

Merge Sort

An optimal sorting algorithm based on Divide & Conquer approach. ($\Theta(n \log n)$ time)

Divide & Conquer

Many useful algorithms are recursive in structure

Divide the problem into a number of subproblems that are **smaller** instances of the same problem

Conquer the subinstances by solving them recursively.
if small enough, solve them directly

Combine the solutions of the subinstances into the solution of the original instance

Usually Conquer step is trivial and cheap.
and only one of the two, Divide or Combine, is expensive

Merge Sort

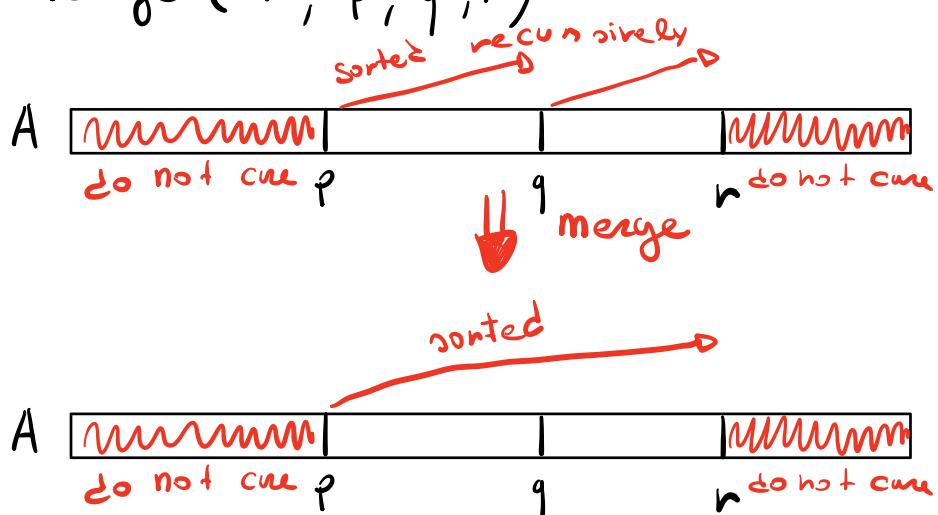
Divide an n -element array into two sub arrays of size $\approx \frac{n}{2}$



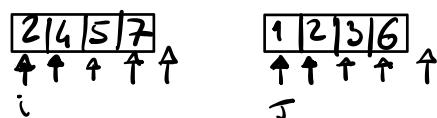
Conquer sorts two subarrays recursively.
if size ≤ 1 , do nothing

Combine merges the two sorted subarrays
to produce their sorted union

Merge (A, p, q, r)



Example



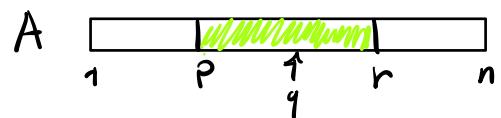
$1 2 2 3 4 5 6 7$

Linear time algorithm

MergeSort (A, p, r)

if $p < r$

$$q = \left\lfloor \frac{p+r}{2} \right\rfloor \quad // \text{divide}$$



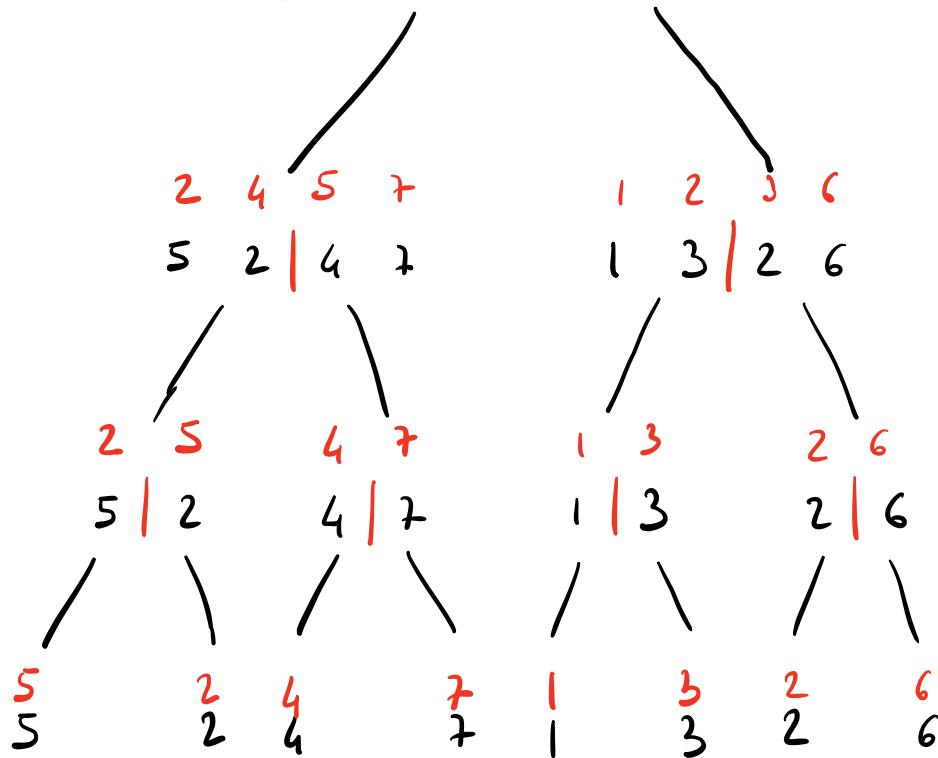
MergeSort (A, p, q)

MergeSort (A, q+1, r)

Merge (A, p, q, r) // Conquer

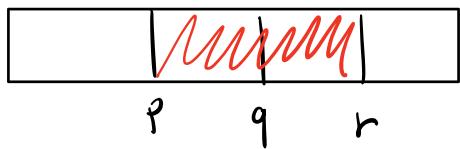
Merge (A, p, q, r) // Combine

1 2 2 3 4 5 6 7
5 2 4 7 | 1 3 2 6
 $p=1$ $q=4$ $n=8$



Open VisualAlg.org

Merge (A, p, q, r)



$$n_1 = q - p + 1$$

MS is not in place

$$n_2 = r - q$$

MS is stable

let $L[1 \dots n_1+1]$ and $R[1 \dots n_2+1]$
be new arrays

for $i = 1$ to n_1

$$L[i] = A[p+i-1]$$

for $i = 1$ to n_2

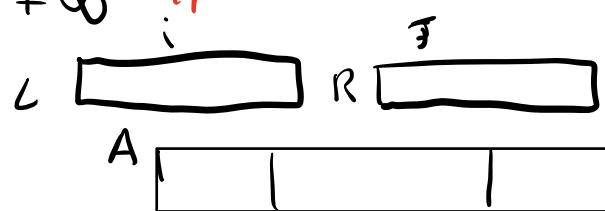
$$R[i] = A[q+i]$$

$$L[n_1+1] = +\infty$$

// sentinels

$$R[n_2+1] = +\infty$$

$$i = j = 1$$



for $k = p$ to r
if $L[i] \leq R[j]$

$$A[k] = L[i]$$

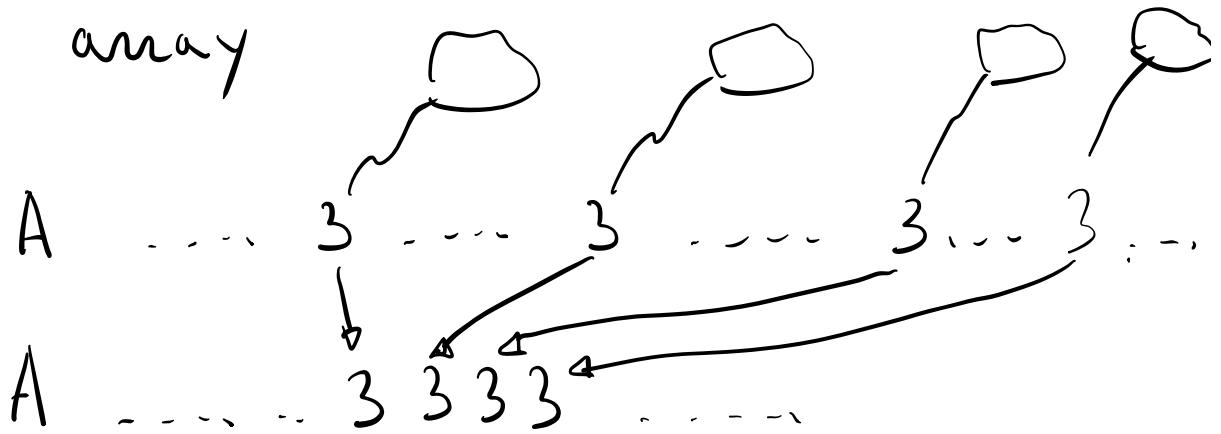
$\Theta(n)$
time

$$i = i + 1$$

$$A[k] = R[j]$$

$$j = j + 1$$

Stable occurrences of the same values will preserve their relative order in the sorted array

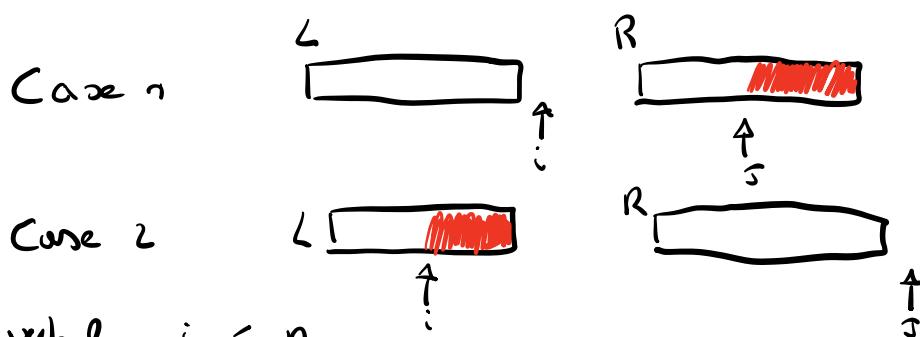


Living without sentinels

Need 3 while loops

while $i \leq n_1$ and $\tau \leq n_2$:

// do comparisons



while $i \leq n_1$

// copy from L

while $\tau \leq n_2$

// copy from R

MergeSort: Analysis

Are best and worst cases different? NO!

A recurrence $T(n)$ for the running time of a divide & conquer-based algorithm

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq c \\ a T\left(\frac{n}{b}\right) + D(n) + C(n) \end{cases}$$

a = # of subinstances

$\frac{n}{b}$ = size of each subinstance

$D(n)$ = cost of dividing a instance of size n

$C(n)$ = cost of combining instances of size n

$$\begin{aligned} T(n) &= \begin{cases} \Theta(1) & \text{if } n \leq 1 \\ 2 T\left(\frac{n}{2}\right) + \cancel{\Theta(1)} + \Theta(n) \end{cases} \\ &= 2 T\left(\frac{n}{2}\right) + \Theta(n) \end{aligned}$$

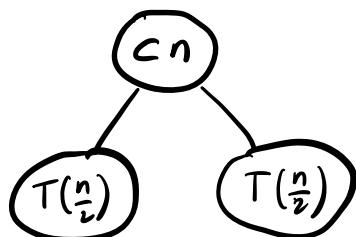
Solve recurrences with recursion tree

$$T(n) = \begin{cases} < & \text{if } n \leq 1 \\ \underline{c} T\left(\frac{n}{2}\right) + \underline{c} n \end{cases}$$

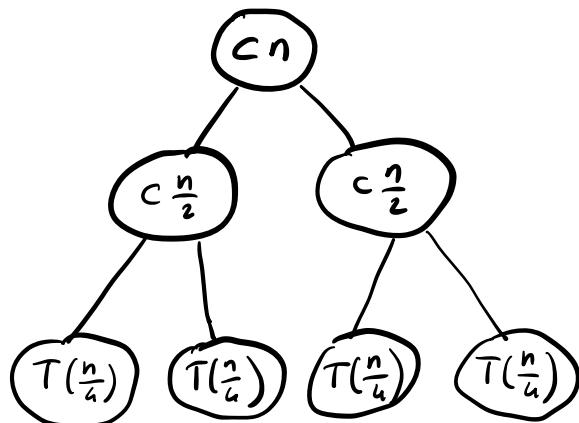
①

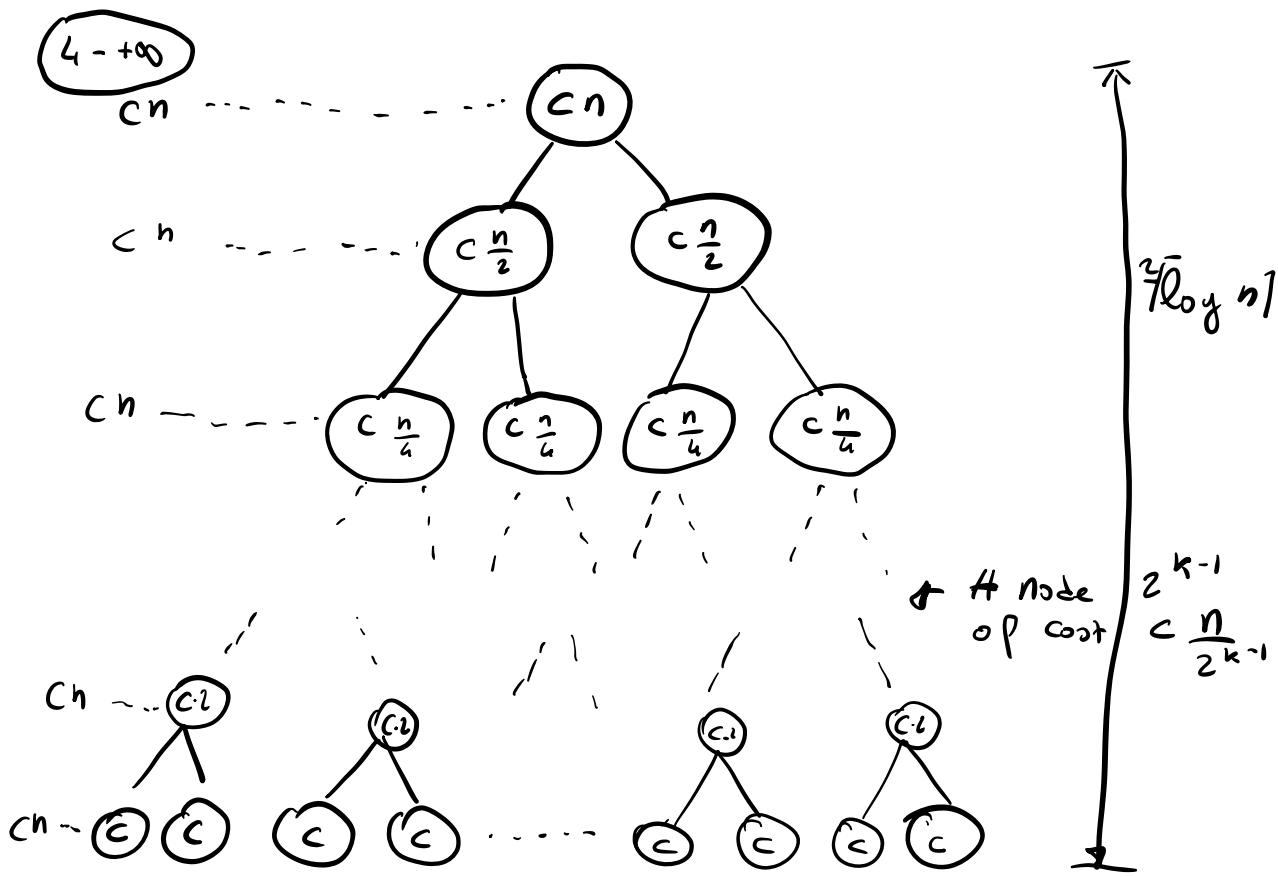


②



③





$$\begin{aligned}
 T(n) &= \sum \text{cost of every node} \\
 &= \text{cost of each level} \cdot \# \text{ of levels} \\
 &= Cn \cdot \log n = \Theta(n \log n) \text{ time}
 \end{aligned}$$

Exercise

Merge Sort recurrence is

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 1 \\ 2T\left(\frac{n}{2}\right) + \Theta(n) & \text{otherwise} \end{cases}$$

and it solves to $\Theta(n \log n)$ time.

Find a recurrence that solves to $\Theta(n)$ time.

Solutions

$$T(n) = \underbrace{a}_{\substack{\# \text{ of} \\ \text{children}}} T\left(\frac{n}{5}\right) + \underbrace{D(n) + C(n)}_{\substack{\text{label of} \\ \text{cost in the node}}}$$

each child

①

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 1 \\ T(1) + \Theta(n) & \text{otherwise} \end{cases}$$

my_max(A, p, r)

if $p == r$ return $A[p]$

$M = \text{my_max}(A, p, r)$

for $i = p+1$ to r

if $M < A[i]$ $M = A[i]$ $C_n = \Theta(n)$

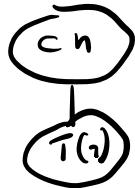
return M

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 1 \\ 2T(1) + \Theta(n) \\ a = n \end{cases}$$

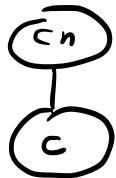
①

$T(n)$

②

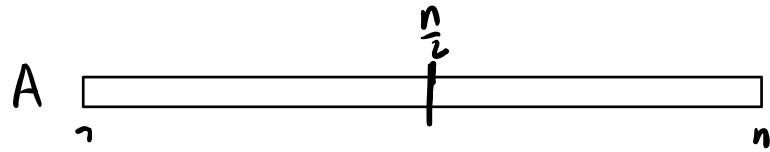


③



$$T(n) = c_n + c = \Theta(n)$$

$$\textcircled{2} \quad T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 1 \\ 2T\left(\frac{n}{2}\right) + \Theta(1) & \text{otherwise} \end{cases}$$



`my-max(A, p, r)` {

if $p < r$

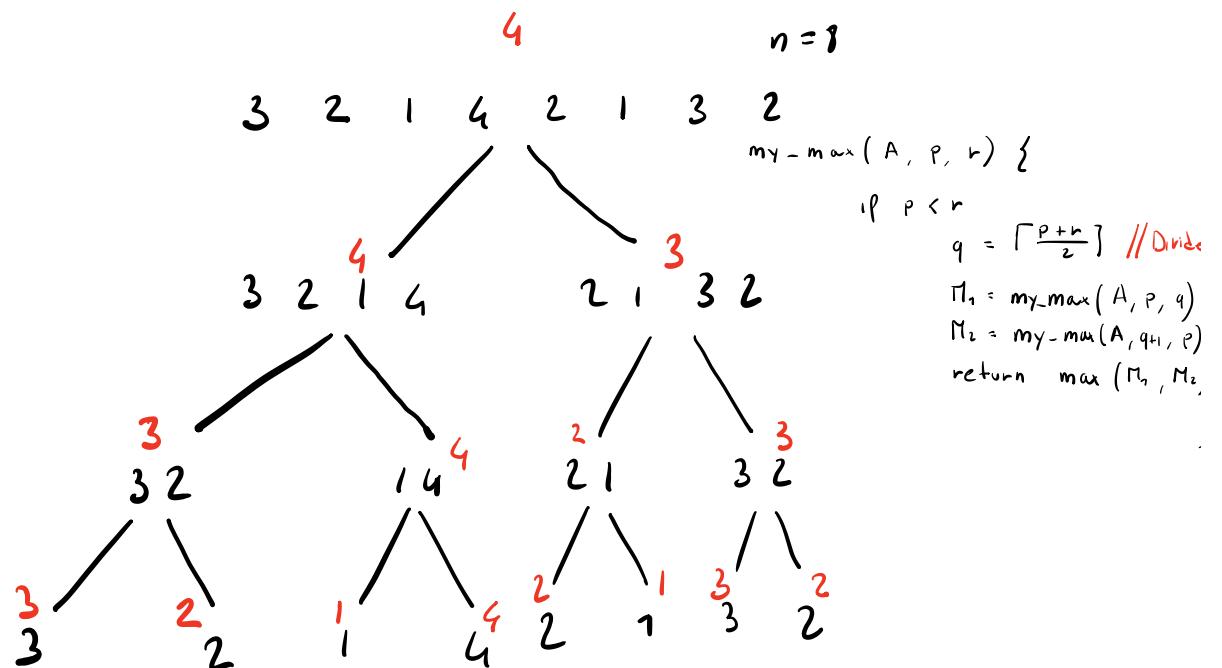
$$q = \lceil \frac{p+r}{2} \rceil \quad // \text{Divide}$$

$M_1 = \text{my-max}(A, p, q) \quad // \text{Conquer}$

$M_2 = \text{my-max}(A, q+1, r) \quad // \text{Conquer}$

return $\max(M_1, M_2) \quad // \text{Combine}$

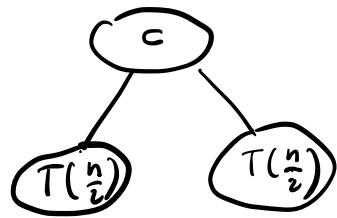
}



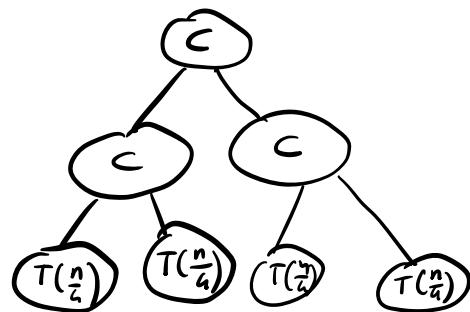
①

$T(n)$

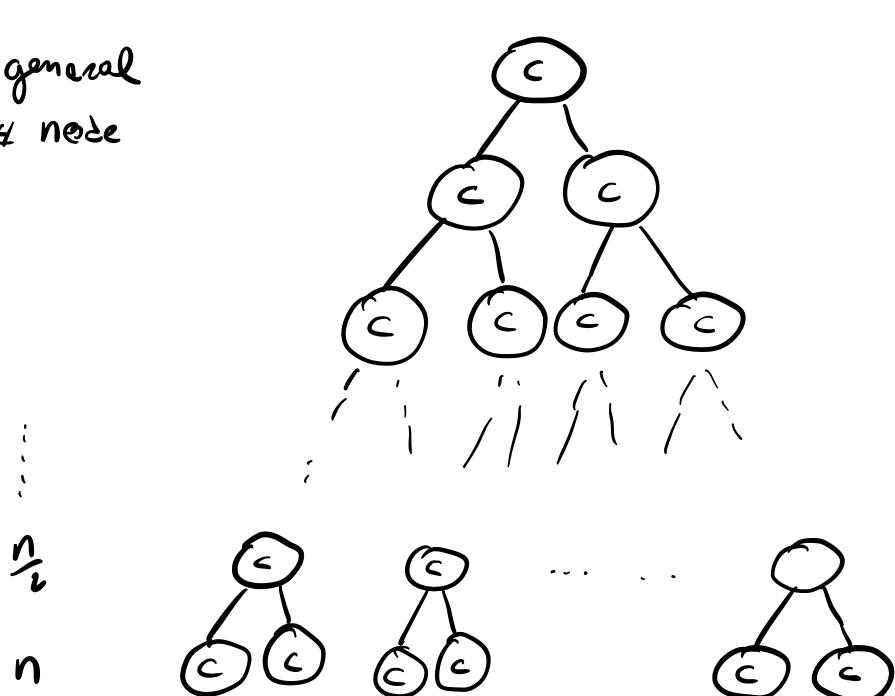
②



③



In general
node



$$n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \dots + 4 + 2 + 1 \approx 2n \quad \Theta(n)$$

③

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 1 \\ -T(n-1) + \Theta(1) & \text{otherwise} \end{cases}$$

my-max (A, p, r)
 if $p = r$ return $A[p]$
 $M = \text{my-max} (A, p, r-1)$
 return $\max(M, A[r])$

①

$T(n)$

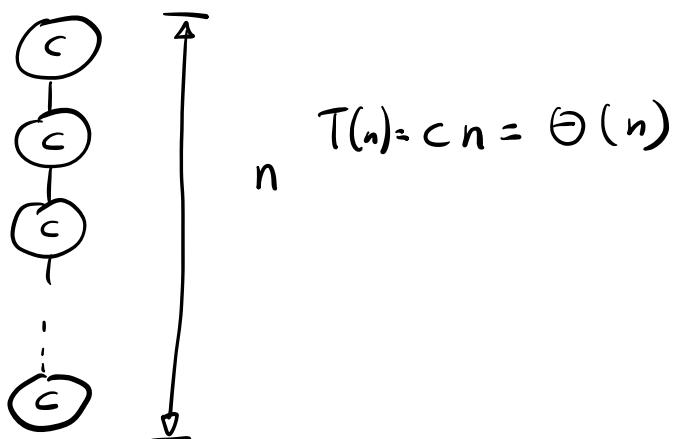
②

c
 $T(n-1)$

③

c
 c
 $T(n-2)$

In general,



my-max2(A, p, r)

if $p == r$ return $A[p]$

$M = \text{my-max} (A, p+1, r-1)$

return $\max (M, A[p], A[r])$

$$T(n) = \begin{cases} \Theta(1) \\ T(n-2) + \Theta(1) \end{cases}$$

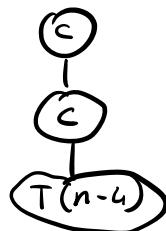
①



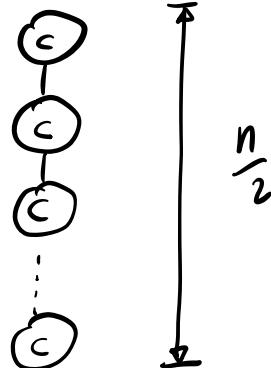
②



③



In general



$$\frac{n}{2}$$

$$T(n) = c \frac{n}{2} = \Theta(n)$$

Exercise

Searching for a key k in a collection (static)

Given array A of n integers

query (k) $\begin{cases} \text{True} & \text{if } \exists i \text{ such that } A[i] = k \\ \text{False} & \text{otherwise} \end{cases}$

- ① Provide pseudocode of Linear Search
and analyze it

Linear Search (A, k)

```
for i : 1 to n
    if  $A[i] == k$  :
        return True
    return False
```

Best case $A[1] == k$ $\Theta(1)$ time

Worst case k not in A $\Theta(n)$ time

② Divide & Conquer approach

(a) A is not sorted

(b) A is sorted

(2a)

Stupid Search (A, p, r, k)

if $p = r$ return $A[p] = z$

if $p < r$

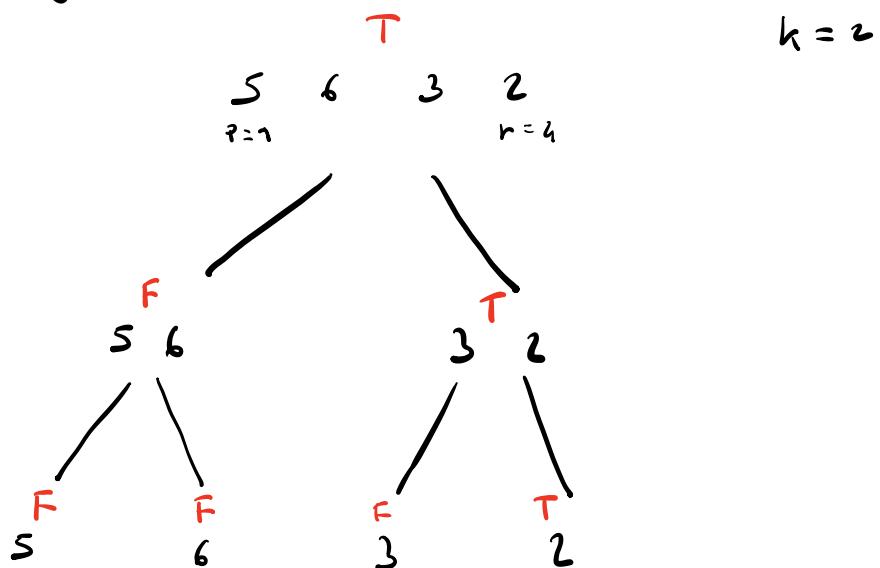
$q = \lceil \frac{p+r}{2} \rceil$ // Divide

$R_1 = \text{Stupid Search}(A, p, q, k)$ // C

$R_2 = \text{Stupid Search}(A, q+1, r, k)$

return R_1 or R_2 // Combine

Running Example



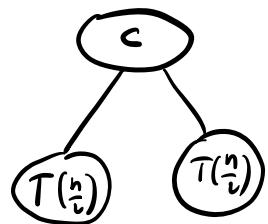
Recurrence

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 1 \\ a T\left(\frac{n}{b}\right) + D(n) + C(n) \\ 2 T\left(\frac{n}{2}\right) + \Theta(1) \end{cases}$$

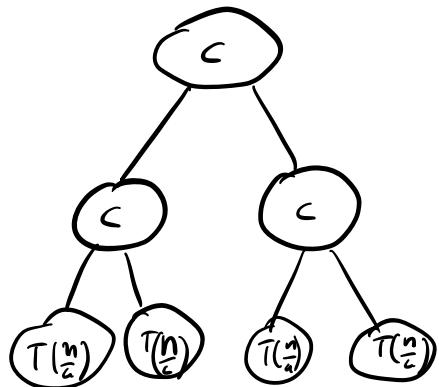
①



②



③



$$T(n) = \Theta(n) \text{ time}$$

(25) A is sorted

Binary Search (A, p, r, k)

if $p > r$ return False

if $p = r$
return $A[p] == k$

$q = \lceil \frac{p+r}{2} \rceil$ // Divide

if $A[q] == k$ return True

if $A[q] < k$
return Binary Search ($A, q+1, r, k$)

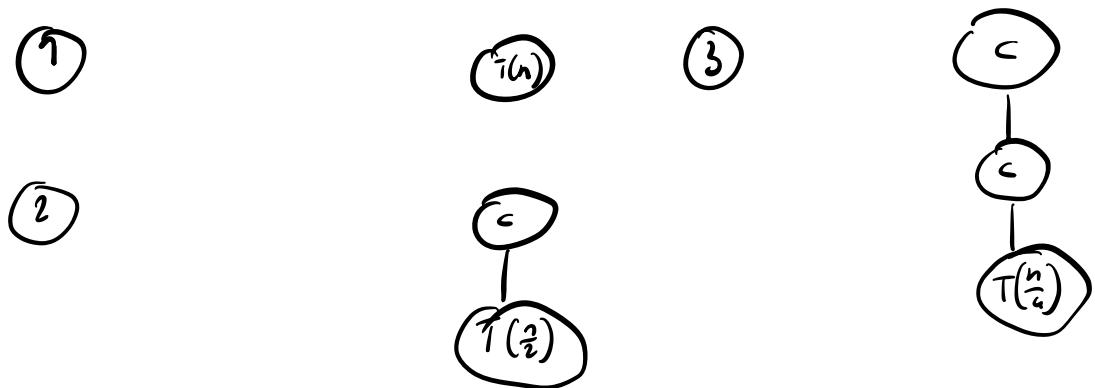
else

return Binary Search ($A, p, q-1, k$)

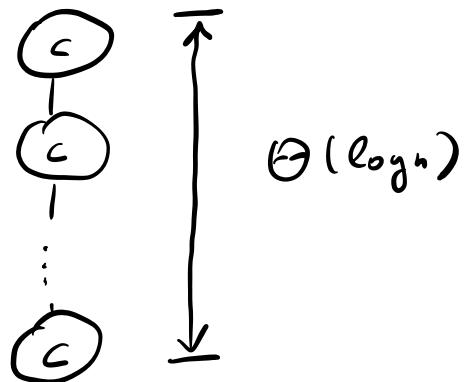
Do a running example by yourself

Recurrence

$$T(n) = \begin{cases} \Theta(1) & \text{if } n=2 \\ a T\left(\frac{n}{2}\right) + D(n) + C(n) \\ \underline{+ T\left(\frac{n}{2}\right) + \Theta(1)} \end{cases}$$



In general



$$T(n) = \Theta(\log n) \text{ time}$$

(Ex)

Modify the pseudocode above
to return \rightarrow if $k \notin A$
or p if $A[p] == k$

$|D|$ = dataset size

$|C|$ = customer size

①

customers = list(...)

for x in data:

if x in customers:

do - something

$$\approx |D| \times |C|$$

②

customers = sorted(list(...))

for x in data:

if binary search(x , customers):

do - something

$$\approx |C| \log |C| + |D| \log |C|$$

③

customers = set(list(...))

for x in data:

if x in customers:

do - something

$$\approx |C| + |D|$$