Design and Analysis of Algorithms
CS218M

Designing Sorting Algorithms
Divide and Conquer

Paritosh Pandya

Indian Institute of Technology, Bombay

Autumn, 2022

Sorting Arrays

$\{A = A_0\}$ *SORT* $\{ perm(A, A_0) \wedge sorted \}$ where

Sorting Arrays

$\{A = A_0\}$ *SORT* $\{perm(A, A_0) \wedge sorted(A, 1, n)\}$ where

$Sorted(A, i, j) \stackrel{\text{def}}{=} \forall k : \; i \leq k < j.\; A[k] \leq A[k+1]$

Sorted(A, 2, 2-1) holds

Sorting Arrays

$\{A = A_0\}$ *SORT* $\{perm(A, A_0) \wedge sorted(A, 1, n)\}$ where

$$Sorted(A, i, j) \stackrel{\text{def}}{=} \forall k : \ i \leq k < j.\ A[k] \leq A[k+1]$$

Iterative Strategy

Permute the array such that first i elements are sorted.
Maintaining this, increment i in each iteration.

Sorting Arrays

$\{A = A_0\}$ *SORT* $\{perm(A, A_0) \wedge sorted(A, 1, n)\}$ where

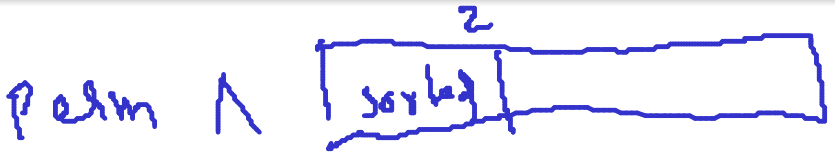
$$Sorted(A, i, j) \stackrel{\text{def}}{=} \forall k : \ i \leq k < j.\ A[k] \leq A[k+1]$$

Iterative Strategy

Permute the array such that first i elements are sorted.
Maintaining this, increment i in each iteration.

$$inv : \ perm(A, A_0) \wedge 0 \leq i \leq n \wedge sorted(A, 1, i)$$

Design Step 1



$\{A = A_0\}$ (1)

INIT

$\{inv : perm(A, A_0) \wedge 0 \leq i \leq n \wedge sorted(A, 1, i)\}$

$\{bound : \ n - i\}$

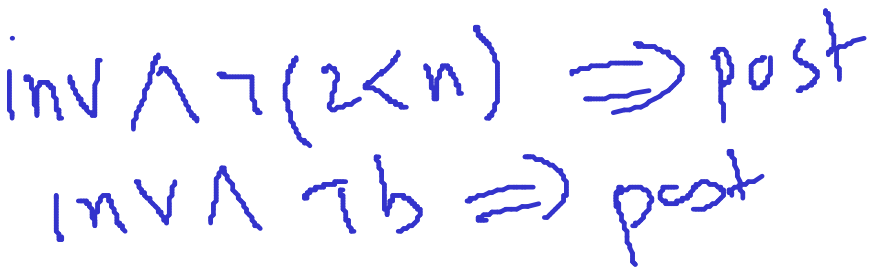
while $i < n$ do

BODY

i:= i+1

od

$\{\ perm(A, A_0) \wedge sorted(A, 1, n)\ \}$



Design Step 2A: Maintain stronger invariant

$i = 1$

$$inv1: \quad perm(A, A_0) \wedge 0 \leq i \leq n \wedge sorted(A, 1, i)$$
$$\wedge A[i+1..n] = A_0[i+1..n]$$

Design Step 2A: Maintain stronger invariant

$$inv1: \quad perm(A, A_0) \wedge 0 \leq i \leq n \wedge sorted(A, 1, i) \\ \wedge A[i+1..n] = A_0[i+1..n]$$

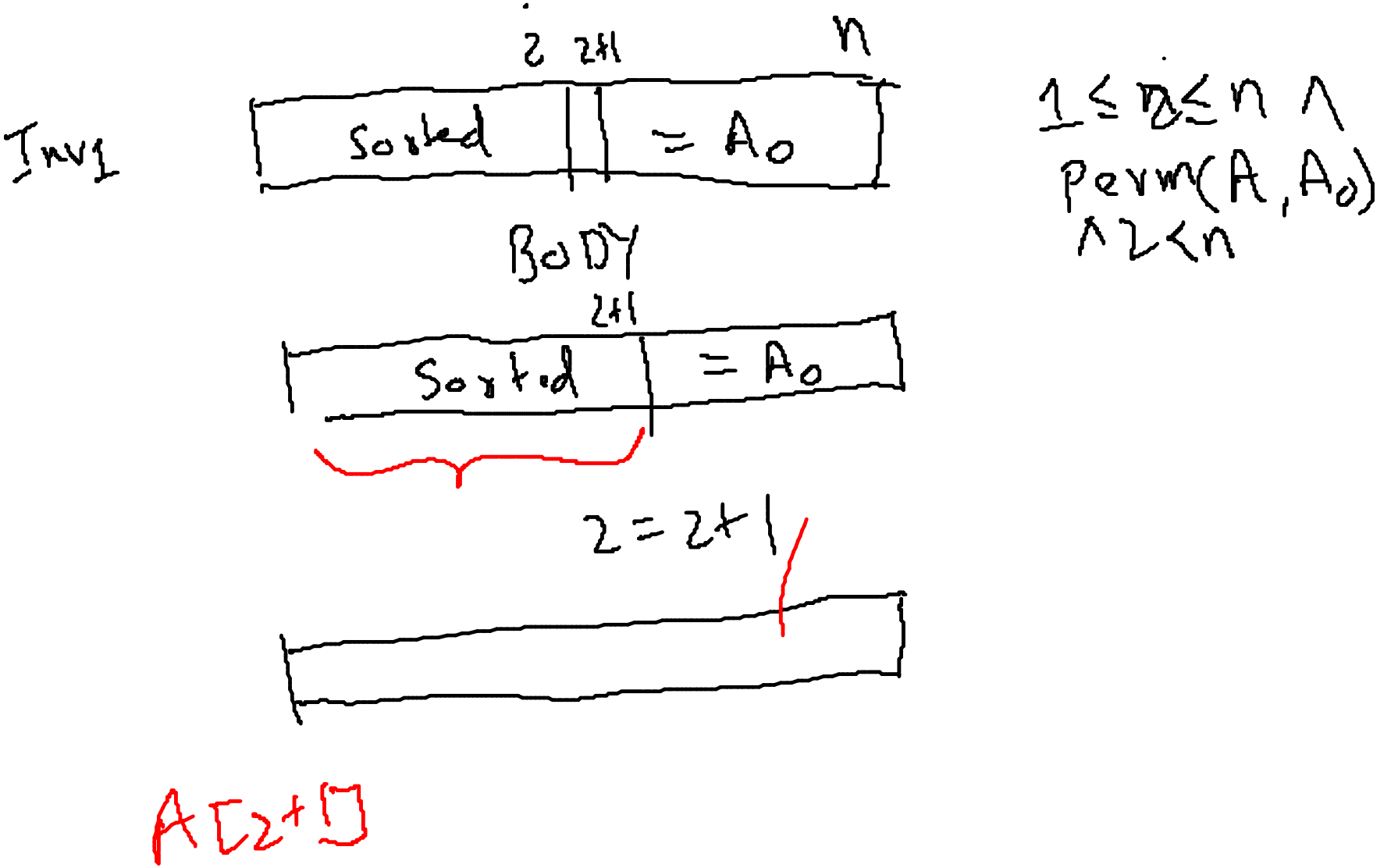
- Design INIT s.t. $Inv1$ holds after it. $i = 1$

Design Step 2A: Maintain stronger invariant

$$inv1: \quad perm(A, A_0) \wedge 0 \leq i \leq n \wedge sorted(A, 1, i) \\ \wedge A[i+1..n] = A_0[i+1..n]$$

- Design INIT s.t. $Inv1$ holds after it.
- Design BODY s.t. $\{Inv1 \wedge i < n\}$ $BODY$; $i = i + 1$ $\{Inv1\}$

Insertion Sort



Design Step 2B: Maintain stronger invariant

$$inv2 : perm(A, A_0) \wedge 0 \leq i \leq n \wedge sorted(A, 1, i)$$
$$\wedge \quad (\qquad A[i] \leq^* A[i+1..n] \ )$$

Design Step 2B: Maintain stronger invariant

i:=0

$$inv2 : perm(A, A_0) \wedge 0 \leq i \leq n \wedge sorted(A, 1, i)$$
$$\wedge \quad (\qquad A[i] \leq^* A[i+1..n] \ )$$

Design Step 2B: Maintain stronger invariant

i:=0

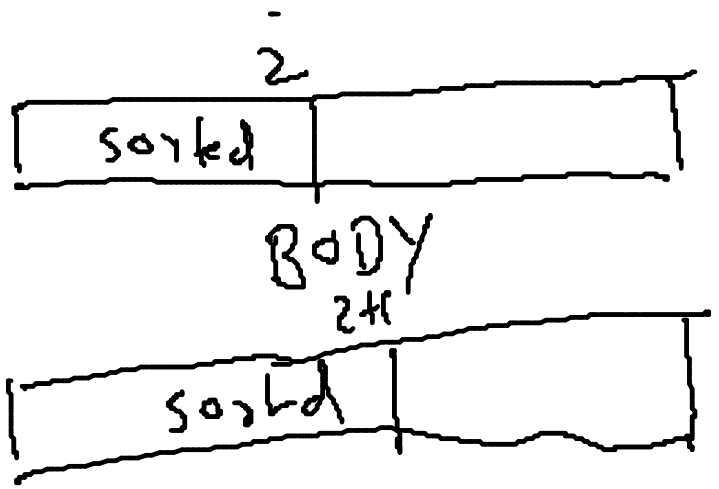
$$inv2 : perm(A, A_0) \wedge 0 \leq i \leq n \wedge sorted(A, 1, i)$$
$$\wedge \quad (i = 0 \vee A[i] \leq^* A[i+1..n] \ )$$

Design Step 2B: Maintain stronger invariant

i:=0

$$inv2 : perm(A, A_0) \wedge 0 \leq i \leq n \wedge sorted(A, 1, i) \\ \wedge \ (i = 0 \vee A[i] \leq^* A[i+1..n] \ )$$

Design BODY s.t. $\{Inv2 \wedge i < n\}$ $BODY$; $i = i + 1$ $\{Inv2\}$



$0 \leq i < n \wedge$ perm

$A[i] \leq^* A[i+1..n]$

find j: $A[j] =$ $\min(A[i+1..n])$

Design Step 2B: Maintain stronger invariant

i:=0

$$inv2 : perm(A, A_0) \wedge 0 \leq i \leq n \wedge sorted(A, 1, i) \\ \wedge \ (i = 0 \vee A[i] \leq^* A[i+1..n] \ )$$

Design BODY s.t. $\{Inv2 \wedge i < n\} \ \ BODY; \ \ i = i + 1 \ \{Inv2\}$

- (Selection sort)
  $j = FINDMIN(A, i + 1, n); \ \ SWAP(A[i + 1], A[j]$

Design Step 2B: Maintain stronger invariant

i:=0

$$inv2 : perm(A, A_0) \wedge 0 \leq i \leq n \wedge sorted(A, 1, i) \\ \wedge \ (i = 0 \vee A[i] \leq^* A[i+1..n] \ )$$

Design BODY s.t. $\{Inv2 \wedge i < n\}$ $BODY$; $i = i + 1$ $\{Inv2\}$

- (Selection sort)
  $j = FINDMIN(A, i+1, n)$; $SWAP(A[i+1], A[j]$
- (Bubble sort) Starting from $j = n$ down to $i + 2$, exchange $A[j], A[j-1]$ if $A[j] < A[j-1]$

Divide and Conquer: Finding Maximum in Array

$\{1 \leq p \leq r \leq n\}$ $FINDMAX(A, p, r)$ $\{ret = max(A[p..r])\}$

Divide and Conquer: Finding Maximum in Array

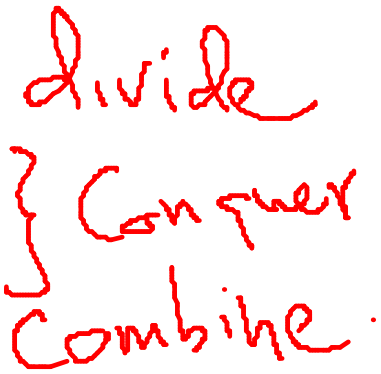
$\{1 \leq p \leq r \leq n\}$ $FINDMAX(A, p, r)$ $\{ret = max(A[p..r])\}$

```
1    FINDMAX(A, p, r)
2        if p < r
3            q = ⌊(p + r)/2⌋
5            a = FINDMAX(A, p, q)
6            b = FINDMAX(A, q + 1, r)
8            return max(a, b)
10       else
11           return A[p]
```

Divide and Conquer: Finding Maximum in Array

$\{1 \leq p \leq r \leq n\}$ $FINDMAX(A, p, r)$ $\{ret = max(A[p..r])\}$

```
1    FINDMAX(A, p, r)
2        if p < r
3            q = ⌊(p + r)/2⌋
5            a = FINDMAX(A, p, q)
6            b = FINDMAX(A, q + 1, r)
8            return max(a, b)
10       else
11           return A[p]
```



- Divide Divide Problem into a number of smaller sub problems.
- Conquer Solve the sub problems recursively.
- Combine Combine the results of subproblems to generate result for the whol problem.
- If the instance is small enough solve it directly.

Finding Maximum: Correctness

$\{1 \leq p \leq r \leq n\}$ $FINDMAX(A, p, r)$ $\{ret = max(A[p..r])\}$

Finding Maximum: Correctness

$\{1 \leq p \leq r \leq n\}$ $FINDMAX(A, p, r)$ $\{ret = max(A[p..r])\}$

```
1     FINDMAX(A,p,r)
2           if p < r
3                 q = ⌊(p + r)/2⌋
4
5                 a = FINDMAX(p, q)
6                 b = FINDMAX(q + 1, r)
7
7
8                 return max(a, b)
9
10          else
11                return A[p]
12
```

Finding Maximum: Correctness

$\{1 \leq p \leq r \leq n\}$ $FINDMAX(A, p, r)$ $\{ret = max(A[p..r])\}$

Unchang(A, p, r)

```
1     FINDMAX(A,p,r)
2           if p < r
3                 q = ⌊(p + r)/2⌋
4
5                 a = FINDMAX(p, q)
6                 b = FINDMAX(q + 1, r)
7
7
8                 return max(a, b)
9
10          else  {p = r}
11                return A[p]
12                {Post}
```

Finding Maximum: Correctness

$\{1 \leq p \leq r \leq n\}$ $FINDMAX(A, p, r)$ $\{ret = max(A[p..r])\}$

```
1    FINDMAX(A,p,r)
2        if p < r                {p < r}
3            q = ⌊(p + r)/2⌋
4            {1 ≤ p ≤ q < r ≤ n}
5            a = FINDMAX(p, q)
6            b = FINDMAX(q + 1, r)
7            {1 ≤ p ≤ q < r ≤ n∧
7                a = max(A[p..q]) ∧ b = max(A[q + 1..r] }
8            return max(a, b)
9            {Post}
10       else {p = r}
11           return A[p]
12           {Post}
```

FINDMAX: Asymptotic Time Complexity

$n = r - p + 1$

```
1    FINDMAX(A, p, r)
2        if p < r
3            q = ⌊(p + r)/2⌋                    Θ(1)
5            a = FINDMAX(A, p, q)               ⌈n/2⌉
6            b = FINDMAX(A, q + 1, r)           ⌊n/2⌋
8            return max(a, b)
10       else
11           return A[p]                        Θ(1)
```

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(1)$$

$$T(1) = \Theta(1)$$

FINDMAX: Asymptotic Time Complexity

```
1    FINDMAX(A, p, r)
2        if p < r
3            q = ⌊(p + r)/2⌋
5            a = FINDMAX(A, p, q)
6            b = FINDMAX(A, q + 1, r)
8            return max(a, b)
10       else
11           return A[p]
```

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

Recurrence

$$
\begin{aligned}
T(1) &= \Theta(1) \\
T(n) &= T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \theta(1)
\end{aligned}
$$

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

Find $f(n)$

Mergesort and Correctness

```
MERGE-SORT(A, p, r)
1  if p < r
2      q = ⌊(p + r)/2⌋
3      MERGE-SORT(A, p, q)
4      MERGE-SORT(A, q + 1, r)
5      MERGE(A, p, q, r)
```

Mergesort and Correctness

$\{pre: \ A = A_0 \wedge 1 \leq p \leq r \leq n\}$

$\{post: \ A[1..p-1] = A_0[1..p-1] \wedge A[r+1..n] = A_0[r+1..n] \wedge$
$permute(A, A_0) \wedge sorted(A, p, r) \ \}$

MERGE-SORT(A, p, r)

1 **if** $p < r$
2 $\quad q = \lfloor (p+r)/2 \rfloor$
3 $\quad$ MERGE-SORT(A, p, q)
4 $\quad$ MERGE-SORT$(A, q+1, r)$
5 $\quad$ MERGE(A, p, q, r)

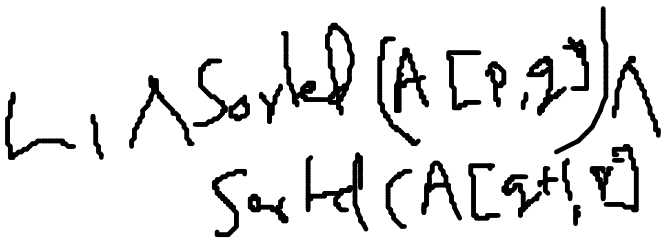
$1 \leq p \leq q < q+1 \leq r \leq n$

Mergesort and Correctness

$\{pre: \ A = A_0 \wedge 1 \leq p \leq r \leq n\}$

$\{post: \ A[1..p-1] = A_0[1..p-1] \wedge A[r+1..n] = A_0[r+1..n] \wedge$
$permute(A, A_0) \wedge sorted(A, p, r) \ \}$

```
MERGE-SORT(A, p, r)
1  if p < r
2      q = ⌊(p + r)/2⌋
3      MERGE-SORT(A, p, q)
4      MERGE-SORT(A, q + 1, r)
5      MERGE(A, p, q, r)
```

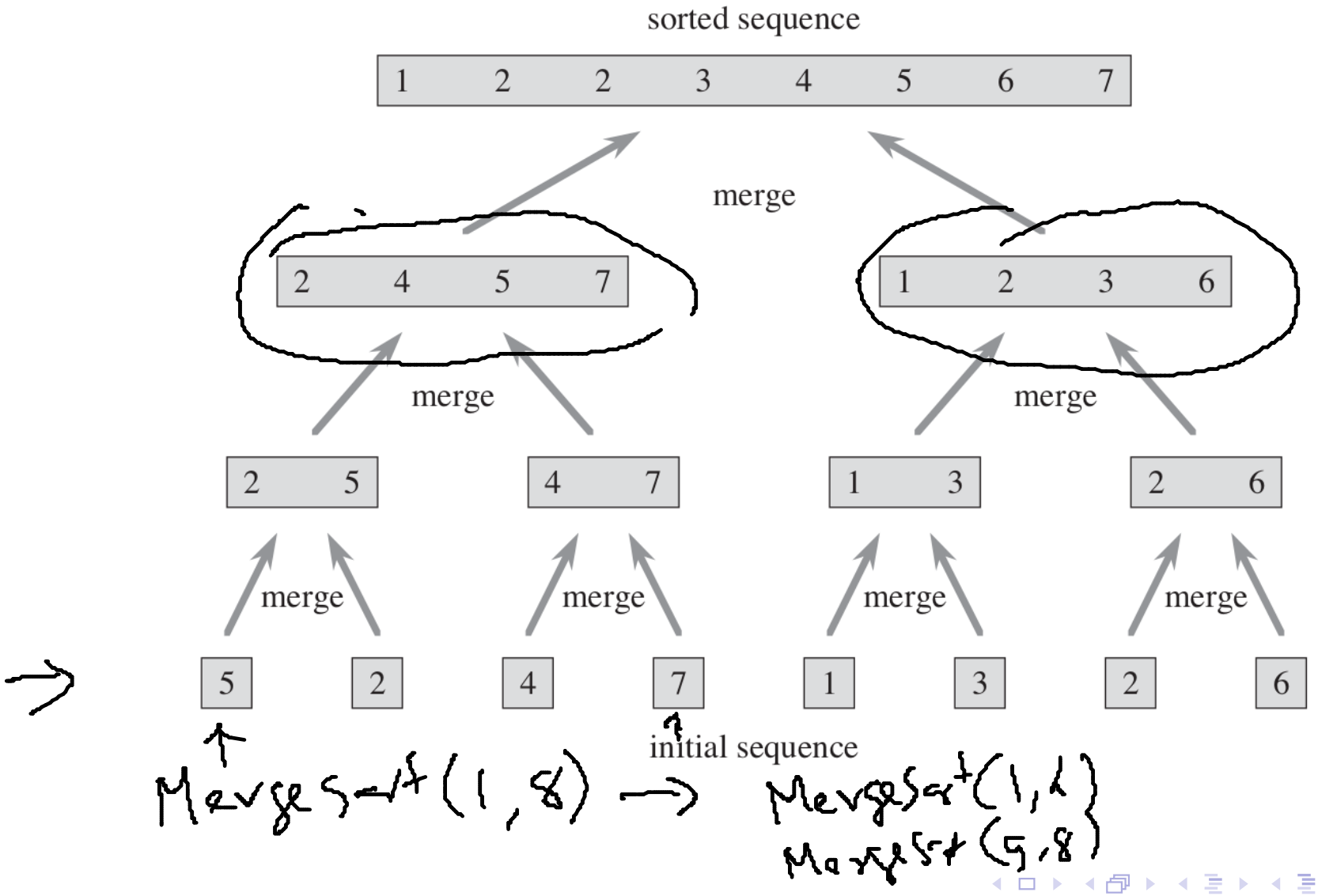



Merge(A,p,q,r)

$\{pre: \ A = B_0 \wedge 1 \leq p \leq q < r \leq n \wedge$
$sorted(A, p, q) \wedge sorted(A, q+1, r)\}$

$\{post: \ A[1..p-1] = B_0[1..p-1] \wedge A[r+1..n] = B_0[r+1..n] \wedge$
$permute(A, B_0) \wedge sorted(A, p, r) \ \}$

Mergesort Execution



Mergesort: Asymptotic Time Complexity

$n = (r - p) + 1$

```
MERGE-SORT(A, p, r)
1  if p < r
2      q = ⌊(p + r)/2⌋
3      MERGE-SORT(A, p, q)
4      MERGE-SORT(A, q + 1, r)
5      MERGE(A, p, q, r)
```

$\lceil n/2 \rceil$

$\lfloor n/2 \rfloor$

$\Theta(1)$ D

$\Theta(n)$ C

$\Theta(1)$

else return

$T(1) = \Theta(1)$

$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n)$ $n > 1$

Mergesort: Asymptotic Time Complexity

```
MERGE-SORT(A, p, r)
1  if p < r
2      q = ⌊(p + r)/2⌋
3      MERGE-SORT(A, p, q)
4      MERGE-SORT(A, q + 1, r)
5      MERGE(A, p, q, r)
```

Recurrence

$$\begin{aligned} T(1) &= \Theta(1) \\ T(n) &= T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \theta(n) \end{aligned}$$

Find $f(n)$ s.t. $T(n) = \Theta(f(n))$

Solving Recurrences

Recurrence

$$\begin{aligned} T(1) &= c_1 \\ T(n) &= 2 * T(n/2) + c_2 * n \cdot \qquad \text{for } n > 1 \end{aligned}$$

Solve it to show that $T(n) = \Theta(f(n))$

Find $f(n)$

Three Methods:

- Substitution: Guess $T(n) = \Theta(f(n))$ and verify using Induction.
- Recusion Tree: A method to discover appropriate $f(n)$. To be verified by Induction.
- Master Theorem Gives solution directly in many cases.

?

Principle of Complete Induction

Notation $\lg(n) = \log_2(n)$

$P(2), P(3)$ base

A method to show that for all n predicate $P(n)$ is true.

Example: $P(n) \stackrel{\text{def}}{=} T(n) \leq kn\lg(n)$.

$$\frac{\forall n.\ (\forall m : m < n.\ P(m)) \Rightarrow P(n)}{\forall n.P(n)}$$

$\forall n > 3.\ P(n_2) \Rightarrow P(n)$

$\forall n > 3.\ P(n)$

$P(1) \wedge P(2) \Rightarrow P(3)$

$P(1) \Rightarrow P(2)$

$True \Rightarrow P(1)$

$\therefore$

$P(3)$

$T(1) = c_1$

$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + c_2$

To prove $T(n) = \Theta(n)$

$T(n) = O(n)$

$T(n) = \Omega(n)$

claim $\forall n \; T(n) \leq k_1 n - k_2$

claim $\forall n \; T(n) \geq k_3 n$