Recurrence relation for $\overbrace{x_1, \ldots, x_n, \ldots}^{\text{sequence}}$

"an equation expressing

x_n , for all $n > n_0$

in terms of (one or more of)

$x_1, \ldots, x_{n-1}$"

EXAMPLE: Triangular numbers

1 ,	3 ,	6 ,	10 ,	?
x_1	x_2	x_3	x_4	x_5

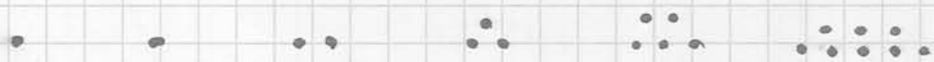
$$x_n = \sum_{i=1}^{n} i = \underbrace{\tfrac{1}{2}(n^2+n)}_{\text{"closed form"}}$$

as you know it

$$x_n = n + x_{n-1}$$
$$x_{\underbrace{1}_{\text{our } n_0}} = 1$$

as a recurrence

EXAMPLE: Fibonacci numbers



0 ,	1 ,	1 ,	2 ,	3 ,	5 ,	8 ,	?
x_1	x_2	x_3	x_4	x_5	...		

$$x_n = x_{n-1} + x_{n-2}$$
$$x_1 = 0$$
$$x_2 = 1$$

← Recurrence

$$x_n = \frac{\varphi^n - (1-\varphi)^n}{\sqrt{5}} \quad ; \ \varphi \approx 1.618$$

closed form

what is this number called?

Divide & Conquer recurrence

"a recurrence relation on the form

$$x_n = a x_{\frac{n}{b}} + g(n)$$

for integers a, b and function g"

---

normally written

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

---

we will only consider cases where

$$g \in O(n)$$

that is, relations on the form

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

How to "solve" (bring to closed form) these?

- I will show 2 special cases.
- You want more: see "Master Theorem".

EXAMPLE: "Binary search"

$B(n) = B(\frac{n}{2}) + 1$
$B(1) = 1$

"Half-decrease" recurrence — unofficial name

Given

T : running time of program P (function of n)

if

$T(n) \leq T(\frac{n}{2}) + c$

then $T \in O(\log_2 n)$ (P runs in $O(\log n)$ time)

Why: let's unroll the recurrence.

- Analyze first few levels:

level	1 :	problem size	n,	contributes	c
"	2 :	"	$n/2$,	"	c
"	3 :	"	$n/4$,	"	c

- identify Pattern

level	j :	"	$n/2^{j-1}$,	"	c

- summing over levels

n reduces to 2 in $\log_2 n$ levels.
Each level contributes c.
$\Longrightarrow$ Total running time at most $c \cdot \log_2 n \in O(\log n)$

In short, (assuming $n = 2^d$) TELESCOPING PROOF

$$
\begin{aligned}
T(n) = T\left(\frac{n}{2^0}\right) &= T\left(\frac{n}{2^1}\right) + c \\
&= T\left(\frac{n}{2^2}\right) + 2c \\
&= \dots \\
&= T\left(\frac{n}{2^{\log_2 n}}\right) + (\log_2 n)\, c \\
&= \underbrace{T(1)}_{\text{constant}} + (\log_2 n)\, c \in O(\log n)
\end{aligned}
$$

Formal proof: induction in n.

EXAMPLE: "Merge sort"

$$M(n) \leq 2M\left(\frac{n}{2}\right) + cn$$
$$M(2) \leq c$$

"Half-divide" recurrence — unofficial name

if

$$T(n) \leq 2T\left(\frac{n}{2}\right) + cn$$

then $T \in \mathcal{O}(n \log n)$

Why:

$$T(n) = T\left(\frac{n}{2^0}\right) = 2T\left(\frac{n}{2^1}\right) + cn$$
$$= 2\left(2T\left(\frac{n}{2^2}\right) + \frac{1}{2}cn\right) + cn$$
$$= 2\left(2T\left(\frac{n}{2^2}\right)\right) + 2\cdot\frac{1}{2}cn + cn$$
$$= 2\left(2T\left(\frac{n}{2^2}\right)\right) + cn + cn$$
$$= 2\left(2T\left(\frac{n}{2^2}\right)\right) + 2cn$$
$$= 2^{2}T\left(\frac{n}{2^{2}}\right) + 2cn$$
$$= \ldots = 2^{\log_2 n} T(1) + \log_2 n \cdot cn$$
$$= n\underbrace{T(1)}_{\text{constant}} + cn\log_2 n \in \mathcal{O}(n \log n)$$

Solved Exercise 1

Given:

A : array of n numbers
p : peak index (not given) we just know it exists)

Values in $A[1], \dots, A[p]$ are increasing,
Values in $A[p], \dots, A[n]$ are decreasing.

Goal: Show how p can be found reading at most $\mathcal{O}(\log n)$ entries of A.

Solution: We obtain $\mathcal{O}(\log n)$ by

i/ Doing constant work,
ii/ Dropping half of input,
iii/ Continue recursively on remainder.

"half decrease", if done right

We should use i) to determine which half to drop in ii/. So choose wisely.

We look at centre of A, that is, $A[c]$, with $c = \lfloor \frac{n}{2} \rfloor + 1$ and neighbours.
Three cases can occur.

- $A[c-1] < A[c] < A[c+1]$: We are in "ascending half" of A. $p > c$, so we drop $A[1], \dots, A[c]$, and proceed recursively on rest.
- $A[c-1] > A[c] > A[c+1]$: We are in "descending half" of A. $p < c$, so we drop $A[c], \dots, A[n]$, and proceed recursively on rest.
- $A[c-1] < A[c] > A[c+1]$: We found p (!) ; $p := c$. We are done.

A straightforward algorithm implementing this runs in $\overbrace{\mathcal{O}(\log n)}^{(\text{why?})}$.
More specifically, this idea has recurrence

$$T(n) \le T\left(\tfrac{n}{2}\right) + 3 \quad \text{\# reads from array}$$
$$T(3) \le 3$$

Solved Exercise 2

Given

- n : # days
- i, j : day index ; $1 \le i, j \le n$
- $p(i)$: price of stock on day i

Buy 1000 shares on i,
sell the shares on j ; $i < j$

Goal: Find i, j maximising profit (if profit is possible). Algorithm must run in $O(n \log n)$.

Solution: Apply D&C:

i) Split problem in two,
ii) Find optimal solution for each half,
iii) Combine these solutions into solution for the whole.

If we can find an algorithm implementing this idea, it will have recurrence $T(n) \le 2T(\frac{n}{2}) + cn$, and thus run in $O(n \log n)$.

Assume wlg. that initial input size $n = 2^k$ for some $k \in \mathbb{Z}$

Step i) and ii) are easy; the recursion does this for us. The tricky part is iii).

In iii), there are three cases:

1. best to buy and sell in first half, S,
2. —— " " —— second half, S',
3. best to buy in S and sell in S'.

(i, j) are found by taking maximum of

- optimal solution for S, $\qquad T(n/2)$
- —— " —— S', $\qquad T(n/2)$
- $\max\limits_{1 \le i \le \frac{n}{2};\ \frac{n}{2}+1 \le j \le n} (p(j) - p(i)) = \max\limits_{\frac{n}{2}+1 \le j \le n} (p(j)) - \min\limits_{1 \le i \le \frac{n}{2}} (p(i)) \qquad O(n)$

(i, j) for that value is our result.

An algorithm implementing this has recurrence

$$T(n) = 2T(\tfrac{n}{2}) + cn$$

for some c, and thus $O(n \log n)$.