DIVIDE & CONQUER PARADIGM

- Solving Recurrences

A) Divide the problem into subproblems
B) Solve the subproblems recursively or combine
C) Merge the solutions to subproblems to solve the larger problem.

A canonical example here is the so-called Merge Sort which uses this technique to sort n numbers using $O(n \log n)$ comparisons; each comparison is of the sort, is $a_i < a_j$? where $a_1, a_2, ..., a_n$ are the given numbers.

Often step C is the key to the whole algorithm! [Sometimes, step A might also require some creativity.] Let's first recall/see how step C can be done for MergeSort.
Illustration of "Merge": sorted lists.

Example: Combining two sorted lists, with distinct elements (say).

1 6 10 12
↑
L

2 3 7 8
↑
R

Imagine a left pointer L and a right pointer R which will move to the right, based on the outcome of the comparisons.

They both start at the leftmost on their lists.

At each step, if we compare the elements at the pointers L & R and the smaller element is added to the final combined list and the corresponding pointer moves one step to the right. (The other pointer stays put.)

So at each step one element is added to the combined list and at the very end, the last remaining elements of one remaining list are added to the list. \[ \text{See } \]
Start

1. 6 10 12

2. 3

Compare: 1 < 2?

Combined List

6 10 12 3 7 8

Compare: 6 < 2?

Step 2:

12

Step 3:

6 10 12 3 7 8

Compare: 6 < 3?

123

Step 4:

6 10 12 7 8

6 < 7?

1236

Step 5:

10 12 7 8

10 < 7?

12367
Step 6

\[ 10 \quad 12 \quad 8 \quad L \quad R \]

Compare \(10 \leq 8\)?

Combined List

\[ 1 \quad 2 \quad 3 \quad 6 \quad 7 \quad 8 \]

Now that one list is empty, we can simply attach the remaining list to the combined list!

Final: \[ 1 \quad 2 \quad 3 \quad 6 \quad 7 \quad 8 \quad 10 \quad 12 \]

Qn.: How many comparisons total?

Ans.: At each step one of the pointers moves one place; after one comparison we stop in the worst case, when all but one element is left to be placed on the combined list.

So if the total #elements is \(n\), then the answer is \(n-1\); since we place one element each time a pointer moves.
Merge Sort Analysis

The algorithm is described in Pages 50-51 of the textbook by Dasgupta et al.

\text{Merge Sort} (a[1..n])

\text{Input:} \ array \ of \ n \ numbers: \ a[1..n]
\text{Output:} \ sorted \ array

If \ n \gt 1:
\hspace{1em} \text{return} \ \text{merge} \ (\text{merge sort} \ (a[1..n/2]),
\hspace{1em} \text{merge sort} \ (a[n/2+1..n]))

else:
\hspace{1em} \text{return} \ \text{array} \ a.

The key is the "merge" operation explained before.

Let \ \( T(n) \) = \# comparisons done by Merge Sort. Then we have,

\[
\begin{align*}
T(2) &= 1, \quad \text{and} \\
T(n) &\leq 2T\left(\frac{n}{2}\right) + (n-1),
\end{align*}
\]

Let us simplify by assuming \( n = 2^k \), \( k \geq 1 \).

and replacing \( (n-1) \) by \( n \), so that we get a cleaner recurrence.

\[
T(n) = 2T\left(\frac{n}{2}\right) + n
\]

\[
= 2\left[2T\left(\frac{n}{4}\right) + \frac{n}{2}\right] + n
\]

\[
= 2^2T\left(\frac{n}{2^2}\right) + n + n
\]
\[ T(n) = 2^2 \left[ T\left(\frac{n}{2^3}\right) + \frac{n}{2^2} \right] + 2n \]

\[ = 2^3 \left[ T\left(\frac{n}{2^3}\right) \right] + n + 2n = 3n \]

Repeating it, we get

\[ T(n) = 2^{k-1} \left[ T\left(\frac{n}{2^{k-1}}\right) \right] + (k-1)n. \]

Recall \( n = 2^k \), so that \( T\left(\frac{n}{2^{k-1}}\right) = T(2)=1. \)

So \( T(n) = 2^{k-1} + (k-1)n. \)

What is \( k? \) \( 2^k=n \Rightarrow k=\log_2 n. \)

So \( T(n) = \frac{n}{2} + (\log_2 n - 1) n = n\log_2 n - \frac{n}{2} \)

\[ = \Theta(n\log n) \quad \square. \]

\[ \textit{More recurrences}; \]

\[ \textit{First recall some basics}; \]

Exercise 0.2. Let \( c>0. \)

Then \( 1 + c + c^2 + \cdots + c^n \) is

\[ \text{(a) } \Theta(1) \quad \text{if } c<1 \]

\[ \text{(b) } \Theta(n) \quad \text{if } c=1 \]

\[ \text{(c) } \Theta(c^n) \quad \text{if } c>1. \]
Recall: geometric series:

\[ 1 + c + c^2 + \cdots = \frac{1}{1-c} \quad \text{if} \quad |c| < 1. \]

and

\[ 1 + c + c^2 + \cdots + c^n = \frac{1 - c^{n+1}}{1-c} = \frac{c^{n+1}}{c-1}, \quad \text{if} \quad c \neq 1. \]

To check the above, simply cross multiply

\[(1 + c + c^2 + \cdots + c^n)(1-c) = 1 - c^{n+1} \]

\[\Rightarrow 1 + c + c^2 + \cdots + c^n - (c + c^2 + \cdots + c^{n+1}) = 1 - c^{n+1}. \checkmark\]

So a) if \(c < 1\) then

\[ 1 + c + c^2 + \cdots + c^n = \frac{1-c^{n+1}}{1-c} = O(1), \]

since \(c^{n+1}\) is exponentially small, for \(c < 1\) and \(1-c\) is a constant.

b) obvious

c) if \(c > 1\) then

\[ \frac{c^{n+1} - 1}{c - 1} = O(c^{n+1}) = O(c^n), \]

since \(c\) is a constant.
(II) Also continuing with basics:

\[
\begin{align*}
2 \log_2 n &= n \quad \text{and} \quad 3 \log_2 3 = n \\
\log_a n &= \frac{\log n}{\log a} \\
\log_{a^b} n &= \frac{\log n}{\log a}.
\end{align*}
\]

\[
(\log_a n)(\log_2 a) = \log_2 n
\]

\[
\frac{\log_2 n}{\log_a n} = \log_a 2
\]

(III) Example Recurrences

(i) \( T(n) = T\left(\frac{n}{2}\right) + 1 \)

\[
= \left[ T\left(\frac{n}{4}\right) + 1 \right] + 1 = T\left(\frac{n}{4}\right) + 3
\]

\[
= \left[ T\left(\frac{n}{8}\right) + 1 \right] + 3 = T\left(\frac{n}{8}\right) + 5
\]

\[\ldots\]

\[
= T\left(\frac{n}{2^k}\right) + k,
\]

\[
= T(1) + k, \text{ if } n = 2^k,
\]

\[
= O(\log n), \text{ since } T(1) = O(1).
\]
(ii) \[
T(n) = 3 \cdot T\left(\frac{n}{2}\right) + \Omega(n), \quad (\text{Say}),
\]

\[
T(2) = 3; \text{ let } n = 2^k; k \geq 1.
\]

\[
T(m) = 3 \cdot T\left(\frac{n}{2}\right) + n
\]

\[
= 3 \left[3 \cdot T\left(\frac{n}{4}\right) + \frac{n}{2}\right] + n
\]

\[
= 3 \cdot \left[3 \cdot T\left(\frac{n}{2^2}\right) + \frac{n}{2^2}\right] + \frac{3}{2}n + n
\]

\[
= 3^2 \cdot T\left(\frac{n}{2^3}\right) + \left(\frac{3}{2}\right)^2n + \frac{3}{2}n + n
\]

So continuing this way (or formally by induction on \(m\), etc.),

\[
T(n) = 3^{k-1} \cdot T\left(\frac{n}{2^{k-1}}\right) + \left(\frac{3}{2}\right)^{k-2}n + \left(\frac{3}{2}\right)^{k-3}n + \ldots + \frac{3}{2}n + n
\]

Now

\[
T\left(\frac{n}{2^{k-1}}\right) = T\left(\frac{n}{2^k}\right) = 3^n.
\]

So

\[
3^{k-1} \cdot T\left(\frac{n}{2^{k-1}}\right) = 3^k.
\]

\[
\left[\left(\frac{3}{2}\right)^{k-2} + \left(\frac{3}{2}\right)^{k-3} + \ldots + \frac{3}{2}\right]n = O\left(\left(\frac{3}{2}\right)^{k}n\right) \quad \text{and} \quad O\left(3^k\right),
\]

since \(n = 2^k\).

So

\[
T(m) = O(3^n) = O\left(3^{\log_2 n}\right) = O(n^{\log_2 3}n).
\]

\[\square\]
(111) \[ T(n) = T\left(\frac{3}{4}n\right) + O(n) = n, \text{ say.} \]

\[ = T\left(\left(\frac{3}{4}\right)^2 n\right) + \frac{3}{4}n + n \]

\[ = T\left(\left(\frac{3}{4}\right)^3 n\right) + \left(\frac{3}{4}\right)^2n + \left(\frac{3}{4}\right)n + n \]

\[ = T\left(\left(\frac{3}{4}\right)^k n\right) + \left[\left(\frac{3}{4}\right)^{k-1} + \left(\frac{3}{4}\right)^{k-2} + \ldots + 1\right]n. \]

We want to stop when \( \left(\frac{3}{4}\right)^k n = 1 \) say.

Then \( n = \left(\frac{4}{3}\right)^k \) \( \Rightarrow k = \log_{4/3} n \).

Also note that \( k = \left\lceil \log_{4/3} n \right\rceil, \text{ an integer} \)

\( \left(\frac{3}{4}\right)^{k-1} + \left(\frac{3}{4}\right)^{k-2} + \ldots + \left(\frac{3}{4}\right)+1 = O(1), \text{ since } \frac{3}{4} < 1. \)

(Recall geometric series, with \( c < 1 \).)

Hence \( T(n) = T(1) + O(n) = O(n) \).

This comes up in the Median search algorithm by divide and conquer strategy.
\[ T(n) = 7 T \left( \frac{n}{2} \right) + n^2, \quad n \geq 2. \]
\[ T(1) = 1 \]

Let \( n = 2^k, \quad k \geq 1. \)

\[
T(n) = 7 T \left( \frac{n}{2} \right) + n^2 \\
= 7^2 T \left( \frac{n}{2^2} \right) + 7 \left( \frac{n}{2} \right)^2 + n^2 \\
= 7^2 T \left( \frac{n}{2^3} \right) + \left( \frac{n}{2^2} \right)^2 + n^2 \\
= \cdots \\
= 7^k T(1) + \mathcal{O} \left( \left( \frac{7}{4} \right)^k n^2 \right).
\]

Now
\[
7^k = 7^{\log_2 n} = n^{\log_2 7}, \quad \left( \frac{7}{4} \right)^k n^2 = \frac{7^k n^2}{4^k} = \frac{7^k}{n^2}.
\]

So
\[ T(n) = O(n \log n), \quad \text{smaller than} \quad n^3. \]

This comes up on the matrix multiplication algorithm using divide and conquer.