Khrystian Clark

Assignment 2, CS325

18JAN2022

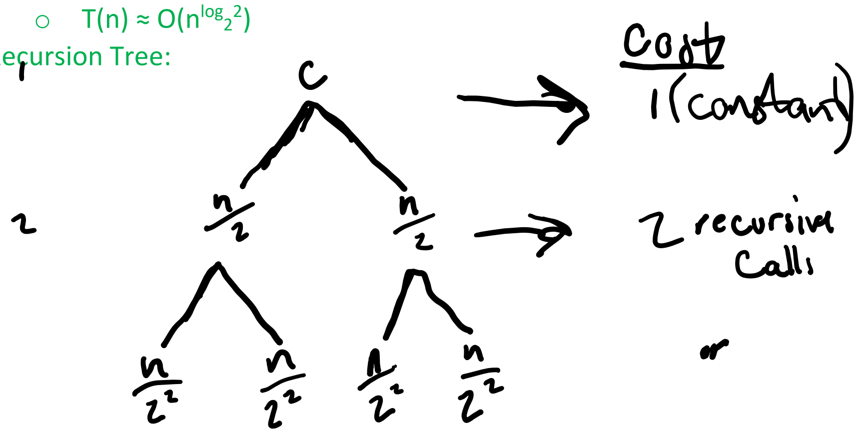
Assignment: Recursion, Recurrence Relations and Divide & Conquer

1. **Solve recurrence relation using three methods**:

   Write recurrence relation of below pseudocode that calculates x^n, and solve the recurrence relation using three methods that we have seen in the explorations.

```
power2(x,n):
    if n==0:
        return 1
    if n==1:
        return x
    if (n%2)==0:
        return power2(x, n//2) * power2(x,n//2)
    else:
        return power2(x, n//2) * power2(x,n//2) * x
```

- Recurrence Relation:
  - Base case
    - T(n)=c1 for n = 0
    - T(n)=c2 for n = 1
    - T(n)=c3 for n%2 = 0
  - Multiple param
    - T(n)=2T(n/2)
  - T(n) = 2T(n/2)+c if n > 1
- Substitution:
  - T(n) = 2T(n/2)+C
    - $T(n/2)=2T(n/2^2)+C$
    - $T(n)=2[2T(n/2^2)+C]+C$
  - $T(n)=2^2T(n/2^2)+2C+C$
  - $T(n)=2^3[2T(n/2^3)+C]+C(2+1)$
  - $T(n)=2^kT(n/2^k)+((k-1)+.....2^2+2+1)C$
    - $=1 \Rightarrow n = 2^k \Rightarrow k=\log_2 n$
    - $2^k(C) + 2^{(k-1)}(C)$
    - $2^k[C+ 2C] = 2^{\log_2 n}(C)$
  - Subst: $T(n)=n^{\log_2 2}$, or $T(n) \approx 2^k$
  - $T(n) \approx O(n^{\log_2 2})$
- Recursion Tree:



3

$\rightarrow 4(n/2^2)$

$\vdots$

Cost $n(1+2+4+...+2^k)$

k $\frac{n}{2^k} = 2^k \Rightarrow n = 2^k \Rightarrow \log n = \log 2^k \rightarrow k \log_2 2$

$k = \log_2 n \rightarrow \frac{2^{\log_2 n} - 1}{2-1} = 2n-1$

$= O(n)$

- Master:
  - T(n) = aT(n/b) + f(n)
    - a=2
    - b=2
    - $n^2 = n^{\log_2 2}$
    - T(n) = 2T(n/2) + c
    - Comparing $n^{\log_b a}$ to c
    - Assume c=1
    - $n^{\log_2 2}$ > c, satisfies case 1
      - $O(n^{\log_2 2}) \Rightarrow O(n)$

2. **Solve recurrence relation using any one method**:
   Find the time complexity of the recurrence relations given below using any one of the three methods discussed in the module. Assume base case T(0)=1 or/and T(1) = 1.
   a) $T(n) = 4T(n/2) + n$
      a. Master method:
         i. a=4, b=2, f(n)=n
         ii. compare $n^{\log_2 4}$ to f(n)
            1. $n^{\log_2 4} > n$
            2. $n^2 > n$
         iii. Satisfies case 1, $T(n) = O(n^{\log_2 4}) = O(n^2)$
   b) $T(n) = 2T(n/4) + n^2$
      a. Master method:
         i. a=2, b=4. $f(n)=n^2$
         ii. compare $n^{\log_4 2}$ to f(n)
            1. $n^{\log_4 2} > n$
            2. $n^{1/2} < n^2$
         iii. Satisfies case 3. So $T(n) = O(n^2)$

3. **Implement an algorithm using divide and conquer technique**: Given two sorted arrays of size m and n respectively, find the element that would be at the k^{th} position in combined sorted array.

   a. Write a pseudocode/describe your strategy for a function kthelement(Arr1, Arr2, k) that uses the concepts mentioned in the divide and conquer technique. The function would take two sorted arrays Arr1, Arr2 and position k as input and returns the element at the k^{th} position in the combined sorted array.

```
Kthelement(Arr1, Arr2, k);
    Combine the arrays into one;
        m = len(Arr1);
        n = len(Arr2);
        Merge the sorted arrays;
            Merge each piece of the sorted arrays one by one into the
            new array
            If you reach the end of one of the arrays, continue to add the
            rest of the other array
    return combinedArr[k];
```

b. Implement the function kthElement(Arr1, Arr2, k) that was written in part a. Name your file **KthElement.py**

Examples:

Arr1 = [1,2,3,5,6] ; Arr2= [3,4,5,6,7] ; k= 5

Returns: 4

Explanation: 5th element in the combined sorted array [1,2,3,3,4,5,5,6,6,7] is 4