Name & Surname:
ID:

CSE 321 - Quiz #2

October 25^{th}, 2024

1. ***45 pts.*** Solve the following recurrence relations with the given initial conditions.

   (a) $x(n) = x(n-1) + 7 \qquad n > 1, \qquad x(1) = 1$

   (b) $x(n) = 2 \cdot x(n-1) + 1 \qquad n \geq 1, \qquad x(1) = 1$

   (c) $x(n+1) = 4 \cdot x(n) - 4 \cdot x(n-1) \qquad n \geq 1, \qquad x(0) = 1, \qquad x(1) = 6$

Solution:

(a) $x(n) = x(n-1) + 7$
$x(n) = x(n-2) + 7 + 7$
$x(n) = x(n-3) + 7 + 7 + 7$
$x(n) = x(n-i) + i \cdot 7$
For $i = n-1$;
$x(n) = x(1) + (n-1) \cdot 7$
$x(n) = 1 + 7 \cdot n - 7$
$x(n) = 7 \cdot n - 6$

(b) $x(n) = 2 \cdot x(n-1) + 1$
$x(n) = 2 \cdot (2 \cdot x(n-2) + 1) + 1 = 2^2 \cdot x(n-2) + 2 + 1$
$x(n) = 2^2 \cdot (2 \cdot x(n-3) + 1) + 2 + 1 = 2^3 \cdot x(n-3) + 4 + 2 + 1$
$x(n) = 2^i \cdot x(n-i) + \sum_{k=1}^{i} 2^{k-1}$
$x(n) = 2^i \cdot x(n-i) + \frac{1-2^i}{1-2}$
$x(n) = 2^i \cdot x(n-i) + 2^i - 1$
For $i = n-1$;
$x(n) = 2^{n-1} \cdot x(1) + 2^{n-1} - 1$
$x(n) = 2^n - 1$

(c) The characteristic equation of the given recurrence relation is $\alpha^2 - 4\alpha + 4 = 0$
$\alpha^2 - 4\alpha + 4 = (\alpha - 2)^2 = 0$
$\alpha_{1,2} = 2$

General solution:
$x(n) = c_1 \cdot \alpha_1^n + c_2 \cdot n \cdot \alpha_2^n = c_1 \cdot 2^n + c_2 \cdot n \cdot 2^n$

Using initial conditions:

- $x(0) = c_1 \cdot 2^0 + c_2 \cdot 0 \cdot 2^0 = c_1 = 1$
  We can update our solution as $x(n) = 2^n + c_2 \cdot n \cdot 2^n$
- $x(1) = 2^1 + c_2 \cdot 1 \cdot 2^1 = 2 + 2 \cdot c_2 = 6$
  Therefore, $c_2 = 2$.

Final solution:
$x(n) = 2^n + n \cdot 2^{n+1} = 2^n \cdot (1 + 2 \cdot n)$

2. ***55 pts.*** Consider an array with n integers that has an ascending order until index i and a descending order after index i, where $0 \leq i \leq n-1$. The following algorithm finds the i^{th} element of the array, i.e. the peak element.

```
FindPeakElement(A, n):
    Input: An integer array A and its size n.
    Output: Peak element.
    begin
        low ← 0
        high ← n − 1
        while low≤high repeat
            mid ← (low + high) // 2
            if (mid == 0 or A[mid] ≥A[mid-1]) and (mid == n-1 or A[mid] ≥ A[mid+1]) then
                return A[mid]
            else
                if mid > 0 and A[mid-1] > A[mid] then
                    high = mid -1
                else
                    low = mid + 1
                end if
            end if
        end while
    end
```

Sample inputs and outputs of the provided algorithm are as follows:

- **Example 1:**
  Input: $A = [1, 2, 2, 4, 3]$, $n = 5$
  Output: 4
- **Example 2:**
  Input: $A = [11, 12, 40, 60, 80, 99]$, $n = 6$
  Output: 99
- **Example 3:**
  Input: $A = [3, 2, 1]$, $n = 3$
  Output: 3

Analyze the average-case time complexity of the given algorithm. Make sure you provide a formal proof rather than merely explaining it.

Solution:

The provided algorithm uses binary search mechanism to find the peak element. So, it has an average-case time complexity of $\Theta(logn)$.

As seen in the pseudo-code, the first *if* statement is where we find the solution and then return. Thus, the number of executions of this if statement will define the time complexity (since there is no other operation that dominates this). There is a limit to the while statement (low≤high), therefore there is a limit to the if comparisons, but that limit defines the worst-case scenario. In this question, we must focus on the average case.

In order to prove the average-case time complexity, we should show that a solution requires approximately $logn$ comparisons on average.

Simply, the probability of finding the peak element at index i where $0 \leq i < n$ is $\frac{1}{n}$. However, the number of comparisons to find the solution at index i varies. For instance, if the solution is the

middle element, it will only require 1 comparison. If the solution is the middle element of the left (or right) half of the array, it will require 2 comparisons. We also know that if the solution is the first or the last element, it will require $logn$ comparisons since that is the worst-case scenario. In other terms, we need at least 1 comparison and at most $logn$ comparisons.

If we are to list all these, we will obtain the following pattern:

of the elements that require 1 comparison: 1 (only the middle element)
of the elements that require 2 comparisons: 2 (the middle elements of the left and right halves)
of the elements that require 3 comparisons: 4
of the elements that require 4 comparisons: 8
...
...
of the elements that require k comparisons: 2^{k-1}
...
...
of the elements that require $logn$ comparisons: 2^{logn-1}

In order to compute the average number of comparisons, we should find the average of these comparisons. Simply, we can compute the summation of these comparisons and divide it by n (since there are n elements).

The summation can be formed as follows:

$1 \cdot 1 + 2 \cdot 2 + 3 \cdot 4 + ... + logn \cdot (2^{logn-1}) = \sum_{k=1}^{logn} (k \cdot 2^{k-1})$

This summation corresponds to the internal path length (IPL) of a binary tree. To be more specific, if we are to call our tree T, this summation is equal to $IPL(T) + n$ (n is being added since we have to make a comparison at each step. In other terms, internal path length considers the length of the root node from the root node as 0, which is expected. However, if the root node is our solution, we should make at least 1 comparison. Therefore, we have an additional operation for each node. Since we have n nodes in the tree, we have n additional operations).

For a binary tree T, $IPL(T) = LPL(T) - 2 \cdot I$, where LPL is the leaf path length and I is the number of internal nodes. We also know that $LPL(T) \geq L \cdot \lfloor logL \rfloor + 2 \cdot (L - 2^{\lfloor logL \rfloor})$, where L is the number of leaves. This gives us the following:

$1 \cdot 1 + 2 \cdot 2 + 3 \cdot 4 + ... + logn \cdot (2^{logn-1}) = ILP(T) + n = LPL(T) - 2 \cdot I + n = LPL(T) - n$
$\geq L \cdot logL + 2 \cdot (L - 2^{logL}) - n = n \cdot logn - n$

As mentioned above, we should divide this solution by n to find the average number of comparisons to find the solution at position i. Thus, we obtain $\frac{(n \cdot logn - n)}{n} = logn - 1$.

An average of (at least) $logn - 1$ comparisons corresponds to the $\Omega(logn)$ (Because of the $\geq$ sign, it should be Ω. We also ignore the constant -1 since it will not affect the asymptotic complexity. This can be proved through the limit method.)

We also know that the worst-case scenario is in $O(logn)$. Which gives us $\Theta(logn)$ for the average-case.