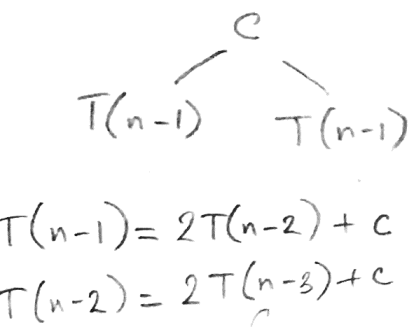
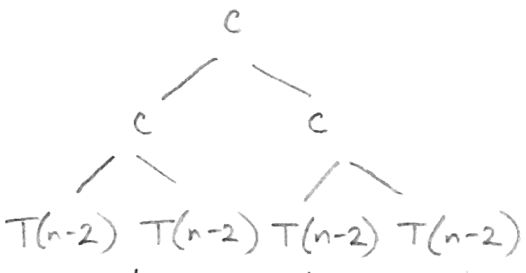
Name: (Print) PHONG VO

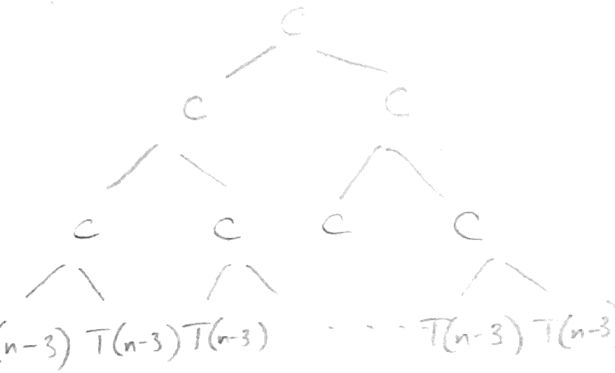
2. **Recurrence** (15 points) Derive a recurrence for the running time of the algorithm below and then solve the recurrence with one of the three methods we have learned. The solution should be a tight upper and lower bound solution. You must show the detailed work of how to solve the recurrence. You do NOT need to write the algorithm.

An algorithm solves a problem of size n by recursively solving **two** sub-problems of size (n – 1), and then combining the solutions in constant time.

$$T(n) = 2T(n-1) + \Theta(1)$$
$$= 2T(n-1) + c \quad (c.\ pos.\ const)$$

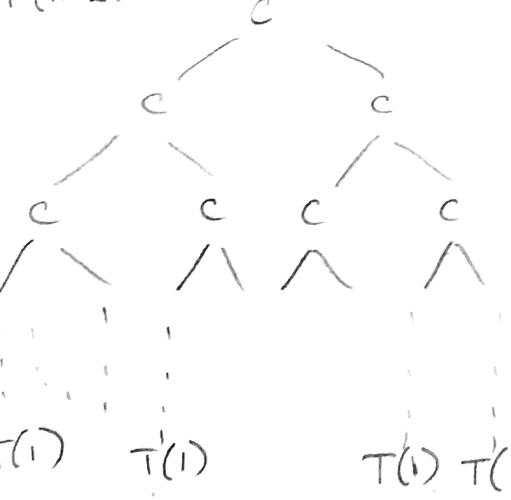






$T(n-1) = 2T(n-2) + c$

$T(n-2) = 2T(n-3) + c$



Level	Total cost
1	c
2	2c
3	4c
4	8c
⋮	⋮
i	$2^{i-1}c$

$$T(n) = c + 2c + 4c + 8c + \dots + 2^{n-1}c$$
$$= c(1 + 2^1 + 2^2 + 2^3 + \dots + 2^{n-1})$$
$$= c\frac{2^n}{2-1} = c \cdot 2^n = \Theta(2^n)$$

$T(1)$: $2^{i-1} = n \Rightarrow i - 1 = \lg n$

4. Analysis of an Algorithm (20 points)

You are given an array A, which stores *n* distinct numbers (n>=2). There is a mystery function called Mystery(A, n) that works on the array. The pseudocode of the algorithm is shown as below.

Please analyze the worst-case asymptotic execution time of this algorithm. (1) List the cost for executing each line of code and the number of executions for each line; and then derive a recurrence of the running time; (2) solve the recurrence by using one of the methods we have learned; must show your work clearly.

```
Mystery(A, n)
    return Mystery_helper(A, 1, n)


Mystery_helper (A, p, r)
  if  p==r-1
      if A[p] > A[r]
          return p
      else return r
  q = ⌊ (p+r)/2 ⌋
  if A[q] < A[q+1]
      return  Mystery_helper (A, q+1, r)
  else
      return  Mystery_helper (A, p, q)
```