DIVIDE AND CONQUER ALGORITHMS

Jargons:

- "Recurrence relation" : time complexity of a recursive algorithm.

Recurrence Relation:

A recurrence relation usually take the form:

$$T(n) = \begin{cases} O(1) & \text{if base case satisfied} \\ a \cdot T\left(\frac{n}{b}\right) + f(n) & \text{otherwise} \end{cases}$$

recursive term: $a \cdot T\left(\frac{n}{b}\right)$ — overhead term (represents all overheads): $f(n)$

where
- n : problem's original size
- $a \geq 1$: number of subproblems
- $b > 1$: factor by which the original size reduced to.
- $f(n)$: any over cost (sorting, merging, etc...)

Example: Determine the asymptotic complexity of the function defined in the given recurrence relation:

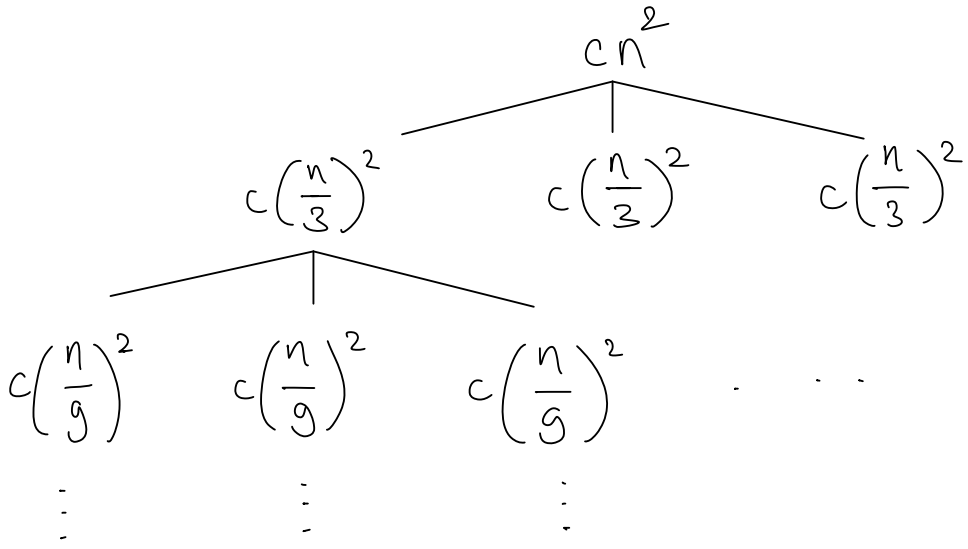
$$T(n) = 3T\left(\frac{n}{3}\right) + cn^2$$
$$T(1) = c$$

If the algorithm is exponential then you only need to provide exponential lower bounds.

Solution:

- We try to bound this using analyzing recursion tree method:

size each level	: Overheads each level:
n	cn^2
$n/3$	$3c\left(\frac{n}{3}\right)^2 = \frac{1}{3}cn^2$
$n/9$	$9c\left(\frac{n}{9}\right)^2 = \frac{1}{9}cn^2$
⋮	⋮
$\Rightarrow \frac{n}{3^i}$	$\Rightarrow \left(\frac{1}{3}\right)^i cn^2$
where i is the level	where i is the level



- Height of the tree: $\frac{n}{3^i} = 1$ → base case

∴ interpret as: height of the tree = number of times n divides by b to reach base case (1 in this case) ∴

- Total costs of the algorithm:

total overheads

total cost at Leaves

$$T(n) = \sum_{i=0}^{h} cn^2 \left(\frac{1}{3}\right)^i + 3^d$$

<removed be its not the dominance term>

$$= cn^2 \sum_{i=0}^{\infty} \left(\frac{1}{3}\right)^i$$

$$= cn^2 \left(\frac{1 - (1/3)^h}{1 - 1/3}\right)$$

<geometric series>
↳ mentioned later in last page

Upper-bound:

$$T(n) = cn^2 \left(\frac{1 - (1/3)^h}{1 - 1/3}\right) \leq cn^2 \left(\frac{1 - (1/3)^{\infty}}{1 - 1/3}\right)$$

$$= cn^2 \left(\frac{1}{1 - 1/3}\right)$$

$$= \frac{3}{2} cn^2$$

$$= O(n^2)$$

Lower-bound:

$$T(n) = cn^2 \left(\frac{1 - (1/3)^h}{1 - 1/3}\right) \geq cn^2 \cdot (1)$$

(1) → first term of geometric series

choose first term because its the dominance term

$$= cn^2$$

$$= \Omega(n^2)$$

Conclusion:

$$T(n) = \Theta(n^2)$$

Analyzing recursion tree method:

Problem: Given a recurrence relation, determine asymptotic complexity

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

$$T(1) = O(1)$$

Steps:

1. Draw out recursion tree (optional)
2. Find expression for "Total overhead costs at all levels"
3. Find expression for "Total costs at leaves" (can skip if the operations performed at leaves is $O(1)$)
4. Determine the upper-bound and lower-bound of this expression:

$T(n)$ = Total overheads + Total costs at leaves

Recursion Tree: useful equations

These hold true with: $\begin{cases} T(n) = a\,T(n/b) + f(n) \\ T(1) = O(1) \end{cases}$

- Overhead term's complexity

$$f(n) = \Theta\left(n^c \log_b^k n\right)$$

- Depth of the tree:

$$d = \log_b n$$

Note: depth = height - 1

- Number of leaves:

$$l = a^{\log_b n}$$

$$\Rightarrow \text{Observe: } a^{\log_b n} = \left(b^{\log_b a}\right)^{\log_b n} = \left(b^{\log_b n}\right)^{\log_b a} = n^{\log_b a}$$

- Total overheads at height i:

$$\text{overheads} = \left(\frac{a}{b^c}\right)^i \cdot n^c \quad \text{where } 0 \leqslant i \leqslant \log_b n$$

Geometric sequence

A geometric sequence ratio r of h terms is defined as

$$S_h = \sum_{i=0}^{h} \left(\frac{1}{r}\right)^i \qquad \text{Note: } r \neq 1$$

$$= r^0 \left(\frac{1 - r^h}{1 - r}\right)$$

r^0: first term, use for lower-bound because its the dominance term

Master Theorem

is a tool used to categorize divide-and-conquer algorithms of the form:

$$T(n) = a\,T\left(\frac{n}{b}\right) + n^c$$

where $a \geq 1$, $b > 1$, $c \geq 0$

Then:

$$T(n) \begin{cases} \Theta(n^c) & \text{if } a < b^c \\ \Theta(n^c \log_b n) & \text{if } a = b^c \\ \Theta(n^{\log_b a}) & \text{if } a > b^c \end{cases}$$

↳ Proof:

- Case 1: $a = b^c$ (balanced)
  - Overhead term: $n^c = n^{\log_b a}$ $\quad \langle a = b^c \Leftrightarrow c = \log_b a \rangle$
  - Recursive term:

$$\begin{aligned} T(n) &= \text{Total overheads} + \text{Total leaf operations} \\ &= \sum_{i=0}^{\log_b n} \left(\frac{a}{b^c}\right)^i \cdot n^c + n^{\log_b a} \\ &\sim \sum_{i=0}^{\log_b n} 1^i \cdot n^c \quad \langle \text{first term dominate} \rangle \\ &= \log_b n \cdot n^c \end{aligned}$$

$\log_b n$: Tree's depth $\Rightarrow$ $T(n) = \Theta(\log_b n \cdot n^c)$

  - Conclusions:
    - Overheads at each level is the same
    - Both recursive calls and overheads contributed to the cost
  - Examples: Merge sort

- **Case 2: $a < b^c$** (root-heavy)
  - Overhead term: $n^c > n^{\log_b a}$ ($a < b^c \Leftrightarrow \log_b a < c$)
  - Recursive term:

$$T(n) = \text{Total overheads} + \text{Total leaf operations}$$

$$= \sum_{i=0}^{\log_b n} \left(\frac{a}{b^c}\right)^i \cdot n^c + n^{\log_b a}$$

$$= n^c + \underbrace{\sum_{i=1}^{\log_b n} \left(\frac{a}{b^c}\right)^i n^c}_{< n^c} + \underbrace{n^{\log_b a}}_{< n^c}$$

$$\sim n^c$$

$$\Rightarrow T(n) = O(n^c)$$

  - Conclusions:
    - Total overheads is dominated by root's overhead
    - Examples: Some linear-time algorithms. ($a=1, b=2, c=1$)

Case 3: $a > b^c$ (leaf-heavy)

- Overhead term: $n^c < n^{\log_b a}$ ($a > b^c \Leftrightarrow \log_b a > c$)
- Recursive term:

$$T(n) = \text{Total overheads} + \text{Total leaf operations}$$

$$= \sum_{i=0}^{\log_b n - 1} \left(\frac{a}{b^c}\right)^i n^c + n^{\log_b a}$$

$$= n^c \left(\frac{1 - \left(\frac{a}{b^c}\right)^{\log_b n - 1}}{1 - \left(\frac{a}{b^c}\right)} \right) + n^{\log_b a}$$

finite geometric series $\in [0, (\log_b n - 1)]$

$$= n^c \cdot O(1) + n^{\log_b a}$$

$$\sim n^{\log_b a} \quad (\text{since } \log_b a > c)$$

$$\Rightarrow T(n) = \Theta(n^{\log_b a})$$

- Conclusions:
  - Total overheads is dominated by the term associated with the depth $\log_b n - 1$

Maximum Subarray:

Problem:

Given array $A = [a_0, a_1, \dots, a_{n-1}]$ where a_i is integer, find subarray $A[i:j]$ with the largest sum.

Brute force:

Pseudo for brute force:

```
for i: 0 → (n-1):
    for j: i → (n-1):
        res = max{res, A[i:j]}
```

- Can easily see that this would cost $O(n^3)$ – 2 loops, 1 sum calculation
- We could optimize this slightly by keep a record of "current sum" and add element $A[j]$ for each iteration, that would reduce the cost to $O(n^2)$

⇒ But, we can do better

Divide-and-Conquer

The idea is given arbitrary subarray $A[low:high]$, the maximum subarray $A[i:j]$ of $A[low:high]$ can only be in 3 places:

- Left half of $A[low:high]$, particularly $A[low:mid]$
- Right half of $A[low:high]$, particularly $A[mid+1:high]$
- At the crossing, i.e $A[i:j]$ where $\begin{cases} low \le i < mid \\ mid \le j < high \end{cases}$

This means we can solve the problem recursively by solving the subproblems

Pseudo for Divide-and-Conquer:

```
FIND_MAX_SUBARRAY (A, low, high)
    if low ≥ high:
        return (low, high, A[low])      # base case: only 1 element

    mid = (low + high) // 2

    left_sum, left_idx := -∞, mid
    curr_sum := 0
    for i : mid → 0 :   (loop backwards)
        curr_sum += A[i]
        if curr_sum > left_sum:
            left_sum := curr_sum
            left_idx := i
    right_sum, right_idx := -∞, mid+1
    curr_sum := 0
    for j : (mid+1) → (n-1):
        curr_sum += A[j]
        if curr_sum > right_sum:
            right_sum := curr_sum
            right_idx := j

    res = arg max (i,j, A[i:j]) { FIND_MAX_SUBARRAY (A, low, mid),   left-half
                                  FIND_MAX_SUBARRAY (A, mid+1, high), right-half
                                  (left_sum + right_sum) }            crossing

    return res
```

find sum and indices at crossing, cost $O(n)$

Recurrence relation:

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ \underbrace{2 \cdot T(n/2)}_{\text{2 subproblems of half size (left and right)}} + \underbrace{O(n)}_{\text{cost of calculating the crossing subarray}} & \text{if } n > 1 \end{cases}$$

Performance:

From master theorem, this is the second case ($a = b^c$). So the performance is $\Theta(n^c \log_b n) = \Theta(n \log_2 n)$

Strassen's method

Recall how matrix multiplication is done, for matrices $A, B, C \in \mathbb{R}^{n \times n}$

$$C = A \cdot B$$

Each element in matrix C can be expressed as:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

$\Rightarrow$ Cost $\Theta(n^3)$

Divide-and-Conquer matrix multiplication:

Now if we try multiplying matrices recursively.

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \cdot \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

Then the calculations we need to perform per recursive loop is:

$$\begin{cases} C_{11} = A_{11} B_{11} + A_{12} B_{21} \\ C_{12} = A_{11} B_{12} + A_{12} B_{22} \\ C_{21} = A_{21} B_{11} + A_{22} B_{21} \\ C_{22} = A_{21} B_{12} + A_{22} B_{22} \end{cases}$$

The recurrence relation is:

$$T(n) = 8\,T(n/2) + \Theta(n^2)$$

↳ explanation: To get the result ($C_{11}, C_{12}, C_{21}, C_{22}$):

- we solve 8 sub-problems
- each sub-problem is half the size of the original problem
- after we got the result from the sub-problems, we perform 4 additions, of 2 matrices size $n/2 \times n/2 \Rightarrow n^2/4$ additions which means $\Theta(n^2)$

By master theorem, this is the third case $a > b^c$:

$\Rightarrow$ cost is $\Theta(n^{\log_b a}) = \Theta(n^3)$

Strassen approach

The idea is to reduce the number of sub-problems from 8 to 7, hence improve the complexity. The four partitions of matrix C :

$$\begin{cases} C_{11} = P + S - T + V \\ C_{12} = R + T \\ C_{21} = Q + S \\ C_{22} = P + S - Q + U \end{cases}$$

where :

$$\begin{cases} P = (A_{11} + A_{22})(B_{11} + B_{22}) \\ Q = (A_{21} + A_{22}) B_{11} \\ R = A_{11}(B_{12} - B_{22}) \\ S = A_{22}(B_{21} - B_{11}) \\ T = (A_{11} + A_{12}) B_{22} \\ U = (A_{21} - A_{11})(B_{11} + B_{12}) \\ V = (A_{12} - A_{22})(B_{21} + B_{22}) \end{cases}$$

$$B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$ (S1 encloses B_{12} and B_{22})

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

The recurrence relation is now :

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 7\,T(n/2) + \Theta(n^2) & \text{if } n > 1 \end{cases}$$

By master theorem, this is the third case $a > b^c$, but now the cost is :

$$\Theta(n^{\log_b a}) = \Theta(n^{\log_2 7}) \simeq \Theta(n^{2.81})$$

which is a slight improvement

BFPRT Algorithm (Median of Medians algorithm)

- Goal: find smallest k^{th} element in unsorted array
- Intuition: Given unsorted array A

SELECT(A, k)

$A = [\underbrace{a_0, a_1}_{g_1}, \underbrace{a_2, \dots}_{g_2}, \underbrace{a_{n-2}, a_{n-1}}_{g_3}]$

Sorted & calculate median

median 1, median 2, median 3

a_m = median of medians

partition

$A_{partition} = [\dots A_{<m}, a_m, \dots A_{>m}]$

recursive

$k \in |A_{<m}|$	$k = |A_{<m}| + 1$	$k \in |A_{>m}|$
recursive on $(A_{<m}, k)$	return $A[k]$	recursive on $(A_{>m}, k - (|A_{<m}| + 1))$

base case

```
if |A_base| = 1:
    return A_base[0]
if |A_base| = 0:
    return ∅
```

Time Complexity:

Total complexity = Find median + Partition + Recursive

Lets go through each component, we are assuming splitting A into $(n/5)$ subarrays of size 5.

- Partition: $O(n)$
- Recursive: $T(0.7n)$

  The claim is that a "good pivot" will satisfy this condition: $|A_{left}|, |A_{right}| \leq 0.7n$

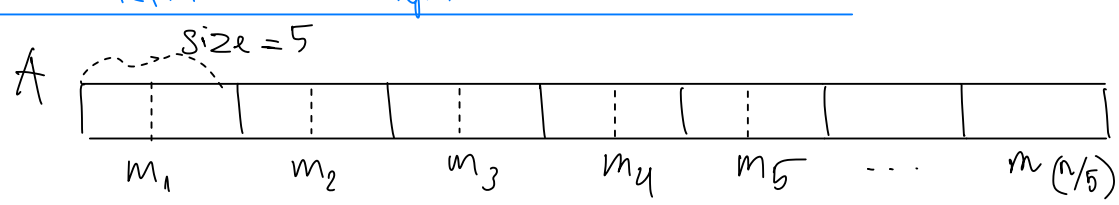
- Find median: $T(n/5) + O(n)$

where
- $T(n/5)$ since we dividing A into $n/5$ subarrays of 5 elements
- $O(n)$ is for finding the median of medians

Hence, combined we have the time complexity of BFPRT algorithm:

$$T(n) \leq T(n/5) + T(0.7n) + O(n)$$

Prove $|A_{left}|, |A_{right}| \leq 0.7n$



pivot x = median $\{m_1, m_2, m_3, m_4, m_5, \ldots m_{(n/5)}\}$

Key insight:
- if $m_i \leq x$, then the group whose m_i is the median contributes at least 3 elements to A_{left}
- if $m_i > x$, then the group whose m_i is the median contributes at least 3 elements to A_{right}

Since x is "median of medians", and we have $(n/5)$ medians, meaning there are $(n/10)$ medians on each side of x, so:

$$\underbrace{\{m_1, m_2, \ldots, m_{(n/10)}\}}_{\sim |A_{left}|} \cup \{x\} \cup \underbrace{\{m_{(n/10+2)}, m_{(\frac{n}{10}+3)}, \ldots m_{(n/5)}\}}_{\sim |A_{right}|}$$

From the insight above, we can lower bound the size of partitions:

$$|A_{left}|, |A_{right}| \geq \frac{3n}{10} \approx 0.3n$$

And then we can derive the upper bound since $|A_{left}| + |A_{right}| = n$

$$|A_{right}|, |A_{left}| \leq 0.7n$$