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

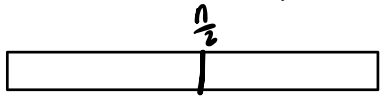
Conquer the subinstances by solving them recursively. if small enough, solve them directly

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

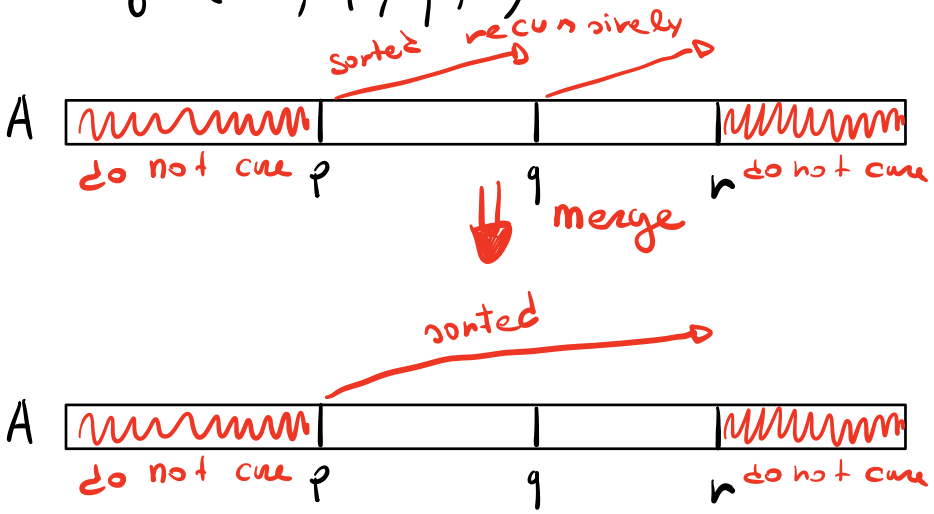
Divide an n-element array into <u>two</u> subarrays of size $\approx \frac{n}{2}$



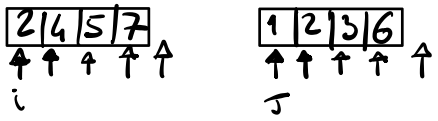
Conquer sorts two subarrays recursively.
If size ≤ 1, do nothing

Combine merges the two sorted subarrays to produce their sorted union

Merge (A, p, q, r)

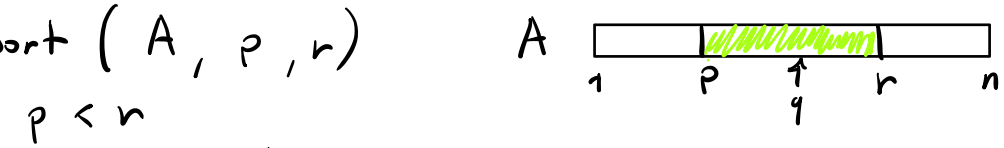


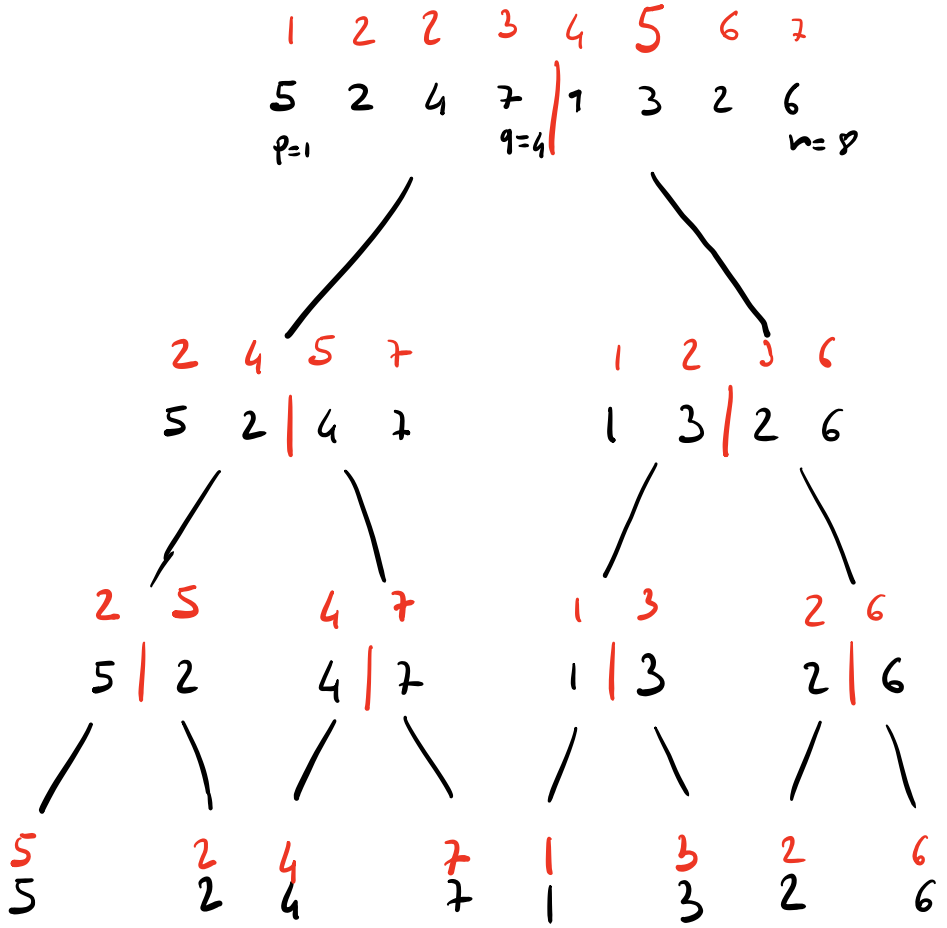
Example



1 2 2 3 4 5 6 7 Linear time algorithm

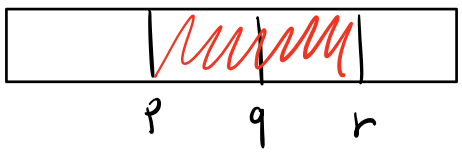
```
MergeSort ( A , p , r)
  if  p < r
      q = ⌊ (p + r) / 2 ⌋        // Divide
      MergeSort ( A , p , q)
      MergeSort ( A , q+1 , r)   // Conquer
      Merge ( A , p , q , r)     // Combine
```





Open VisuAlgo

Merge (A, p, q, r)



$n_1 = q - p + 1$

$n_2 = r - q$

MS is not inplace

MS is stable

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

c

for $i = 1$ to n_1

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

c

for $i = 1$ to n_2

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

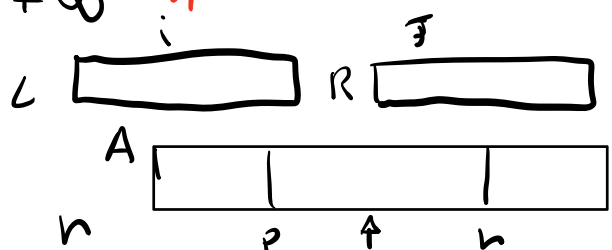
$c(n_1 + n_2) = cn$

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

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

sentinels

$i = j = 1$



cn

for $k = p$ to r

if $L[i] \le R[j]$

$A[k] = L[i]$

$i = i + 1$

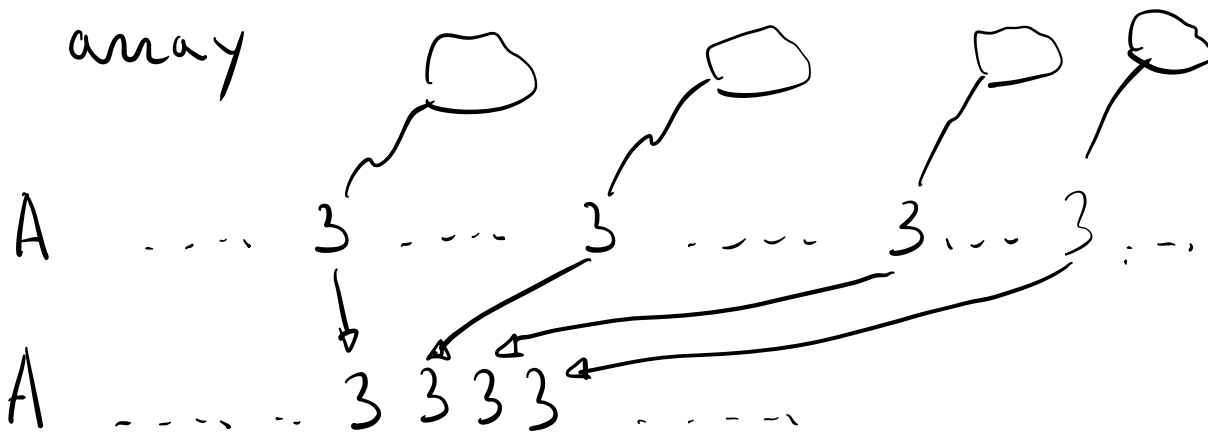
else

$A[k] = R[j]$

$j = j + 1$

$\Theta(n)$ time

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

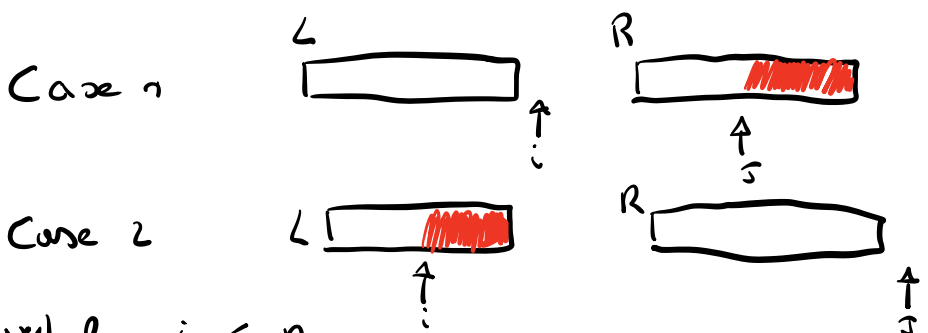


Living without sentinels

Need 3 while loops

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

// do comparisons



while $i \leq n_1$

// copy from L

while $j \leq n_2$

// 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

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

a = # of sub instances

$\frac{n}{b}$ = size of each sub instance

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

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

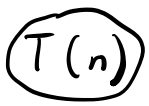
$$T(n) = \begin{cases} \Theta(1) & \text{if } n \le 1 \\ 2\,T\left(\frac{n}{2}\right) + \cancel{\Theta(1)} + \Theta(n) \end{cases}$$

$$= 2\,T\left(\frac{n}{2}\right) + \Theta(n)$$

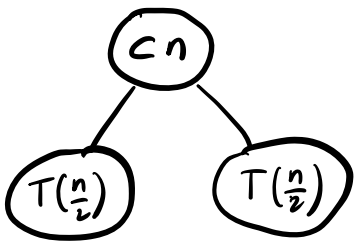
Solve recurrences with recursion tree

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

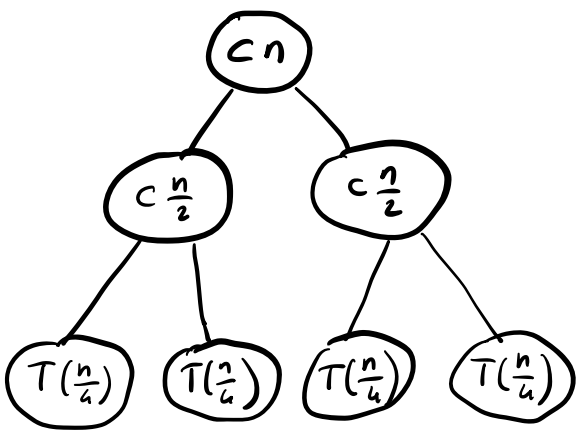
1

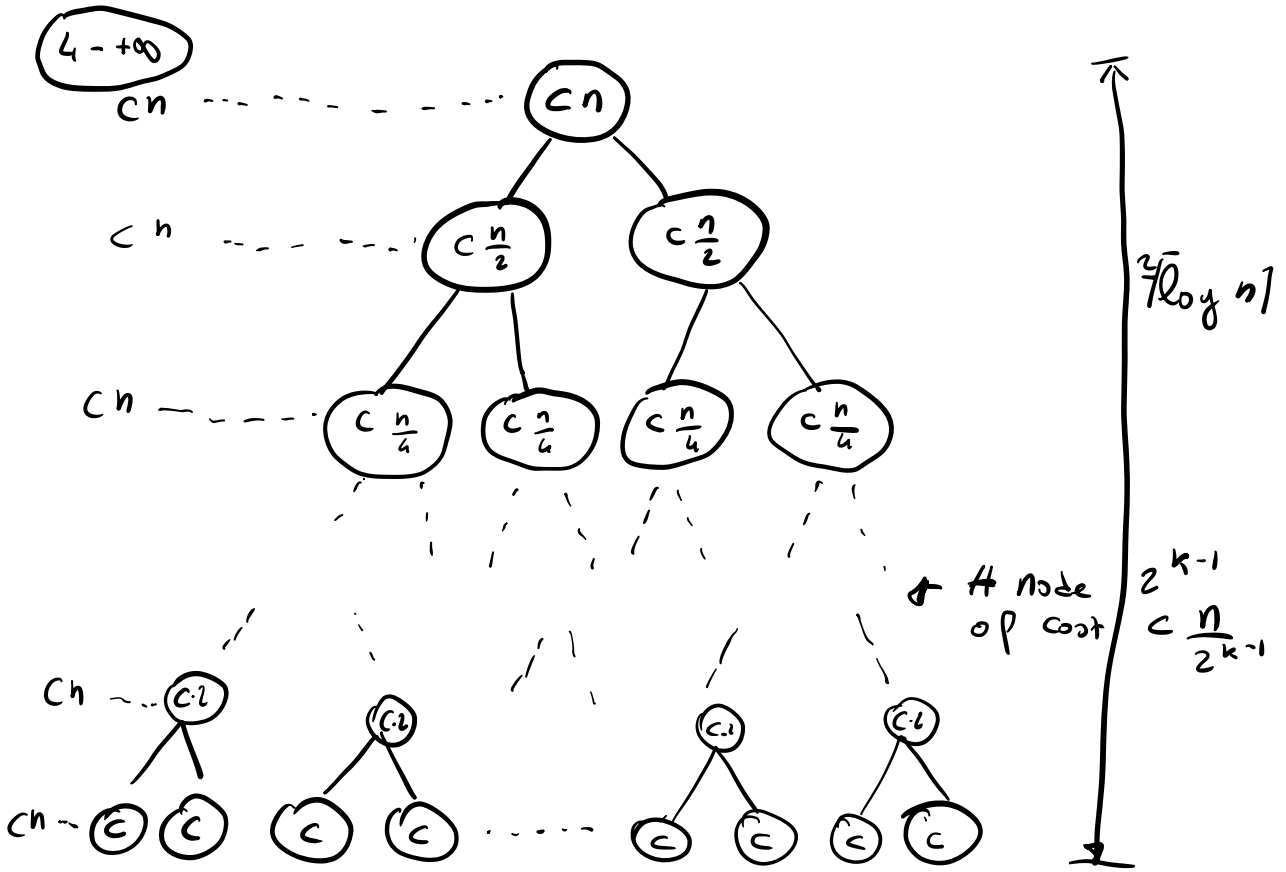


2



3





$$T(n) = \sum \text{cost of every node}$$

$$= \text{cost of each level} \cdot \#\text{ of levels}$$

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

Exercise

Merge Sort recurrence is

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

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

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

Solutions

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

(1)

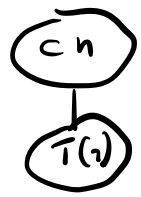
$$T(n) = \begin{cases} \Theta(1) & \text{if } n \le 1 \\ T(1) + \Theta(n) & \end{cases}$$

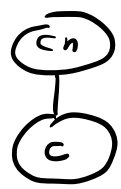
```
my_max ( A, p, r)

    if  p == r   return  A[p]
    M = my_max ( A, p, p)
    for  i = p+1  to  r
          if  M < A[i]
                M = A[i]
    return  M
```

c | $cn = \Theta(n)$

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

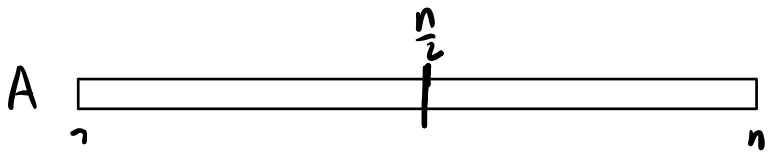
① 

② 

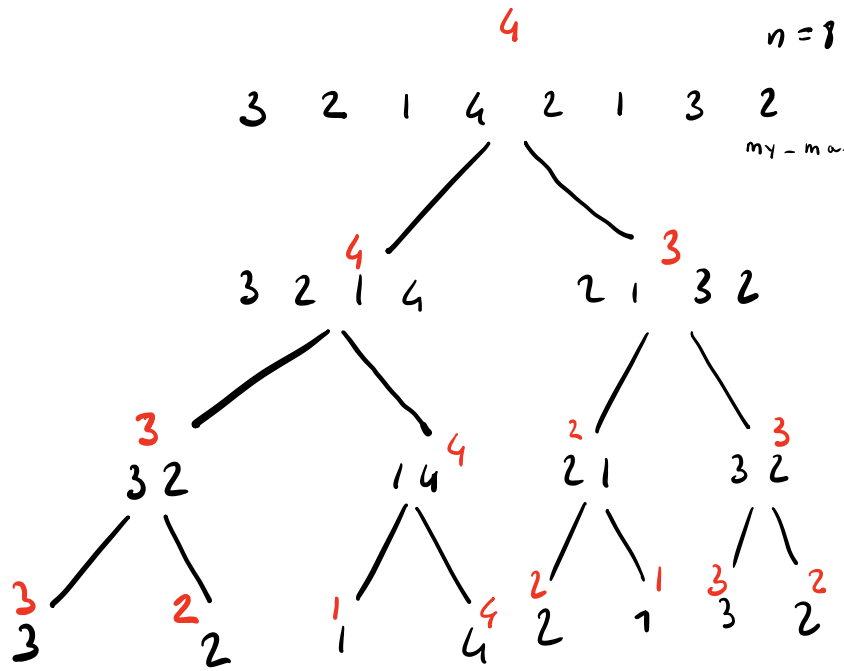
③ 

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

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



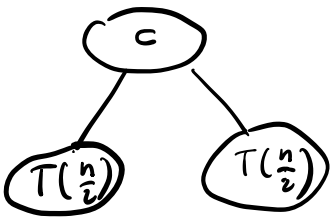

```
my_max(A, p, r) {
    if p < r
        q = ⌈(p+r)/2⌉   // Divide
        M1 = my_max(A, p, q)
        M2 = my_max(A, q+1, p)   // Conquer
        return max(M1, M2)   // Combine
}
```



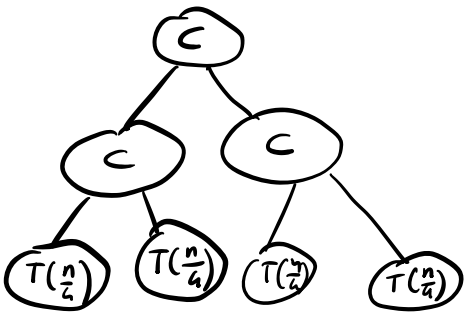

```
my_max(A, p, r) {
    if p < r
        q = ⌈(p+r)/2⌉   // Divide
        M1 = my_max(A, p, q)
        M2 = my_max(A, q+1, p)
        return max(M1, M2
```

① $T(n)$

②



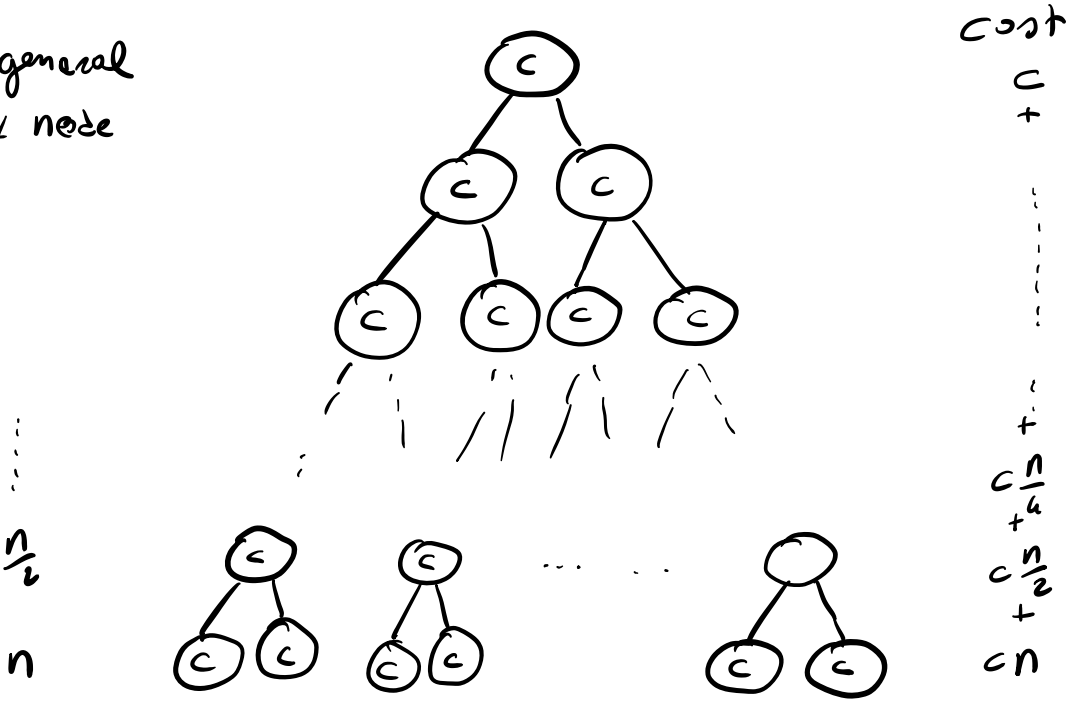
③



In general

node

cost



$n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \cdots + 4 + 2 + 1$ $\Theta(n)$

$\simeq 2n$

③

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

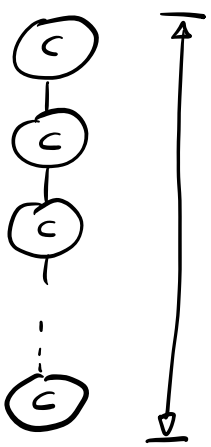
```
my-max ( A, p, r)
    if  p == r    return  A[p]
    M =  my-max ( A, p, r-1)
    return   max ( M, A[r])
```

① $T(n)$

②

③

In general,

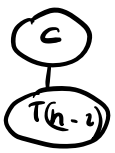


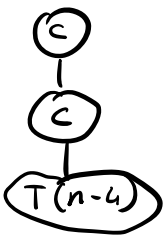
$T(n) = cn = \Theta(n)$

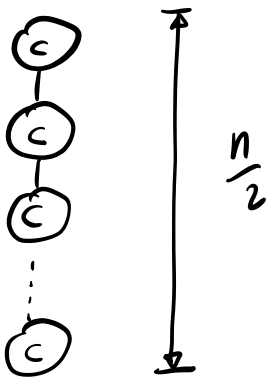
```
my-max2( A, p, r)
    if p == r    return A[p]
    M = my-max (A, p+1, r-1)
    return max (M, A[p], A[r])
```

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

① 

② 

③ 

In general 

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

Exercise

Searching for a key k in a collection (static)

Given array A of n integers

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

① Provide pseudocode of Linear Search and analyze it

```
LinearSearch(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

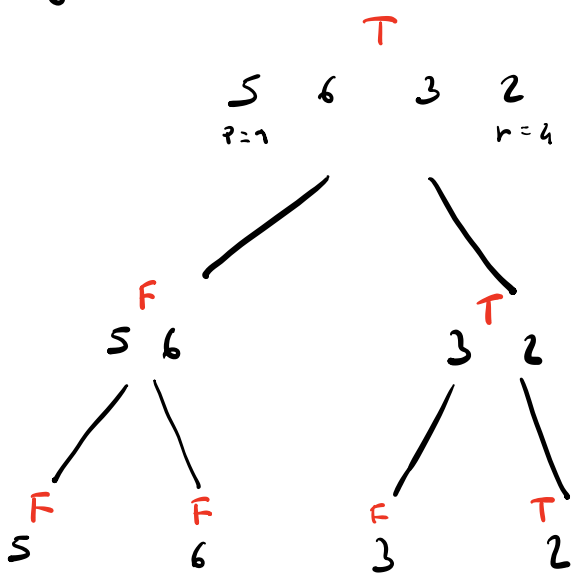
(a) A is not sorted

(b) A is sorted

(2a)

```
Stupid Search ( A, p, r, k)
    if p == r   return  A[p] == k
    if p < r
        q = ⌈ (p+r)/2 ⌉        // Divide
        R1 = Stupid Search ( A, p, q, k)
        R2 = Stupid Search ( A, q+1, r, k)   // C
        return  R1 or R2       // combine
```

Running Example

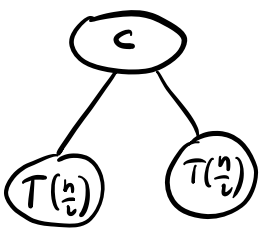


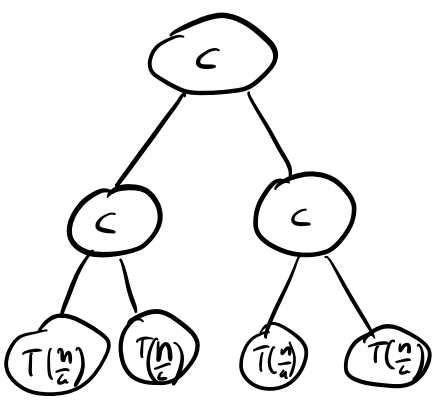
k = 2

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. $T(n)$

2. 

3. 

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

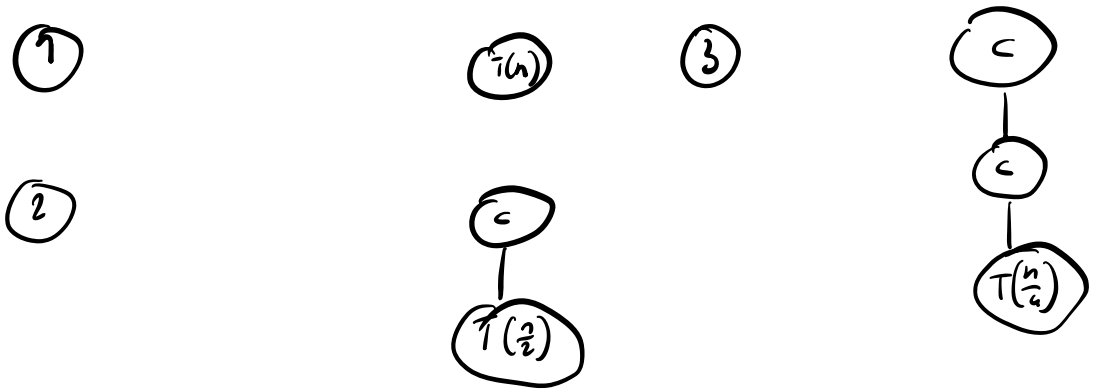
(25) A is sorted

```
BinarySearch (A, p, r, k)
    if p > r   return False
    if p == r
            return A[p] == k
    q = ⌈(p+r)/2⌉        // Divide
    if A[q] == k   return True
    if A[q] < k
        return BinarySearch (A, q+1, r, k)
    else
        return BinarySearch (A, p, q-1, k)
```

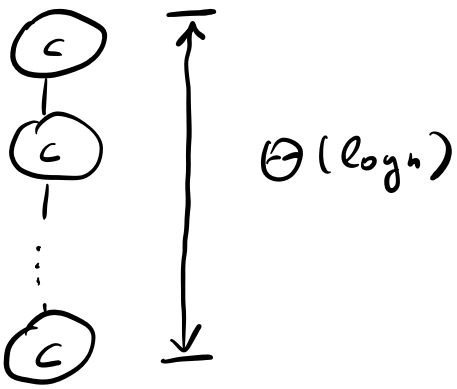
Do a running example by yourself

Recurrence

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



In general



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

Ex Modify the pseudo code above to return -1 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
```

$\simeq |D| \times |C|$

②

```
customers = sorted(list(...))
for x in data:
    if binary_search(x, customers):
        do_something
```

$\simeq |C| \log |C| + |D| \log |C|$

③

```
customers = set(list(...))
for x in data:
    if x in customers:
        do_something
```

$\simeq |C| + |D|$