1. Basics of Algorithmic Complexity

Algorithmic complexity measures the time and space resources consumed by an algorithm as the input size grows. Time complexity describes the rate of increase in the number of execution steps and is often expressed in asymptotic notation (O, Ω, Θ); space complexity describes the rate of increase in memory usage.

2. Recurrences

The time complexity of a divide-and-conquer algorithm is typically expressed as the recursive formula: $T(n) = aT(n/b) + f(n)$. Here, a represents the number of subproblems, n/b represents the size of the subproblems (b>1), and f(n) represents the complexity of the split and merge steps.

3. Substitution Method

The substitution method uses mathematical induction to prove a solution by guessing the solution. The steps are: (1) guess the form of the solution ;(2) assume that the solution holds for all sizes smaller than n; (3) substitute the solution into the recursive form to verify that n holds.

4. Master Theorem

The Master Theorem is used to quickly solve recursive forms of the form $T(n) = aT(n/b) + f(n)$, which can be divided into three cases:

- Case 1 (Leaf-node Dominant): If $f(n) = O(n^{\log_b a-\varepsilon})$ ($\varepsilon>0$), then $T(n) = \Theta(n^{\log_b a})$.
- Case 2 (balanced at each level): If $f(n)=\Theta(n^{\log_b a} \log^k n)$ ($k\geq0$), then $T(n)=\Theta(n^{\log_b a} \log^{k+1} n)$.
- Case 3 (root-dominated): If $f(n)=\Omega(n^{\log_b a+\varepsilon})$ ($\varepsilon>0$) and the regularity condition is met, then $T(n)=\Theta(f(n))$. 5. Recursion Tree Method

6. Selection Algorithm (Select(A, k) Problem)

The selection algorithm is used to find the kth smallest element in an unsorted array, with a worst-case time complexity of O(n).

Task 2: Recurrence using Substitution Method

Solve following recurrence using substitution method:

- $T(n) = \begin{cases} nT(n-1), & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases}$
- $T(n) = \begin{cases} T(n-1) + \log n, & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases}$
- $T(n) = \begin{cases} 3T(n/2) + n, & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases}$

A :

1. T(n) = n * T(n-1), then :

   T(n-1) = (n-1) * T(n-2), we can put T(n-1) back in to the first one.

   T(n) = n * (n-1) * T(n-2). So we follow this logic we can get rest of the functions:

   T(n) = n * (n-1) *(n-2) * T(n-3)…….. * T(1)

   Because when n=1 the T(n) = 1, therefore that's where the function end.
   Finally we can get T(n) = n * (n-1) * (n-2) * …….. = n!

   T(n) = n!

2. T(n)=T(n−1)+ logn :

   T(n-1) = T(n-2) + log(n-1), we can put T(n-1) back in to the first one.

   T(n) = T(n-2) + log(n-1) + log n, continue:

   T(n) = T(n-3) + log(n-2)+log (n-1) + log n …….etc,

   Because we know that T(1) = 1 , log 1 = 0

So T(n) = 1+ log(n!)

3. T(n)=3T(n/2)+n

T(n/2) = 3T(n/4) + n/2, we can put T(n/2) back in to the first one.

T(n) = 3[3T(n/4) + n/2] + n => 9T(n/4) + 3n/2 + n . Continue:

T(n/4) = 3T(n/8) + n/4

T(n) = 9[3T(n/8) + n/4] + 3n/2 + n => 27T(n/8) + 9n/4 + 3n/2 + n

Find out the rule for F(n):

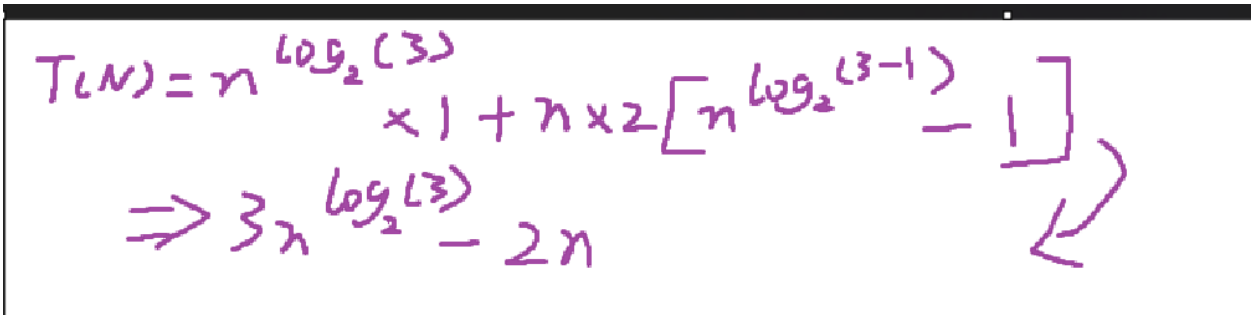
T(n) = 3^x * T(n/2^x)+ n * Σ k-1/i = 0 (3/2)^i

When n/2^x = 1 , so we will have :

T(n) = 3 ^log2(n) * T(1) + n * Σ log2 (n-1)/i = 0 (3/2)^i

Because base on the rule for a ^logb (c) = c ^ logb(a)
So 3^logx(n) = n^log2(3), plus log2 (3) = 1.58 so is larger than 1.

Therefore, T(n)=3n log2(3)−2n.



Task 3: Recurrence using Master Method

Solve following recurrence using Master Theorem:

- $$T(n) = \begin{cases} T(\sqrt{n}) + \log n, & \text{if } n > 1 \\ 1, & \text{if } n = 1 \end{cases}$$

A :

$T(n) = T(\sqrt{n}) + \log(n)$

let $n = 2^m$, so $\sqrt{n} = 2^{m/2}$, $\log n = m$

Let $S(m) = T(2^m)$

$S(m) = S(\frac{m}{2}) + m$

apply Master method

$S(m) = aS(\frac{m}{b}) + f(m)$, when

$a = 1, b = 2, f(m) = m$, so

$m^{\log_b a} \Rightarrow m^{\log_2 1} \Rightarrow m^0 = 1$

$S(m) = m + \frac{m}{2} + \frac{m}{4} + \ldots\ldots \Rightarrow 2m(\Theta(m))$

Put back To $T(n)$

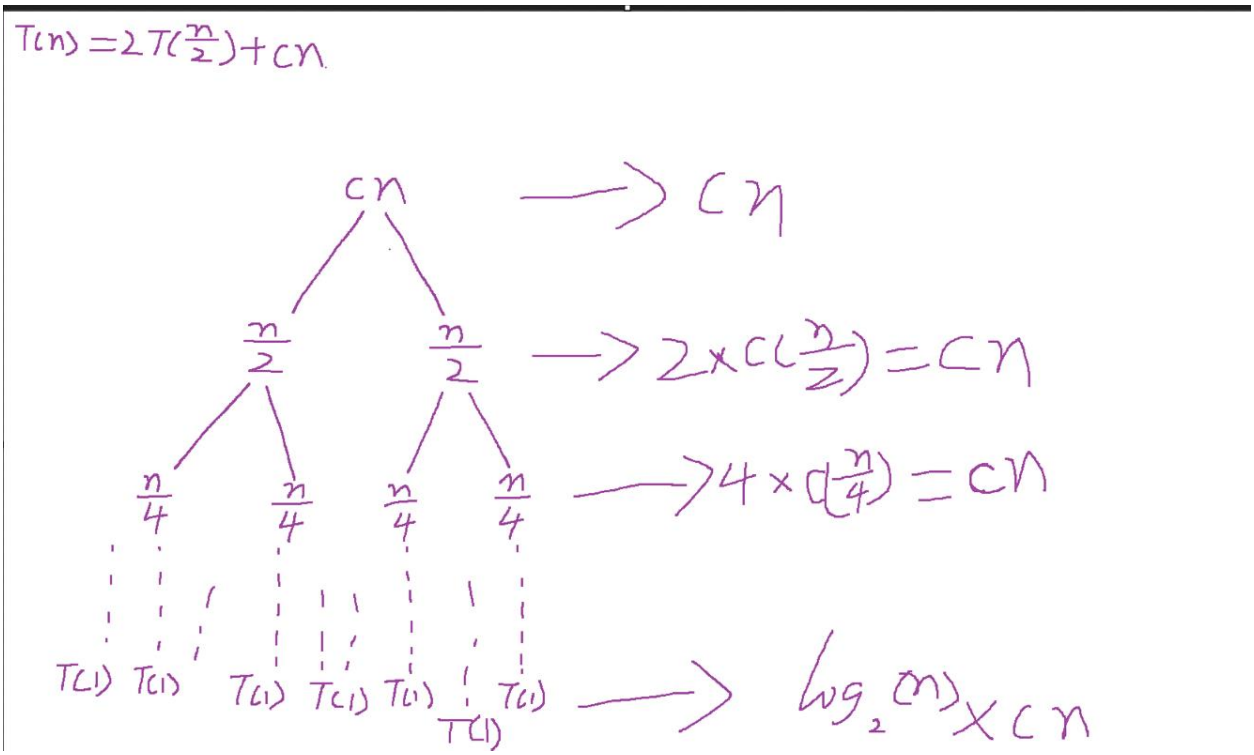
$T(n) = S(\log_2(n)) \Rightarrow \Theta(\log(n))$

so, $T(n) = \Theta(\log(n))$

Task 4: Recurrence using Recursion Tree Method

Solve following recurrence using recursion tree method:

- $T(n) = 2T(n/2) + cn$



The tree height : n/2^h = 1 , so h = log2 (n).

Number of leaf nodes: 2 ^ log2(n) = n

Leaf layer cost: n*T(1) = θ (n), when T(1) = θ (1)

So

T(n) = cn * log2(n) + θ (n) = θ (n*log(n))

Task 5: Solving select(A,K) Recurrence

We had a detailed analysis of select(A,k) problem in this module. Based on what you learned from this problem, analyse the complexity of Quick Sort Algorithm. Note, quick sort will lead to unbalanced sub-problems (much like select(A,k)) problem. You are expected to use either Akra-Baazi or induction method to solve the recurrence.

$$T(n) = \frac{2}{n}\sum_{q=0}^{n-1} T(q) + \Theta(n)$$

prove that: $T(n) \le cn \times \log(n)$

Let $k < n$, $T(k) \le ck \times \log(k)$

$$\therefore T(n) \le \frac{2}{n} \times \sum_{q=0}^{n-1} [cq \times \log(q)] + Dn$$

$$\Rightarrow \frac{2c}{n} \times \sum_{q=2}^{n-1} [q \times \log(q)] + O(1) + Dn$$

$$\because \sum_{q=2}^{n-1} q \times \log(q) \le \int_2^n x \cdot \log(x)\, dx$$

$$\le \left[\frac{x^2}{2} \times \log(x) - \frac{x^2}{4}\right]_2^n$$

$$\le \left(\frac{n^2}{2} \times \log(n) - \frac{n^2}{4}\right)$$

$$\therefore T(n) \le \left(\frac{2c}{n}\right) \times \left(\frac{n^2}{2}\log(n) - \frac{n^2}{4}\right) + Dn$$

$$\Rightarrow cn \cdot \log(n) - \frac{cn}{2} + Dn$$

Therefore, when $c \ge 2D$

$$T(n) \le cn \cdot \log(n) - \frac{cn}{2} + Dn$$

$$\le cn \cdot \log(n)$$

Because quick sort selects a pivot each time, divides the array into left and right parts, and sorts recursively.

We assume that the size of the left subarray after partitioning is “q”, and “D” is a constant of Θ(n).

So if a constant number C is larger than 0, therefore K < n, T(k) <= ck*log(k)

If we pick n=2:

The T(2) = 2/2[T(0) + T(1)] + θ (2) => θ (1)

Which means that if θ (1) <= 2c * log2 , so C >= θ (1) / 2

At the end of the function, we need to prove that the

cn * log(n) - cn/2 + Dn + O(1) <= cn*log(n)

So

-cn/2 + Dn + O(1) <= 0

When C >= 2D:

n*(D-c/2) <= 0

Therefore : T(n) <= cn*log(n)

The average time complexity of quick sort is O(nlogn)

Reference:

1. https://blog.csdn.net/yangtzhou/article/details/105339108 ; yangtzhou;6/4/2020;
2. https://www.cnblogs.com/hithongming/p/9200705.html ; HongmingYou ;
3. https://www.bilibili.com/video/BV1kQ4y1o7jx/?spm_id_from=333.337.search-card.all.click&vd_source=3492fb7dfe3b2952c09c6868f0bced4a ; 2021-05-21 04:04:02 ; 雾漫大武汉
4. https://zhuanlan.zhihu.com/p/267890781 ; 2020-10-24; **王金戈**