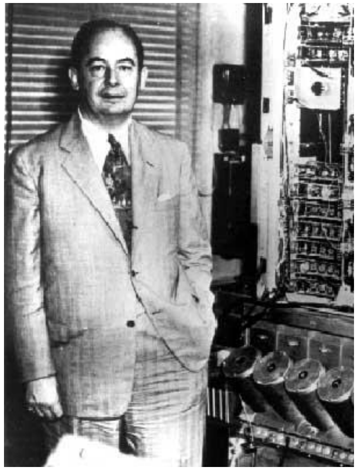
- Merge Sort
- Master Theorem
- Divide-and conquer paradigm
- Other algorithms by D&C

Merge Sort



Jon von Neumann (1945)

Merge-Sort $A[1..n]$

1. If $n = 1$, done.
2. Recursively sort $A[1..n/2]$

   and $A[n/2+1..n]$
3. "*Merge*" the 2 sorted lists.

Merge Sort

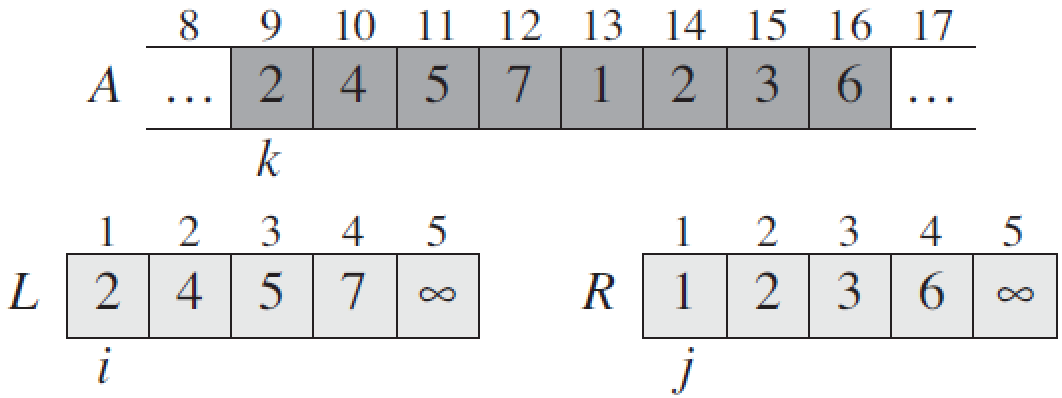
MERGE-SORT(A, p, r)

if $p < r$
    $q = \lfloor (p + r)/2 \rfloor$
    MERGE-SORT(A, p, q)
    MERGE-SORT$(A, q + 1, r)$
    MERGE(A, p, q, r)

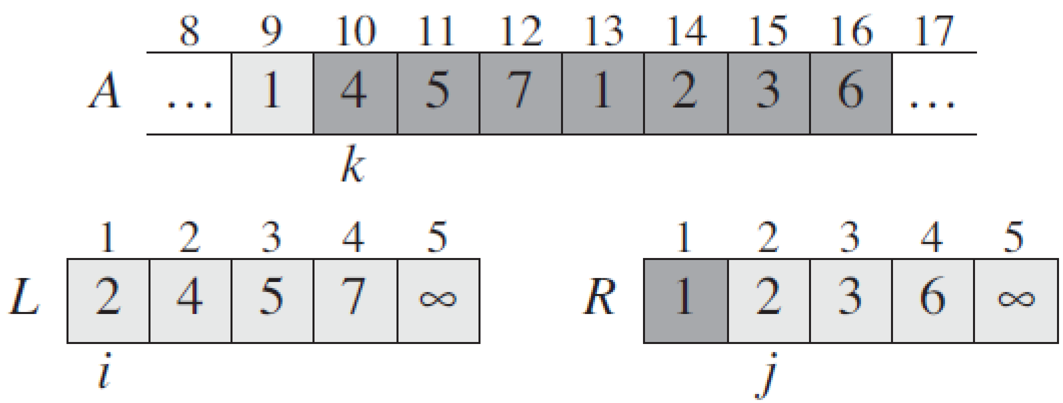
Merge() takes two sorted subarrays of A and merges them into a single sorted subarray of A (how long should this take?)

MERGE(A, p, q, r)

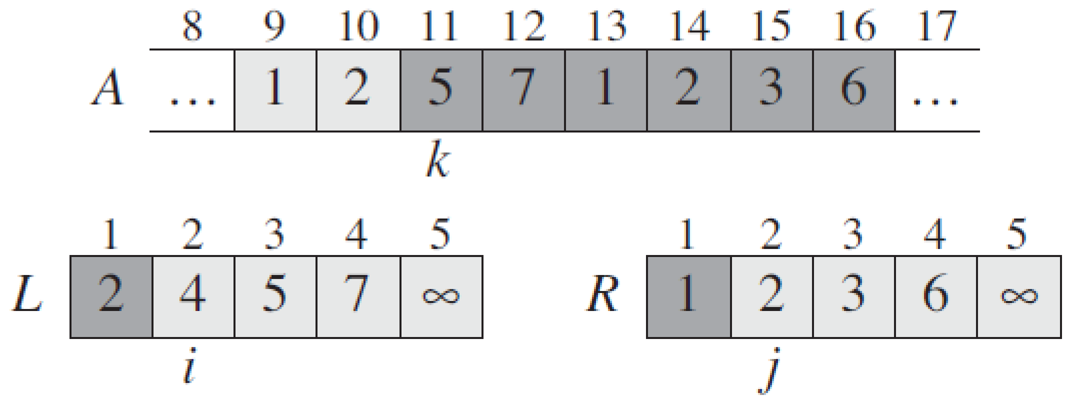
```
MERGE(A, p, q, r)
n1 = q - p + 1
n2 = r - q
let L[1 .. n1 + 1] and R[1 .. n2 + 1] be new arrays
for i = 1 to n1
    L[i] = A[p + i - 1]
for j = 1 to n2
    R[j] = A[q + j]
L[n1 + 1] = ∞
R[n2 + 1] = ∞
i = 1
j = 1
for k = p to r
    if L[i] ≤ R[j]
        A[k] = L[i]
        i = i + 1
    else A[k] = R[j]
        j = j + 1
```


8 9 10 11 12 13 14 15 16 17
A … 2 4 5 7 1 2 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j

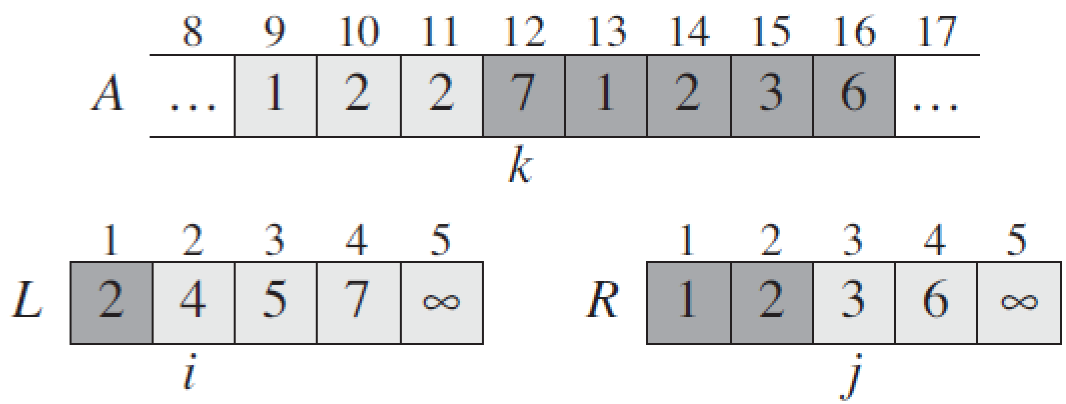
(a)


8 9 10 11 12 13 14 15 16 17
A … 1 4 5 7 1 2 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j

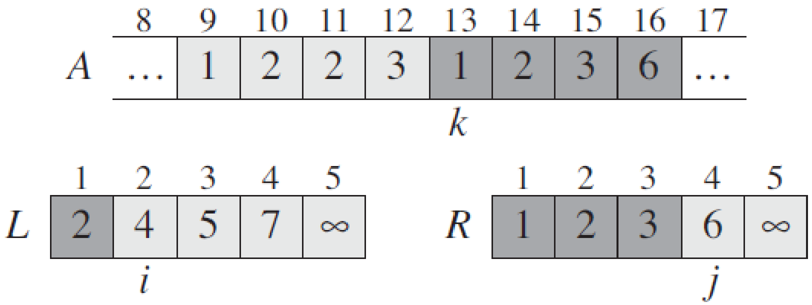
(b)


8 9 10 11 12 13 14 15 16 17
A … 1 2 5 7 1 2 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j

(c)

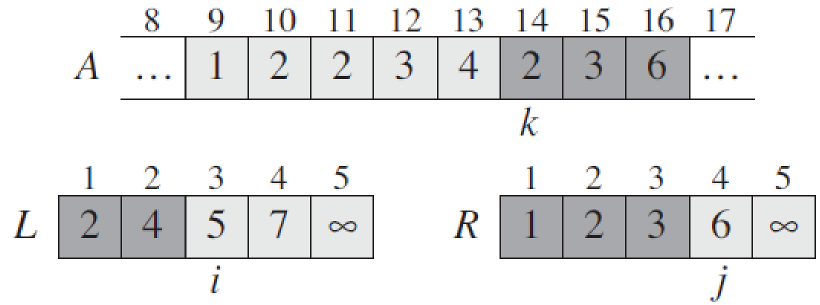

8 9 10 11 12 13 14 15 16 17
A … 1 2 2 7 1 2 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j

(d)



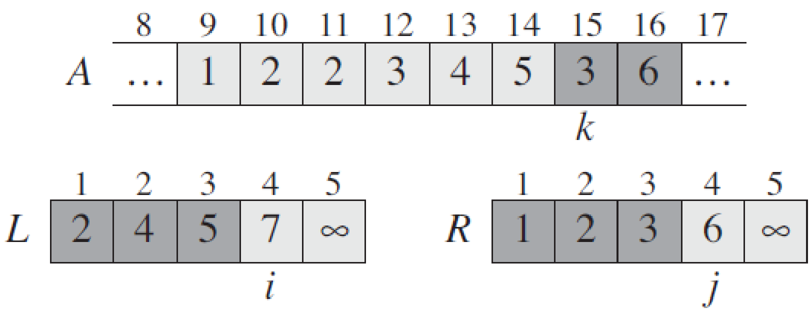
8 9 10 11 12 13 14 15 16 17
A … 1 2 2 3 1 2 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j

(e)



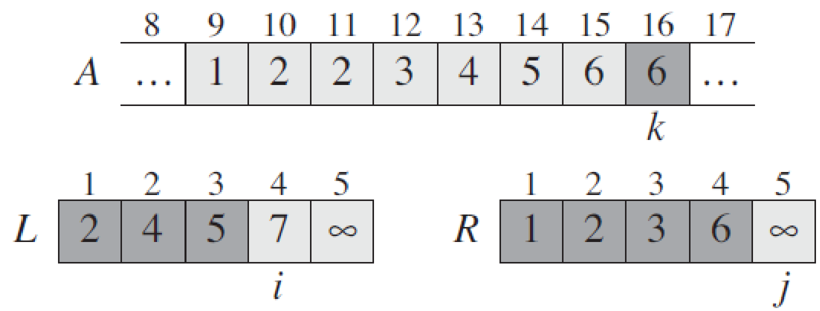
8 9 10 11 12 13 14 15 16 17
A … 1 2 2 3 4 2 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j

(f)



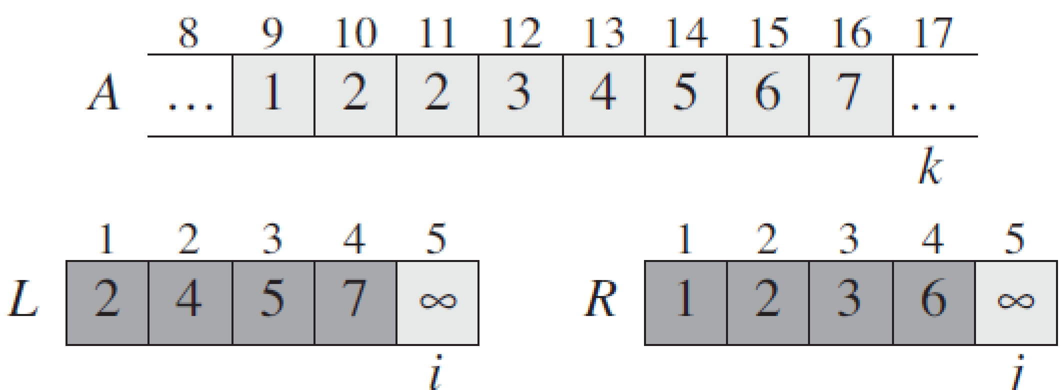
8 9 10 11 12 13 14 15 16 17
A … 1 2 2 3 4 5 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j

(g)



8 9 10 11 12 13 14 15 16 17
A … 1 2 2 3 4 5 6 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j

(h)



8 9 10 11 12 13 14 15 16 17
A … 1 2 2 3 4 5 6 7 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j

(i)

Analysis of Merge Sort

Statement	Effort

```
MergeSort(A, left, right) {                          T(n)
   if (left < right) {                               Θ(1)
      mid = floor((left + right) / 2);                  Θ(1)
      MergeSort(A, left, mid);                          T(n/2)
      MergeSort(A, mid+1, right);                       T(n/2)
      Merge(A, left, mid, right);                       Θ(n)
   }
}
```

So $T(n) = \begin{cases} \Theta(1) & \text{when } n = 1\text{, and} \\ 2T(n/2) + \Theta(n) & \text{when } n > 1 \end{cases}$

So what (more succinctly) is T(n)?

Recurrences

- The expression:

$$T(n) = \begin{cases} c & n = 1 \\ 2T\left(\dfrac{n}{2}\right) + cn & n > 1 \end{cases}$$

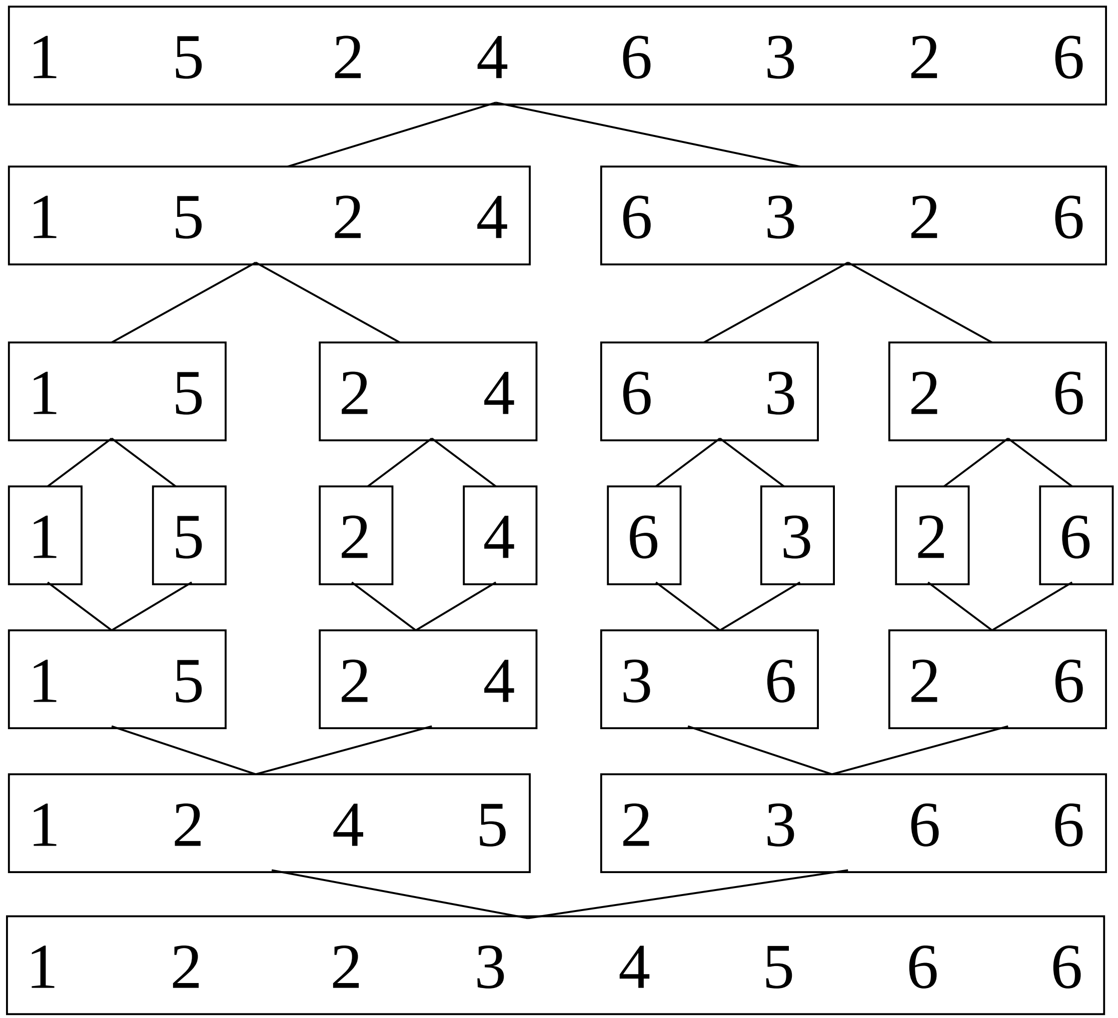
is a *recurrence*.

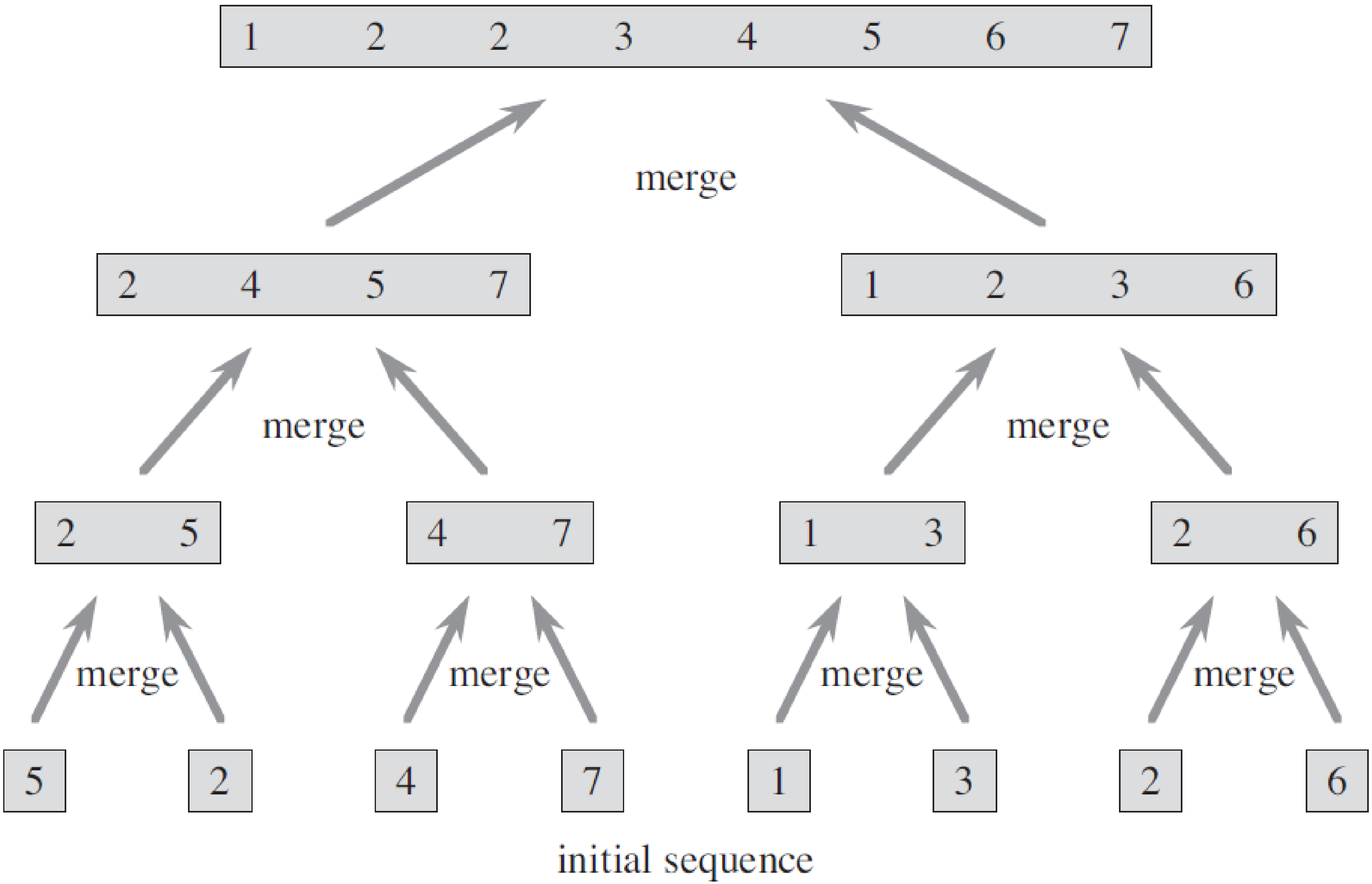
  – Recurrence: an equation that describes a function in terms of its value on smaller functions

Structure of merge sort algorithm

1. break problem into similar (smaller) subproblems
2. recursively solve subproblems
3. combine solutions to produce final answer

Example of Merge Sort




1 2 2 3 4 5 6 7
merge
2 4 5 7
1 2 3 6
merge
merge
2 5
4 7
1 3
2 6
merge
merge
merge
merge
5
2
4
7
1
3
2
6
initial sequence

Divide-and-conquer paradigm

1. *Divide* problem into subproblems.
2. *Conquer* subproblems by solving recursively.
3. *Combine* subproblem solutions.

Example of D&C Paradigm

Merge sort as Divide-and-conquer algorithm

1. Divide: Divide n-array into two $n/2$-subarrays.
2. Conquer: Sort the two subarrays recursively.
3. Combine: Linear-time merge.

Recurrence for Merge sort

subproblem size

$T(n) \quad = \quad 2 \quad T(\quad n/2 \quad)$

#subproblems

$+ \quad \Theta(n)$

work dividing & combining

$$T(n) = 2T(n/2) + \Theta(n)$$

A Useful Recurrence Relation

- Mergesort recurrence.

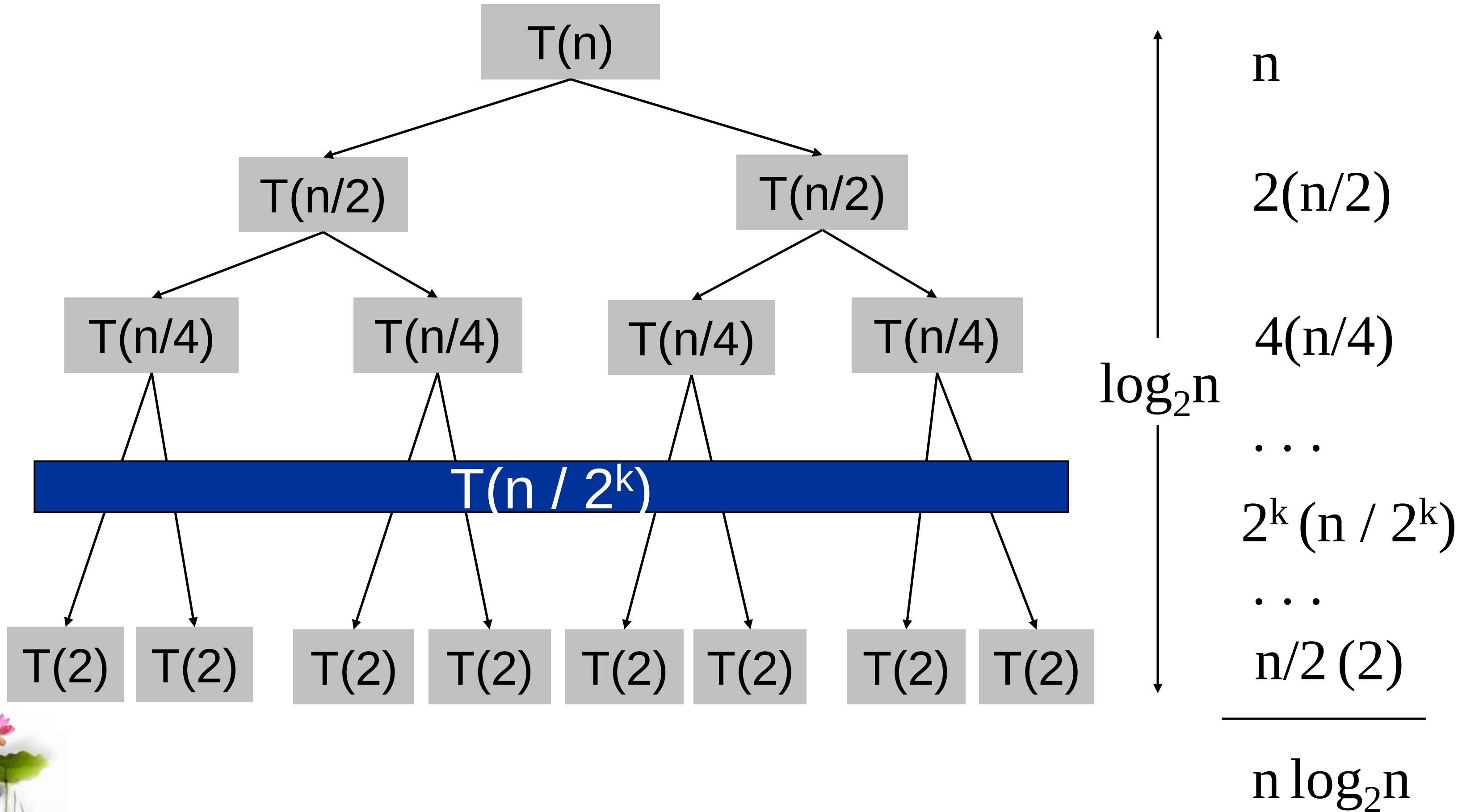
$$\mathrm{T}(n) \le \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{T(\lceil n/2 \rceil)}_{\text{solve left half}} + \underbrace{T(\lfloor n/2 \rfloor)}_{\text{solve right half}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

- Solution. T(n) = O(n $\log_2$ n).

- Assorted proofs. We describe several ways to prove this recurrence. Initially we assume n is a power of 2 and replace ≤ with =.

Proof by Recursion Tree

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$



Proof by Telescoping

- Claim. If T(n) satisfies this recurrence, then $T(n) = n \log_2 n$.

assumes n is a power of 2

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

- Pf. For n > 1:

$$\begin{aligned} \frac{T(n)}{n} &= \frac{2T(n/2)}{n} + 1 \\ &= \frac{T(n/2)}{n/2} + 1 \\ &= \frac{T(n/4)}{n/4} + 1 + 1 \\ &\cdots \\ &= \frac{T(n/n)}{n/n} + \underbrace{1 + \cdots + 1}_{\log_2 n} \\ &= \log_2 n \end{aligned}$$

Proof by Induction

- Claim. If T(n) satisfies this recurrence, then T(n) = n $\log_2$ n.

assumes n is a power of 2

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

- Pf. (by induction on n)
  - Base case: n = 1.
  - Inductive hypothesis: T(n) = n $\log_2$ n.
  - Goal: show that T(2n) = 2n $\log_2$ (2n).

$$\begin{aligned} T(2n) &= 2T(n) + 2n \\ &= 2n\log_2 n + 2n \\ &= 2n\big(\log_2(2n) - 1\big) + 2n \\ &= 2n\log_2(2n) \end{aligned}$$

Proof by Induction

- Claim. If T(n) satisfies this recurrence, then $T(n) = n \lceil \lg n \rceil$.

assumes n is a power of 2

$$T(n) \le \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{T(\lceil n/2 \rceil)}_{\text{solve left half}} + \underbrace{T(\lfloor n/2 \rfloor)}_{\text{solve right half}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

- Pf. (by induction on n)
  - Base case: n = 1.
  - Define $n_1 = \lfloor n/2 \rfloor$, $n_2 = \lceil n/2 \rceil$.
  - Induction step: assume true for 1, 2, ... , n–1.

$$\begin{aligned} T(n) &\le T(n_1) + T(n_2) + n \\ &\le n_1 \lceil \lg n_1 \rceil + n_2 \lceil \lg n_2 \rceil + n \\ &\le n_1 \lceil \lg n_2 \rceil + n_2 \lceil \lg n_2 \rceil + n \\ &= n \lceil \lg n_2 \rceil + n \\ &\le n(\lceil \lg n \rceil - 1) + n \\ &= n \lceil \lg n \rceil \end{aligned}$$

$$\begin{aligned} n_2 &= \lceil n/2 \rceil \\ &\le \left\lceil 2^{\lceil \lg n \rceil} / 2 \right\rceil \\ &= 2^{\lceil \lg n \rceil} / 2 \\ \Rightarrow \lg n_2 &\le \lceil \lg n \rceil - 1 \end{aligned}$$

Is there a Simple method?