O(1) + O(1) \end{aligned} \tag{4}$$

We see the pattern is

$$T(n) = T(n/2^k) + O(k). \tag{5}$$

We are interested in the point where $T(n/2^k)$ becomes $T(1)$, because we know $T(1)$. Now $n/2^k = 1$ when $2^k = n$, which happens when $k = \log_2(n)$. Plugging this value of k into Eq. (5), we have

$$T(n) = T(1) + O(\log_2(n)) = O(1) + O(\log_2(n)) = O(\log_2(n)). \tag{6}$$

Thus, the worst case time complexity of binary search is $O(\log_2(n))$.

2. Create a recurrence relation for the number of ways a person can climb n stairs if the person can take one stair or two stairs at a time. How many ways can this person climb a flight of 5 stairs? (For this problem, order matters, so if the person takes three steps by taking the first step by itself and the next two together, that is different than if the person takes the first two steps together, and the third by itself. Think about the options for the very last step.)

Solution Let $T(n)$ be the number of ways the person can take n stairs.

If the person takes n stairs, they could either take the last stair on it's own, or they could take the last two stairs together. If they take the first stair as one step, there are $T(n-1)$ ways that they could take the first $n-1$ stairs to get there, so there are $T(n-1)$ ways of taking the final stair alone. If they take the final two stairs together, there are $T(n-2)$ ways that they could have gotten to the $(n-2)$th stair, and then one way that they can take the final two together.

Thus

$$T(n) = T(n-1) + T(n-2). \tag{7}$$

For the base case, we have

$$T(1) = 1 \tag{8}$$

because there is only one way to take one stair. But if we only have this base case, we fall off the ladder when we try to analyze $T(2)$, so we add $T(2)$ as a base case.

$$T(2) = 2 \tag{9}$$

because the person could either take the 2 stairs at once, or could go up one at a time.

Thus the full recurrence relation is

$$T(1) = 1, \tag{10}$$
$$T(2) = 2, \tag{11}$$
$$T(n) = T(n-1) + T(n-2). \tag{12}$$

Then

$$T(5) = T(4) + T(3) = T(3) + T(2) + T(2) + T(1) = T(3) + 5 = T(2) + T(1) + 5 = 8 \tag{13}$$

3. Let $T(n, k)$ be the number of strings in $\{0, 1, 2\}^n$ whose digits sum to the number k. Create a recurrence relation for $T(n, k)$.

Solution Consider a string in $\{0, 1, 2\}^n$. There are three options for the final digit: 0, 1, or 2. We consider each case separately.

If the string ends with 0, then we need the remaining $n - 1$ digits to add up to k. There are $T(n - 1, k)$ ways of doing this.

If the string ends with 1, then we need the remaining $n - 1$ digits to add up to $k - 1$. There are $T(n - 1, k - 1)$ ways of doing this.

If the string ends with 2, then we need the remaining $n - 1$ digits to add up to $k - 2$. There are $T(n - 1, k - 2)$ ways of doing this.

Thus $T(n, k) = T(n - 1, k) + T(n - 1, k - 1) + T(n - 1, k - 2)$.

Now for the base case. Let's start with $T(1, 0) = 1$. But then if we try to calculate $T(2, 0)$ we fall off the ladder. So we add $T(2, 0) = 1$. But if we try $T(3, 0)$ we fall off the ladder. At this point we hopefully notice that we are never going to stop falling off the ladder, but we also notice $\forall n \in \mathbb{N}, T(n, 0) = 0$. Also, $\forall n \in \mathbb{N}, T(n, 1) = n$. With these two families of base cases, we will never fall off the ladder.

So our final recurrence relation is:

$$\forall n \in \mathbb{N}, T(n, 1) = n, \tag{14}$$
$$\forall n \in \mathbb{N}, T(n, 0) = 1, \tag{15}$$
$$T(n, k) = T(n - 1, k) + T(n - 1, k - 1) + T(n - 1, k - 2) \tag{16}$$

where here instead of individual base cases, we have families of base cases.