Divide-and-conquer: Solving Recurrences & Correctness of Algorithms: Recursion tree, The master method, Master Theorem.

Recursion tree, The master method (EXAM), Master Theorem, Correctness of algorithms.

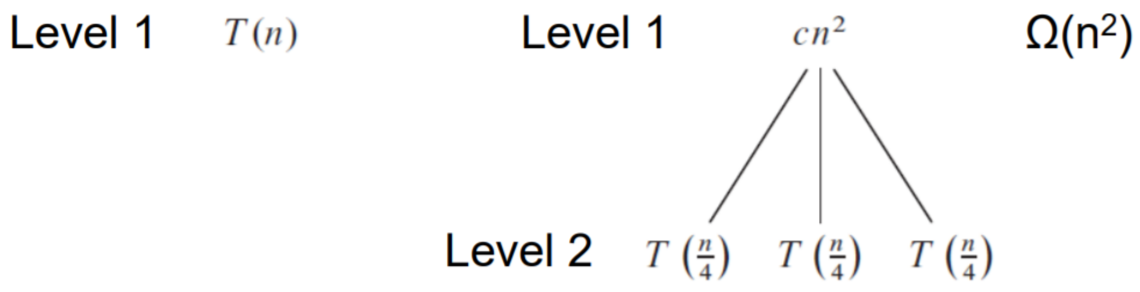
$$\begin{gathered} T(1) = 3 \\ T(n) = T(n-1) + 3n, when\ n > 1 \\ Expand: \\ T(n-1) = T(n-2) + 3(n-1) \\ Substitude: \\ T(n) = T(n-2) + 3(n-1) + 3n \\ Expand: \\ T(n-2) = T(n-3) + 3(n-2) \\ Substitude: \\ T(n) = T(n-3) + 3(n-2) + 3(n-1) + 3n \\ i = T(n-2) + 3 \sum_{k=something}^{n} k \end{gathered}$$

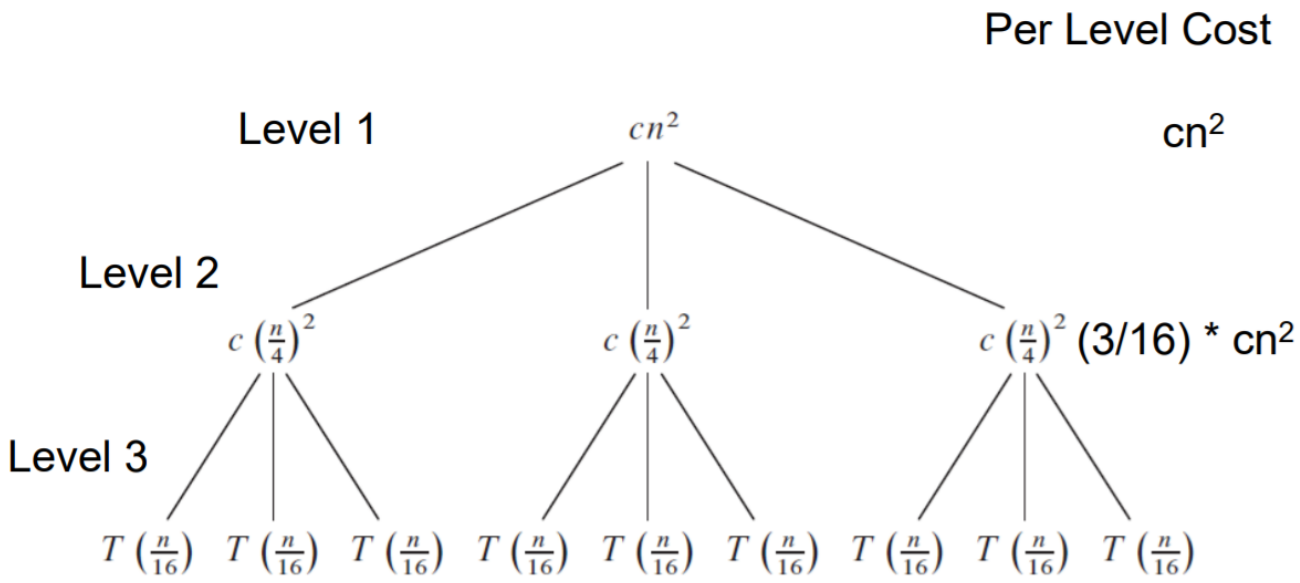
Above example has to be fixed!!

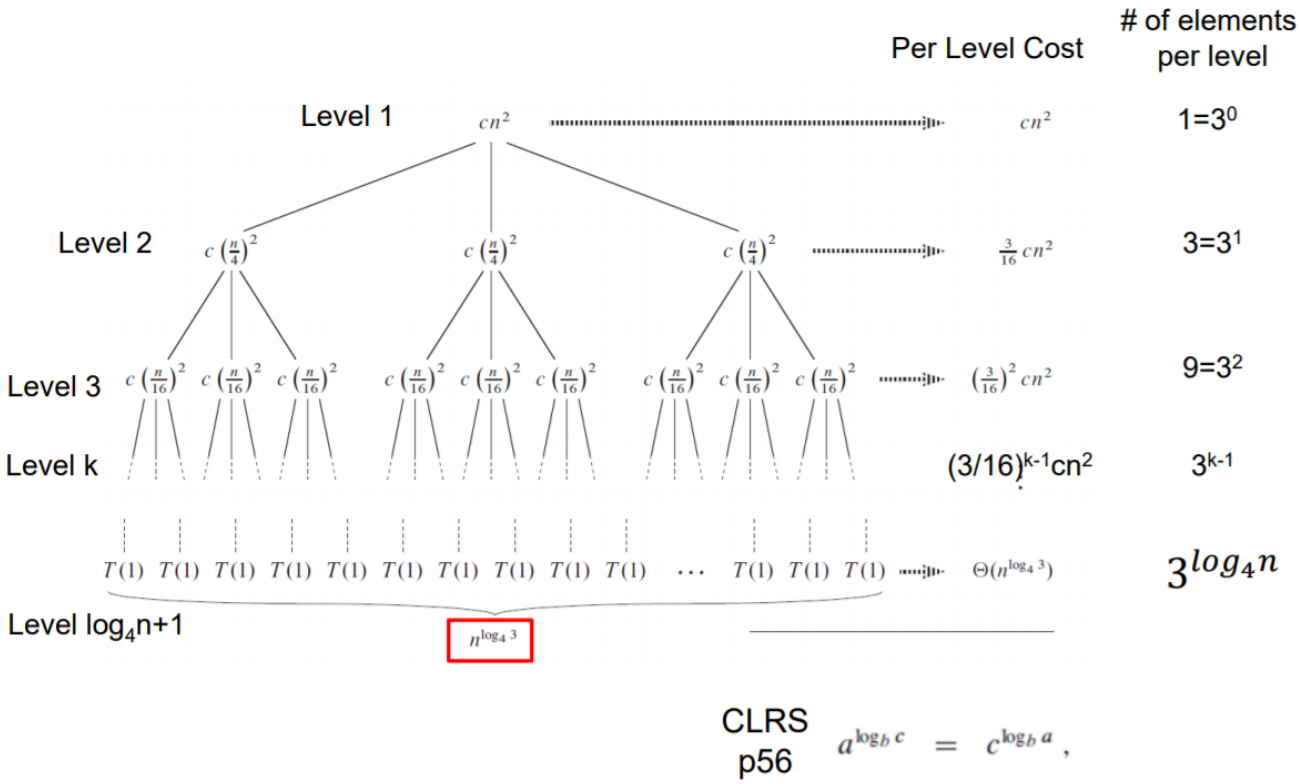
Recursion tree

- A recursion tree is a convenient way to visualize what happens when a recurrence is iterated
- Each **node** represents the cost of a single sub-problem.
- Sum the costs within the each level of the tree to obtain a set of **per-level costs**.
- Sum all the per-level costs to determine the **total cost** of all levels of the recursion.

Example: $T(n) = 3 * T(n/4) + \theta(n^2) = 3 * T(n/4) + c * n^2$







CLRS p56 $a^{\log_b c} = c^{\log_b a}$,

- Sum all the per-level costs to determine the **total cost** of all levels of the recursion.
- T(n) = level 1 cost + level 2 cost + level 3 cost + ... + level $log_4 n$ cost + level $log_4 n + 1$ cost

$$T(n) = cn^2 + \frac{3}{16}cn^2 + \left(\frac{3}{16}\right)^2 cn^2 + \cdots + \left(\frac{3}{16}\right)^{\log_4 n - 1} cn^2 + \Theta(n^{\log_4 3})$$

$$T(n) = \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3})$$

$$< \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3})$$

$$= \frac{1}{1-(3/16)} cn^2 + \Theta(n^{\log_4 3})$$

$$= \frac{16}{13} cn^2 + \Theta(n^{\log_4 3})$$

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}.$$

when |x|<1
CLRS, p. 1147, A.6

$O(n^2)$ Since $\Omega(n^2)$ in the first iteration, thus $\Theta(n^2)$.

Key steps of using recursion trees

- $T(n) = aT(n/b) + D(n) + C(n)$
- How many levels are there in the tree
  - $log_b n + 1$
- What is the cost per non-leaf level?
  - Depends on the cost of divide and combining.
- What is the cost for the leaf level?
  - Depends on how many leave nodes are there: $n^{log_b a}$
  - Each leave node has constant cost.
- Sum the per level costs into the final cost.

Example:

$$Recursion\ tree\ for:\ T(n) = T(n-1) + 3n$$
$$3n \rightarrow 3(n-1) \rightarrow 3(n-2) \rightarrow 3(n-3) \ldots\ 3(n-(i-1))$$

It only has on note on each level because T(n) is not * with anything. *i* is the level. Total levels = n. At level n $T(n) = 3$ base case.

To solve these you can either use recursion trees or repeated substitution method.

The master method (exam)

- The master method provides a template method for solving recurrences **of the form**:
  $T(n) = a * T(n/b) + f(n)$ where $a \geq 1$ and $b > 1$ are constant and f(n) is asymptotically positive.
- T(n) is the runtime for an algorithm and we know that:
- a subproblems of size *n/b* are solved recursively, each in time *T(n/b)*
- *f(n)* is the cost of dividing the problem and combining the results
  - $f(n) = D(n) + C(n)$

Example Merge-sort

- $T(n) = 2 * T(n/2) + \theta(n)$
- a = 2, b = 2: each time we split a problem into a = 2 subproblems and each subproblem is with the half size (b=2) of the original problem.
- $f(n) = \theta(n)$
- Dividing: $\theta(1)$
- Combining: Merge() function is $\theta(n)$

Master Theorem

- Recurrence in the form of $T(n) = a * T(n/b) + f(n)$
- First case: if $f(n) = O(n^{log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \theta(n^{log_b a})$
- Second case: if $f(n) = \Theta(n^{log_b a})$, then $T(n) = \theta(n^{log_b a} * lg * n)$
- Third case: if $f(n) = \Omega(n^{log_b a + \epsilon})$ for some constant $\epsilon > 0$, and the regularity condition is also satisfied, then $T(n) = \Theta(f(n))$.
    - Regularity condition: $a * f(n/b) \leq c * f(n)$ for some constant $c < 1$ and all sufficiently large n.

How to use the master method

- Extract a, b, and f(n) from a given recurrence
- Determine $n^{log_b a}$
- Compare f(n) and $n^{log_b a}$ asymptotically
    - f(n) increases polynomially slower, case 1
    - They increases similarly, case 2
    - f(n) increases polynomially faster, case 3
- Determine appropriate MM case, and apply.

Example: merge-sort

- $T(n) = 2 * T(n/2) + \theta(n)$
- a=2, b=2, f(n)=Θ(n)
- $n^{log_b a} = n^1 = n = \Theta(n)$
- And also, f(n)=Θ(n)
- Thus, case 2, $T(n) = \Theta(n * lg * n)$

Examples of master method

```
Binary-search(A, p, r, s):
    q←(p+r)/2
    if A[q]=s then return q
    else if A[q]>s then
         Binary-search(A, p, q-1, s)
    else Binary-search(A, q+1, r, s)
```

$$T(n) = T(n/2) + 1$$
$$a = 1,\ b = 2;\ n^{log_2 1} = 1$$
$$also\ f(n) = 1,\ f(n) = \Theta(1)$$
$$\rightarrow Case\ 2:\ T(n) = \Theta(lg * n)$$

- T(n)=3T(n/4) + n lgn
- a=3, b=4, $n^{\log_b a}$ = $n^{0,793}$
- f(n)=n lgn = Ω (n) = Ω ($n^{0.793+\varepsilon}$), where, e.g., ε=0.2
- Thus, case 3. But we have to check the regularity condition
  - a*f(n/b)≤c*f(n) for some constant c<1 and all sufficiently large n
- What is f(n)? f(n)=nlgn
- 3*f(n/4)=3 (n/4) lg (n/4)
- Since n/4 ≤ n for n≥0, and lg is non-decreasing, we have lg (n/4) ≤ lg (n)
- 3 (n/4) lg(n/4) ≤ 3 (n/4) lgn = (¾) nlgn = c*f(n) where c = ¾
- Thus, we can apply case 3.
- T(n)= Θ (nlgn)
- T(n)=3*T(n/4) + n^2
- Which case should be applied?

- a=3, b=4, f(n)=n^2
- $n^{\log_b a}$ = $n^{0,793}$
- f(n)=n^2 = Ω ($n^{0.793+\varepsilon}$), where, e.g., ε=0.3
- Thus, case 3. Check the regularity condition
  - a*f(n/b)≤c*f(n) for some constant c<1 and all sufficiently large n
- 3*f(n/4)=3 $(n/4)^2$ =3/16 n^2
- Any 3/16<c<1, e.g., c=0.5, would make 3*f(n/4) ≤ c*f(n)
- Thus, we can apply case 3. T(n)= Θ (n^2)

- So far, we have seen how to use 3 different methods to solve this recurrence.

- T(n)=2T(n/2) + nlgn
- a=2, b=2, $n^{\log_b a}$ = n
- f(n)=nlgn
- Should case 3 apply here?
- No, because we cannot find some positive constant ε such that f(n)=nlgn=Ω ($n^{1+\varepsilon}$)
  - f(n) increases faster, but not ***polynomially*** faster.

- NOTE: master method cannot be applied on this recurrence, even though it looks like the template T(n)=a*T(n/b)+f(n)

- However, an extension to the master method can be used. See exercise 4.6-2.

Solving recurrences

- If a recurrence is in the form that can be solved by the master method, try to use the master method, because it is the simplest method.
  - But, when using the master method, if you are not sure about deciding which case should be applied, especially for checking the regularity condition in case 3, use the repeated substitution method or recursion tree instead.
- If a recurrence is not in the form that can be solved by the master method, use the repeated substitution method or recursion trees.
  - E.g., computing the factorial.
- You can always use the repeated substitution method or recursion trees to solve any recurrence.

Mini quiz

- Solving the recurrence $T(n) = 9T(n/3) + n$
- **Master Method**:
  - $T(n) = 9T(n/3) + n$
  - $a = 9,\ b = 3,\ log_b a = log_3 9 = 3$, so n^2
  - $f(n) = n = O(n^{2-\varepsilon})$ where e.g., $\varepsilon = 0.5$
  - So case 1 applies. $T(n) = \Theta(n^2)$
- **Repeated Substitution Method**:

- $T(n)=9T(n/3)+n$
- $=9(9T(n/3^2)+n/3)+n= 9^2T(n/3^2)+(3n+n)$
- $=9^2(9T(n/3^2)+n/3^2)+(3n+n)=9^3T(n/3^3)+(3^2n+3n+n)$
- $= 9^iT(n/3^i)+ n \sum_{k=0}^{i-1} 3^k$
- $= 9^iT(n/3^i)+ n (3^i-1)/2$
- To use the base case, we make $n/3^i=1$, thus $i=\log_3 n$
- $T(n)=9^{log_3^n} + n(3^{log_3^n}-1)/2$
- $= n^2+ n (n-1)/2$
- $\Theta(n^2)$

Geometric series

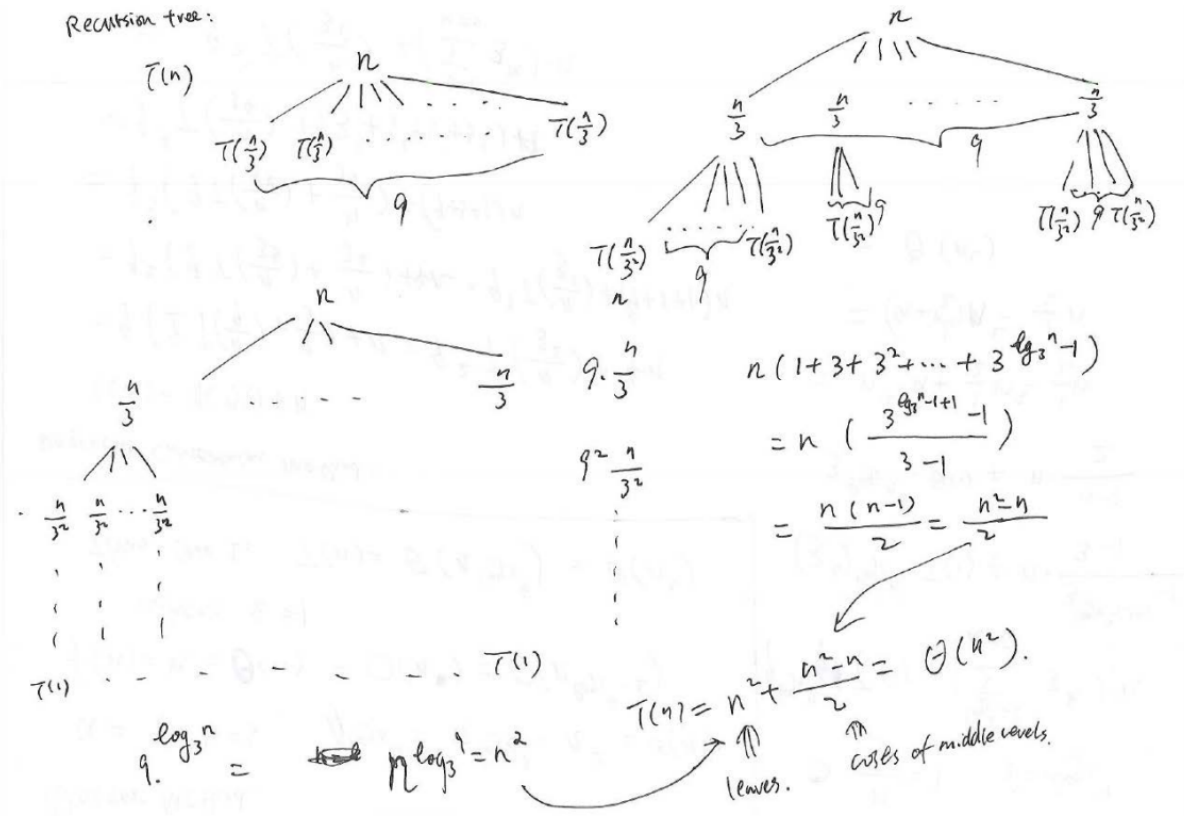
For real $x \neq 1$, the summation

$$\sum_{k=0}^{n} x^k = 1 + x + x^2 + \cdots + x^n$$

is a ***geometric*** or ***exponential series*** and has the value

$$\sum_{k=0}^{n} x^k = \frac{x^{n+1} - 1}{x - 1} .$$

Recursion Tree



Analysing D&C Algorithms

- Write down the recurrence
  - How many subproblem? What is the size of each subproblem?
- Solve the recurrence

- If the recurrence is in the form of T(n)=a*T(n/b)+f(n), try the master method.
  - If you can use master method, use the master method.
  - Otherwise, go for the repeated substitution method or using a recursion tree.
    - Not polynomially faster or slower in case 1 or case 3.
    - Regularity condition cannot be satisfied in case 3.
- Repeated substitution method and recursion trees can be applied to any recurrence, no matter what form the recurrence is.
  - Generic solutions.
  - I personally like the repeated substitution method, because it is much easier to make mistakes when drawing a recursion tree.

Correctness of Algorithms

- The algorithm is **correct** if for any legal input it terminates and produces the desired output.

Loop invariants

- **Invariants** – assertions (i.e, statements about the states of the execution) that are valid any time they are reached (many times during the execution of an algorithm, e.g., in loops)
- We must show three things about loop invariants:
  - **Initialization** – it is true prior to the first iteration
  - **Maintenance** – if it is true before an iteration, *then* it remains true before the next iteration
  - **Termination** – when loop terminates the invariant gives a useful property to show the correctness of the algorithm

Example for insertion sort

- Invariant: at the start of each for loop, A[1…j-1] consists of elements originally in A[1…j-1] but in sorted order

```
for j = 2 to n
    key = A[j]
    // Insert A[j] into the sorted A[1..j-1].
    i=j-1
    while i>0 and A[i]>key
        A[i+1]=A[i]
        i--
    A[i+1]=key
```

- Initialization: j = 2, the invariant trivially holds because A[1] is a sorted array
- Invariant: at the start of each for loop, A[1…j-1] consists of elements originally in A[1…j-1] but in sorted order
- Maintenance: the inner while loop moves elements A[j-1], A[j-2], …, A[j-k] by one position to the right without changing their order until it finds the proper position for A[j]. Then, A[j] is inserted into k-th position such that A[k-1] ≤ A[k] ≤ A[k+1]. Thus, A[1…j] consists of the elements originally in A[1…j] but in sorted order.

- Invariant: at the start of each for loop, A[1...j-1] consists of elements originally in A[1...j-1] but in sorted order
- Termination: the loop terminates, when j=n+1. Then the invariant states: "A[1...n] consists of elements originally in A[1...n] but in sorted order"

b) (*6 points*)
Mark the correct answer by putting a cross in the corresponding field.

Consider the recurrence (for simplicity, assume that n is a power of 3).
$T(1) = 1$
$T(n) = 4T(\frac{n}{3}) + \frac{1}{2}n^2$.
$T(n)$ is

☐ $\Theta(n^3)$ ☐ $\Theta(n^{log_3 4})$ ☐ $\Theta(n^{log_3 4} \log n)$ ■ $\Theta(n^2)$

Master Method case 3:
Regularity condition *a*f(n/b)≤c*f(n)*
satisfies when e.g., c=6/9