- Exam 1 : Mon, Feb 20.
- No lecture on Mon, Feb 20.
- Recitation on Saturday.

Sorting.

Input: Array A of n distinct integers.

Objective: Sort A in ascending order.

IS (A [1..n])

for $j \leftarrow 2$ to n do → n times.

n times

key ← A[j]

$i \leftarrow j-1$ } const $O(1)$

→ while $i > 0$ and $A[i] > key$ do

≤ n times

$A[i+1] \leftarrow A[i]$

$i \leftarrow i-1$ } const $O(1)$

$A[i+1] \leftarrow key$

$O(n^2)$ in the worst case

Example .

A:

18	3	24	16
1	2	3	4

j = ~~2~~ ~~3~~ 4

key : ~~3~~ ~~24~~ 16

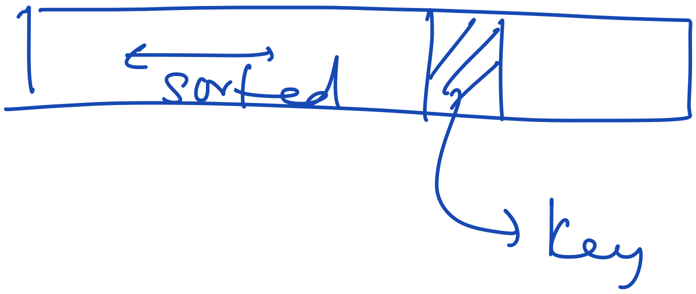
i = ~~1~~ ~~0~~ ~~2~~
~~3~~ ~~2~~ 1

A:

3 ~~18~~	18	24	16
1	2	3	4

3	18	24	24

3	~~18~~ 16	18	24



Running time:

Best case : $\Theta(n)$.

tight bound.

Worst case : $\Theta(n^2)$.

Asymptotic notation.

asymptotic lower bound

O, Ω, Θ, o, ω.

tight bound.

↳ asymptotic upper bound.

$n \to \infty$

worst case input

Rest of the course: worst-case running time of algorithm.

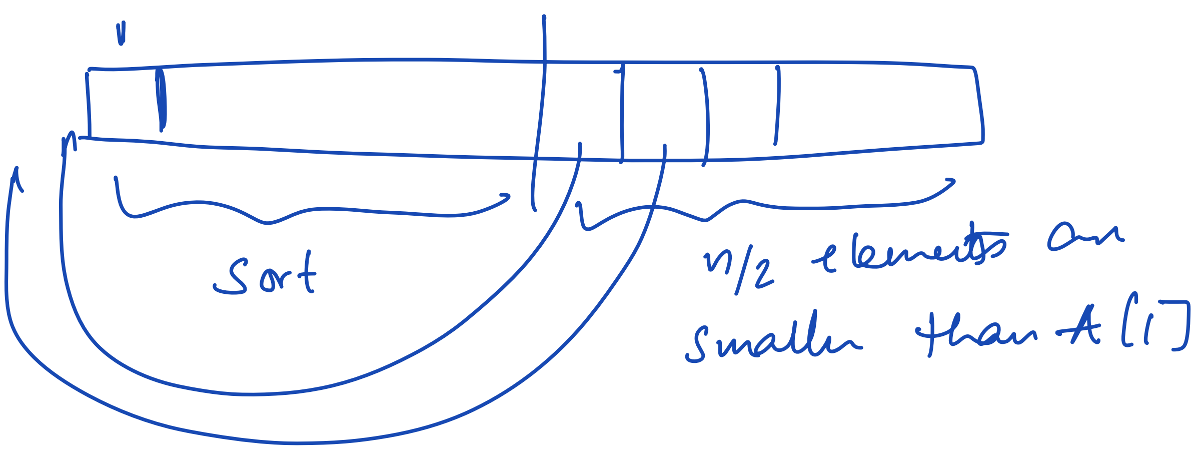
$T(n) = O(n^2)$,

- in the worst case IS will not take more than n^2 time within constants.

$$7\boxed{n^2} + 18n - 500$$

Why is $T(n) = \Omega(n^2)$?

Why is it true that IS takes $\approx n^2$ time in the worst case?



Each item will need to make $\geq n/2$ swaps. There are $n/2$ such elements.

$$\therefore T(n) \geq \frac{n}{2} \cdot \frac{n}{2} = \frac{n^2}{4}$$

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

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

Suppose there is an alg. A for which $T(n) = O(n^3)$, ✓

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

$T(n) = \Theta(\underline{n^2})$ ✓✓

$\Theta(\underline{n^3})$ ✓✓

$\Theta(n^{2.5})$ ✓✓

o — ✓ — O

o — $\neq$ — Θ

$n^{1.5} = o(n^2)$

ω — ✓ — Ω

ω — $\neq$ — Θ

$\underline{\underline{n}}^{1.5} = \omega(n)$

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \begin{cases} 0 & \Rightarrow f(n) = o(g(n)) \\ \infty & \Rightarrow f(n) = \omega(g(n)) \\ c & \Rightarrow f(n) = \Theta(g(n)) \end{cases}$$

RAM model of computation.

- all high-level operations take const time.

  - comparisons, arithmetic, return, fn. call, array indexing.

- all int & nos can be stored in a word of memory.

- all memory access takes const time.

Divide & Conquer.

Input: int $n \geq 0$

Obj: To compute 2^n.

$n = 100$

(A) pow2(int n) $T(n)$

if $n = 0$ then
return 1 $\}$ $O(1)$

$\underline{\underline{1}}$

else

return $2 * \underbrace{\text{pow2}(n-1)}_{\boxed{\text{pow2}(n-1)}}$ $T(n-1)$

2^{n-1}

(B) pow (int n)

if n = 0 then

return 1

else

return pow(n-1) + pow(n-1)

>> 100 million years.

≈ 1

why is that bad!?

(C) pow (int n) $T(n)$

if n = 0 then

return 1 $\Big\}\ O(1)$ ≈ 1.2

else

x ← pow(n-1) $T(n-1)$

return x + x c

Dynamic Prog.

(D) pow (int n) → $T(n)$

if n = 0 then

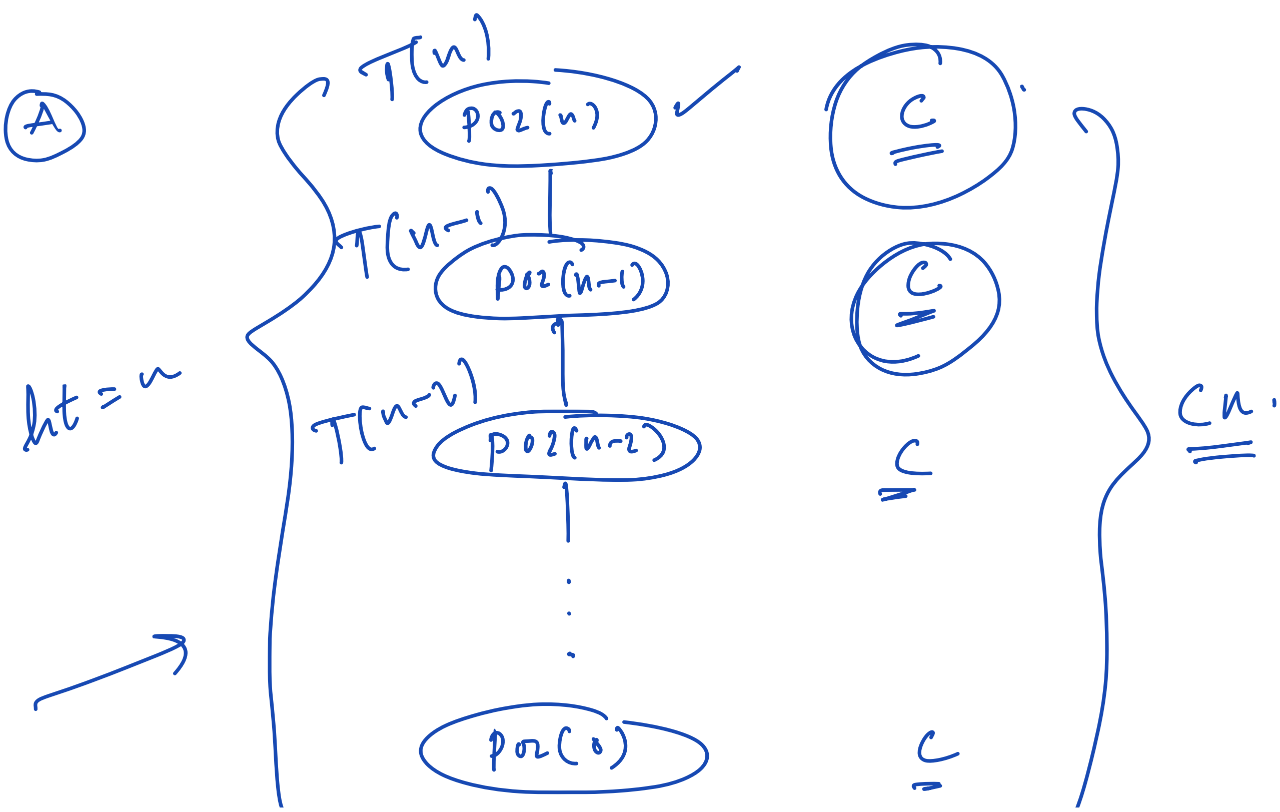
return 1

else

```
tmp ← po2(⌊n/2⌋)          T(n/2)
if n is even then
    return tmp * tmp       } O(1)
else
    return tmp * tmp * 2.
```

Runtime analysis.

$T(n)$: worst case running time of po2 on an input of size n.



$$T(n) = T(n-1) + c$$

$$= (T(n-2)+c) + c$$

$$= T(n-3) + 3c$$

$$\vdots$$

$$= T(n-k) + kc$$

Recursion bottoms out when

$n-k = 0$, i.e, when $k=n$.

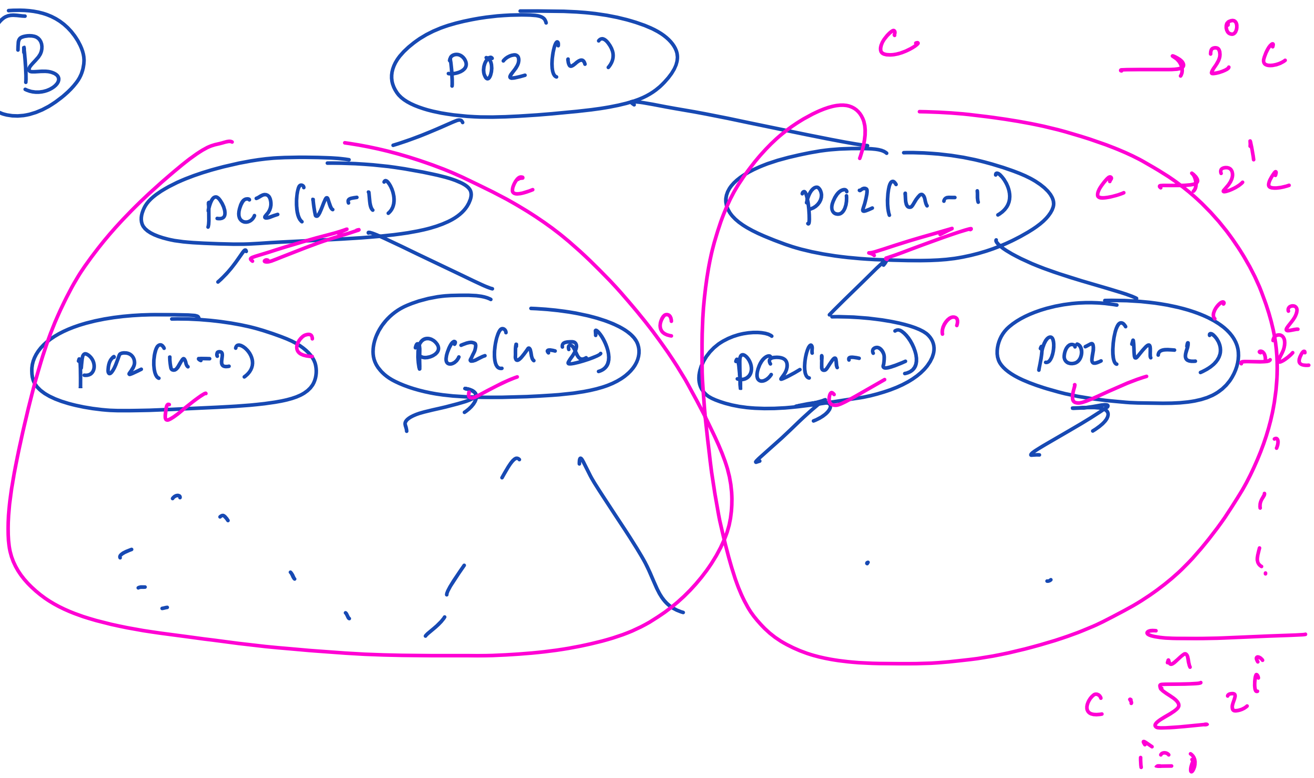
When that happens, we have

$$T(n) = T(0) + nc$$

$= O(1) + nc$

$= \boxed{\Theta(n)}$.

B



Complete bin. tree with ht. n.

This has $\Omega(2^n)$ nodes.

$$T(n) = \begin{cases} O(1), & n = 0 \quad \text{(Base case)} \\ 2T(n-1) + c, & \text{otherwise.} \end{cases}$$

$$T(n) = 2T(n-1) + c$$

$$= 2\left(2T(n-2) + c\right) + c$$

$$= 2^2 T(n-2) + 2^1 c + 2^0 c$$

$$= 2^2\left(2T(n-3) + c\right) + 2^1 c + 2^0 c$$

$$= 2^3 T(n-3) + 2^2 c + 2^1 c + 2^0 c$$

$$\vdots$$

$$= 2^k T(n-k) + c \cdot \sum_{i=0}^{k-1} 2^i$$

Recursion bottoms out when $n-k=0$, i.e., $k=n$.

When this happens, we have

$$T(n) = 2^n T(0) + c \cdot \sum_{i=0}^{n-1} 2^i$$

$$= 2^n O(1) + c \cdot (2^n - 1)$$

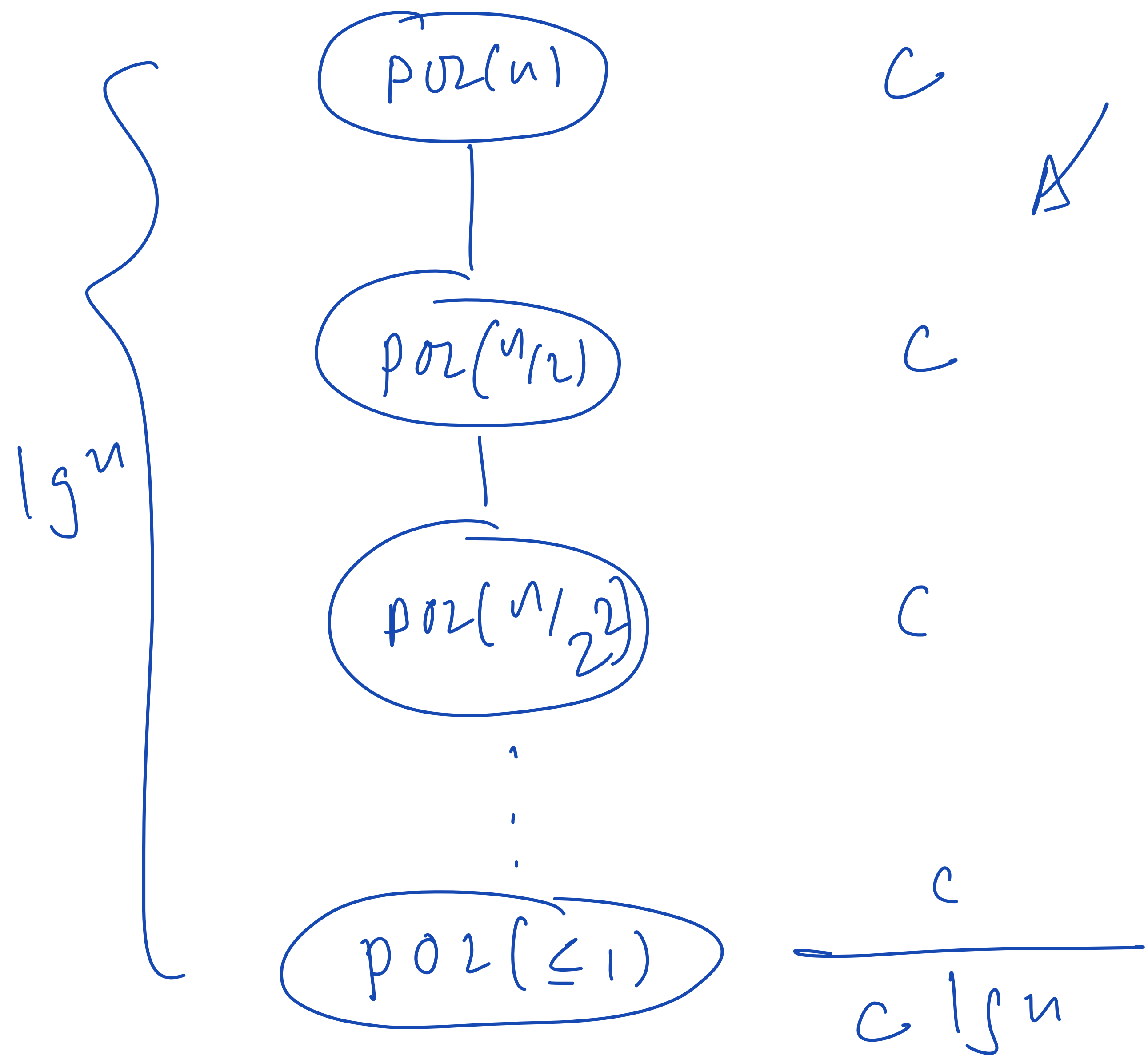
$$= \Theta(2^n).$$

(c). $T(n) = \begin{cases} O(1), & n=0 \\ T(n-1)+c, & \text{o.w.} \end{cases}$

Same as (A).

④ Assuming n is an exact power of 2.

$$T(n) = \begin{cases} O(1), & \text{if } n=0 \\ T\left(\frac{n}{2}\right) + c, & \text{o.w.} \end{cases}$$



$$T(n) = T\left(\frac{n}{2}\right) + \underline{\underline{c}}$$

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

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

$$\vdots$$

$$= T\left(\frac{n}{2^k}\right) + kc$$

Recursion bottoms out when

$$\frac{n}{2^k} < 1 \implies k > \lg n.$$

$\therefore\ T(n) \leq T(0) + (\lg n + 1)\, c$.

$$= \underline{\underline{O(\lg n)}}.$$

<u>Input</u>: - Array A of n distinct int.

- int k

<u>Obj</u> : To output Yes, if $k \in A$.

No, o.w.

```
LS (A[1..n], k)
    if n == 1 then
        return A[1] == k.
    else
```

$$\text{return } \underbrace{(A[n] == k)}_{c} \text{ or } \underbrace{(LS(A[1..n-1], k))}_{T(n-1)}$$

Runtime recurrence.

$T(n)$: worst case running time of LS on an i/p of size n.

$$T(n) = \begin{cases} O(1), & n = 1 \\ T(n-1) + c. & \end{cases}$$

$$T(n) = \Theta(n).$$