Divide-and-Conquer

Murat Osmanoglu

Divide-and Conquer

- probably the best known design technique

Divide-and Conquer

- probably the best known design technique
- similar to Decrease-and-Conquer, the technique exploits the relationship between a solution of a given instance of a problem and a solution of its smaller instance

Divide-and Conquer

- probably the best known design technique
- similar to Decrease-and-Conquer, the technique exploits the relationship between a solution of a given instance of a problem and a solution of its smaller instance
  - divide the problem into a number of subproblems
  - solve each subproblem recursively
  - combine solutions to obtain a solution for the original problem

Divide-and Conquer

- probably the best known design technique
- similar to Decrease-and-Conquer, the technique exploits the relationship between a solution of a given instance of a problem and a solution of its smaller instance
  - divide the problem into a number of subproblems
  - solve each subproblem recursively
  - combine solutions to obtain a solution for the original problem
- in general, subproblems are independent of each other

Divide-and Conquer

Multiplication of Large Integers

- Given two n-digit integers a and b, compute $a \times b$
  (especially in modern crypto, some algorithms deal with integers having more than 500 digits)

Divide-and Conquer

Multiplication of Large Integers

- Given two n-digit integers a and b, compute a × b
  (especially in modern crypto, some algorithms deal with integers having more than 500 digits)

- brute-force solution :

$$\begin{array}{r} 123456 \\ \times\ 654321 \\ \hline \end{array}$$

Divide-and Conquer

Multiplication of Large Integers

- Given two n-digit integers a and b, compute a × b
  (especially in modern crypto, some algorithms deal with integers having more than 500 digits)

- brute-force solution :

$$
\begin{array}{r}
123456 \\
\times\ 654321 \\
\hline
123456
\end{array}
$$

Divide-and Conquer

Multiplication of Large Integers

- Given two n-digit integers a and b, compute a × b
  (especially in modern crypto, some algorithms deal with integers having more than 500 digits)

- brute-force solution :

$$\begin{array}{r} 123456 \\ \times\ 654321 \\ \hline 123456 \\ 246912 \end{array}$$

Divide-and Conquer

Multiplication of Large Integers

- Given two n-digit integers a and b, compute a × b
  (especially in modern crypto, some algorithms deal with integers having more than 500 digits)

- brute-force solution :

```
   123456
✖  654321
---------
   123456
  246912
     .
     .
     .
```

Divide-and Conquer

Multiplication of Large Integers

- Given two n-digit integers a and b, compute a × b
  (especially in modern crypto, some algorithms deal with integers having more than 500 digits)

- brute-force solution :

```
    123456
  ✖ 654321
  --------
    123456
   246912
      .
      .
      .
```

  – takes $O(n^2)$,

Divide-and Conquer

Multiplication of Large Integers

- Given two n-digit integers a and b, compute a × b
  (especially in modern crypto, some algorithms deal with integers having more than 500 digits)

- brute-force solution :

$$
\begin{array}{r}
123456 \\
\times \quad 654321 \\
\hline
123456 \\
246912 \\
\vdots
\end{array}
$$

  - takes $O(n^2)$, i.e. multiply each digit of the second one with the digits of the first one, put them in the correct positions and calculate the final sum

Divide-and Conquer

Multiplication of Large Integers

- Given two n-digit integers a and b, compute a × b
  (especially in modern crypto, some algorithms deal with integers having more than 500 digits)

- brute-force solution :

```
      123456
  ✖   654321
  ----------
      123456
     246912
        .
        .
        .
```

Can we get better one ?

  - takes $O(n^2)$, i.e. multiply each digit of the second one with the digits of the first one, put them in the correct positions and calculate the final sum

Divide-and Conquer

Multiplication of Large Integers

- rewrite the integers $a = 123456$ and $b = 654321$ as

$$a = 123000 + 456, \qquad b = 654000 + 321$$

Divide-and Conquer

Multiplication of Large Integers

- rewrite the integers $a = 123456$ and $b = 654321$ as

  $a = 123000 + 456, \qquad b = 654000 + 321$

  – thus, $a \times b = 123{\times}654{\times}10^6 + (123{\times}321 + 456{\times}654) \times 10^3 + 456{\times}321$

Divide-and Conquer

Multiplication of Large Integers

- rewrite the integers $a = 123456$ and $b = 654321$ as

  $a = 123000 + 456, \qquad b = 654000 + 321$

  - thus, $a \times b = 123\times654\times10^6 + (123\times321 + 456\times654) \times10^3 + 456\times321$

Multiply(a, b, n)

input : two n-digit numbers
output : $a \times b$
if $n \le 1$
    return $a \times b$
$a \leftarrow a_1\times10^{n/2} + a_2$; $b \leftarrow b_1\times10^{n/2} + b_2$
$A \leftarrow$ Multiply(a_1, b_1, $n/2$) ; $B \leftarrow$ Multiply(a_1, b_2, $n/2$)
$C \leftarrow$ Multiply(a_2, b_1, $n/2$) ; $D \leftarrow$ Multiply(a_2, b_2, $n/2$)
return $A\times10^n + (B + C) \times10^{n/2} + D$

Divide-and Conquer

Multiplication of Large Integers

Multiply(a, b, n)

input : two n-digit numbers
output : $a \times b$

if $n \leq 1$
    return $a \times b$

$a \leftarrow a_1 \times 10^{n/2} + a_2$; $b \leftarrow b_1 \times 10^{n/2} + b_2$
$A \leftarrow$ Multiply(a_1, b_1, n/2) ; $B \leftarrow$ Multiply(a_1, b_2, n/2)
$C \leftarrow$ Multiply(a_2, b_1, n/2) ; $D \leftarrow$ Multiply(a_2, b_2, n/2)

return $A \times 10^n + (B + C) \times 10^{n/2} + D$

Divide-and Conquer

Multiplication of Large Integers

Multiply(a, b, n)

input : two n-digit numbers
output : $a \times b$

if $n \leq 1$
    return $a \times b$

$a \leftarrow a_1 \times 10^{n/2} + a_2$; $b \leftarrow b_1 \times 10^{n/2} + b_2$
$A \leftarrow$ Multiply(a_1, b_1, n/2) ; $B \leftarrow$ Multiply(a_1, b_2, n/2)
$C \leftarrow$ Multiply(a_2, b_1, n/2) ; $D \leftarrow$ Multiply(a_2, b_2, n/2)

return $A \times 10^n + (B + C) \times 10^{n/2} + D$

- recurrence relation for the running time

$$T(n) = 4T(n/2) + O(n)$$

Divide-and Conquer

Multiplication of Large Integers

Multiply(a, b, n)

input : two n-digit numbers
output : $a \times b$

if $n \leq 1$
    return $a \times b$

$a \leftarrow a_1 \times 10^{n/2} + a_2$; $b \leftarrow b_1 \times 10^{n/2} + b_2$
$A \leftarrow$ Multiply(a_1, b_1, n/2) ; $B \leftarrow$ Multiply(a_1, b_2, n/2)
$C \leftarrow$ Multiply(a_2, b_1, n/2) ; $D \leftarrow$ Multiply(a_2, b_2, n/2)

return $A \times 10^n + (B + C) \times 10^{n/2} + D$

- recurrence relation for the running time

  $T(n) = 4T(n/2) + O(n)$

  where $O(n)$ accounts for partitioning, additions and shifting (merging time)

Divide-and Conquer

Multiplication of Large Integers

Multiply(a, b, n)

input : two n-digit numbers
output : $a \times b$

if $n \leq 1$
  return $a \times b$

$a \leftarrow a_1 \times 10^{n/2} + a_2$; $b \leftarrow b_1 \times 10^{n/2} + b_2$
$A \leftarrow$ Multiply(a_1, b_1, n/2) ; $B \leftarrow$ Multiply(a_1, b_2, n/2)
$C \leftarrow$ Multiply(a_2, b_1, n/2) ; $D \leftarrow$ Multiply(a_2, b_2, n/2)

return $A \times 10^n + (B + C) \times 10^{n/2} + D$

- recurrence relation for the running time

  $T(n) = 4T(n/2) + O(n)$

  where O(n) accounts for partitioning, additions and shifting (merging time)

- if you apply Master Theorem, $T(n) = O(n^2)$

Divide-and Conquer

Multiplication of Large Integers

Karatsuba(a, b, n)

input : two n-digit numbers
output : $a \times b$

if $n \leq 1$
    return $a \times b$

$a \leftarrow a_1 \times 10^{n/2} + a_2$; $b \leftarrow b_1 \times 10^{n/2} + b_2$
$A \leftarrow$ Karatsuba(a_1, b_1, $n/2$)
$B \leftarrow$ Karatsuba(a_2, b_2, $n/2$)
$C \leftarrow$ Karatsuba($a_1 + a_2$, $b_1 + b_2$, $n/2$)

return $A \times 10^n + (C - A - B) \times 10^{n/2} + B$

Divide-and Conquer

Multiplication of Large Integers

Karatsuba(a, b, n)

input : two n-digit numbers
output : $a \times b$

if $n \leq 1$
    return $a \times b$

$a \leftarrow a_1 \times 10^{n/2} + a_2$; $b \leftarrow b_1 \times 10^{n/2} + b_2$
$A \leftarrow$ Karatsuba(a_1, b_1, $n/2$)
$B \leftarrow$ Karatsuba(a_2, b_2, $n/2$)
$C \leftarrow$ Karatsuba($a_1 + a_2$, $b_1 + b_2$, $n/2$)

return $A \times 10^n + (C - A - B) \times 10^{n/2} + B$

- recurrence relation for the running time

$$T(n) = 3T(n/2) + O(n)$$

Divide-and Conquer

Multiplication of Large Integers

Karatsuba(a, b, n)

input : two n-digit numbers
output : $a \times b$

if $n \leq 1$
    return $a \times b$

$a \leftarrow a_1 \times 10^{n/2} + a_2$; $b \leftarrow b_1 \times 10^{n/2} + b_2$
$A \leftarrow$ Karatsuba(a_1, b_1, $n/2$)
$B \leftarrow$ Karatsuba(a_2, b_2, $n/2$)
$C \leftarrow$ Karatsuba($a_1 + a_2$, $b_1 + b_2$, $n/2$)

return $A \times 10^n + (C - A - B) \times 10^{n/2} + B$

- recurrence relation for the running time

  $T(n) = 3T(n/2) + O(n)$

  where $O(n)$ accounts for partitioning, additions (merging time)

Divide-and Conquer

Multiplication of Large Integers

Karatsuba(a, b, n)

input : two n-digit numbers
output : $a \times b$

if $n \leq 1$
    return $a \times b$

$a \leftarrow a_1 \times 10^{n/2} + a_2$; $b \leftarrow b_1 \times 10^{n/2} + b_2$
$A \leftarrow$ Karatsuba(a_1, b_1, $n/2$)
$B \leftarrow$ Karatsuba(a_2, b_2, $n/2$)
$C \leftarrow$ Karatsuba($a_1 + a_2$, $b_1 + b_2$, $n/2$)

return $A \times 10^n + (C - A - B) \times 10^{n/2} + B$

- recurrence relation for the running time

  $T(n) = 3T(n/2) + O(n)$

  where $O(n)$ accounts for partitioning, additions (merging time)

- if you apply Master Theorem, $T(n) = O(n^{1.585})$

Divide-and-Conquer
(Sorting Problem)

Mergesort

(divide the elements according to their position in the array)

Divide-and-Conquer (Sorting Problem)

Mergesort

(divide the elements according to their position in the array)

- given an array of n orderable items $[a_1, a_2, \ldots, a_n]$, reorder the items as $[a_1', a_2', \ldots, a_n']$ such that $a_1' \le a_2' \le \ldots \le a_n'$

Divide-and-Conquer (Sorting Problem)

Mergesort

(divide the elements according to their position in the array)

- given an array of n orderable items $[a_1, a_2, \ldots, a_n]$, reorder the items as $[a_1', a_2', \ldots, a_n']$ such that $a_1' \leq a_2' \leq \ldots \leq a_n'$
- divide the given sequence into two halves, sort each of them recursively, and merge the smaller sorted array into a single sorted array

Divide-and-Conquer
(Sorting Problem)

Mergesort
(divide the elements according to their position in the array)

- given an array of n orderable items $[a_1, a_2, \ldots, a_n]$, reorder the items as $[a_1', a_2', \ldots, a_n']$ such that $a_1' \le a_2' \le \ldots \le a_n'$
- divide the given sequence into two halves, sort each of them recursively, and merge the smaller sorted array into a single sorted array

Merge-Sort(X[1,n], p, r)

input: *an array of n orderable items*
output : *sorted array of n items*

```
if p < r
   q ← (p + r)/2
   Merge-Sort(X, p, q)
   Merge-Sort(X, q + 1, r)
   Merge(X, p, q, r)
```

Divide-and-Conquer
(Sorting Problem)

Mergesort
(divide the elements according to their position in the array)

- given an array of n orderable items $[a_1, a_2, \ldots, a_n]$, reorder the items as $[a_1', a_2', \ldots, a_n']$ such that $a_1' \leq a_2' \leq \ldots \leq a_n'$
- divide the given sequence into two halves, sort each of them recursively, and merge the smaller sorted array into a single sorted array

Merge-Sort(X[1,n], p, r)

input: *an array of n orderable items*
output : *sorted array of n items*

```
if p < r
    q ← (p + r)/2
    Merge-Sort(X, p, q)
    Merge-Sort(X, q + 1, r)
    Merge(X, p, q, r)
```

$T(n) = \theta(1)$ if $n = 1$

$T(n) = 2.T(n/2) + f(n)$ if $n > 1$

Divide-and-Conquer
(Sorting Problem)

Mergesort
(divide the elements according to their position in the array)

- given an array of n orderable items $[a_1, a_2, \ldots, a_n]$, reorder the items as $[a_1', a_2', \ldots, a_n']$ such that $a_1' \le a_2' \le \ldots \le a_n'$
- divide the given sequence into two halves, sort each of them recursively, and merge the smaller sorted array into a single sorted array

Merge-Sort(X[1,n], p, r)

input: *an array of n orderable items*
output : *sorted array of n items*

```
if p < r
    q ← (p + r)/2
    Merge-Sort(X, p, q)
    Merge-Sort(X, q + 1, r)
    Merge(X, p, q, r)
```

$T(n) = \theta(1)$ if $n = 1$

$T(n) = 2.T(n/2) + f(n)$ if $n > 1$

merging time when $i = 1$ and $j = n$

Divide-and-Conquer
(Sorting Problem)

Mergesort

Merge(X[1,n], p, q, r)

```
a ← q − p + 1
b ← r − q
let L[1, a + 1] and R[1, b + 1] be new arrays
copy X[p, q] to L[1, a]
copy X[q+1, r] to R[1, b]
L[a+1] ← ∞ ; R[b+1] ← ∞
i ← 1 ; j ← 1
for k = p to r
    if L[i] ≤ R[j]
        X[k] ← L[i]
        i ← i + 1
    else
        X[k] ← R[j]
        j ← j + 1
```

Divide-and-Conquer
(Sorting Problem)

Mergesort

Merge(X[1,n], p, q, r)

a ← q – p + 1
b ← r – q
let L[1, a + 1] and R[1, b + 1] be new arrays
copy X[p, q] to L[1, a]
copy X[q+1, r] to R[1, b]
L[a+1] ← ∞ ; R[b+1] ← ∞
i ← 1 ; j ← 1
for k = p to r
    if L[i] ≤ R[j]
        X[k] ← L[i]
        i ← i + 1
    else
        X[k] ← R[j]
        j ← j + 1

1	2	5	9	3	4	7	10	X

Divide-and-Conquer
(Sorting Problem)

Mergesort

Merge(X[1,n], p, q, r)

```
a ← q – p + 1
b ← r – q
let L[1, a + 1] and R[1, b + 1] be new arrays
copy X[p, q] to L[1, a]
copy X[q+1, r] to R[1, b]
L[a+1] ← ∞ ; R[b+1] ← ∞
i ← 1 ; j ← 1
for k = p to r
    if L[i] ≤ R[j]
        X[k] ← L[i]
        i ← i + 1
    else
        X[k] ← R[j]
        j ← j + 1
```

p = 1   q = 4   r = 8

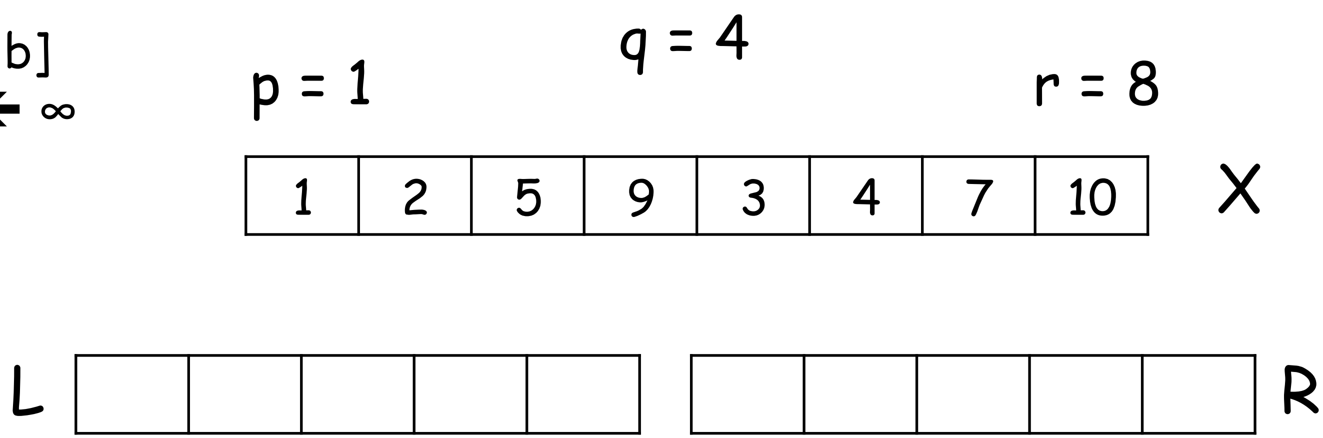
1	2	5	9	3	4	7	10	X

Divide-and-Conquer
(Sorting Problem)

Mergesort

```
Merge(X[1,n], p, q, r)

a ← q – p + 1
b ← r – q
let L[1, a + 1] and R[1, b + 1] be new arrays
copy X[p, q] to L[1, a]
copy X[q+1, r] to R[1, b]
L[a+1] ← ∞ ; R[b+1] ← ∞
i ← 1 ; j ← 1
for k = p to r
     if L[i] ≤ R[j]
          X[k] ← L[i]
          i ← i + 1
     else
          X[k] ← R[j]
          j ← j + 1
```



Divide-and-Conquer
(Sorting Problem)

Mergesort

```
Merge(X[1,n], p, q, r)

a ← q – p + 1
b ← r – q
let L[1, a + 1] and R[1, b + 1] be new arrays
copy X[p, q] to L[1, a]
copy X[q+1, r] to R[1, b]
L[a+1] ← ∞ ; R[b+1] ← ∞
i ← 1 ; j ← 1
for k = p to r
    if L[i] ≤ R[j]
        X[k] ← L[i]
        i ← i + 1
    else
        X[k] ← R[j]
        j ← j + 1
```

$p = 1$ $q = 4$ $r = 8$

X

1	2	5	9	3	4	7	10

L

1	2	5	9	∞

R

3	4	7	10	∞

Divide-and-Conquer
(Sorting Problem)

Mergesort

Merge(X[1 ,n], p, q, r)

```
a ← q – p + 1
b ← r – q
let L[1, a + 1] and R[1, b + 1] be new arrays
copy X[p, q] to L[1, a]
copy X[q+1, r] to R[1, b]
L[a+1] ← ∞ ; R[b+1] ← ∞
i ← 1 ; j ← 1
for k = p to r
    if L[i] ≤ R[j]
        X[k] ← L[i]
        i ← i + 1
    else
        X[k] ← R[j]
        j ← j + 1
```

q = 4

p = 1   r = 8

X

1	2	5	9	3	4	7	10

k = 1

L

1	2	5	9	∞

i = 1

R

3	4	7	10	∞

j = 1

Divide-and-Conquer
(Sorting Problem)

Mergesort

Merge(X[1 ,n], p, q, r)

```
a ← q – p + 1
b ← r – q
let L[1, a + 1] and R[1, b + 1] be new arrays
copy X[p, q] to L[1, a]
copy X[q+1, r] to R[1, b]
L[a+1] ← ∞ ; R[b+1] ← ∞
i ← 1 ; j ← 1
for k = p to r
    if L[i] ≤ R[j]
        X[k] ← L[i]
        i ← i + 1
    else
        X[k] ← R[j]
        j ← j + 1
```

$q = 4$

$p = 1$ $r = 8$

X

1	2	5	9	3	4	7	10

$k = 1$

L

1	2	5	9	∞

$i = 1$

R

3	4	7	10	∞

$j = 1$

Divide-and-Conquer
(Sorting Problem)

Mergesort

Merge(X[1,n], p, q, r)

```
a ← q - p + 1
b ← r - q
let L[1, a + 1] and R[1, b + 1] be new arrays
copy X[p, q] to L[1, a]
copy X[q+1, r] to R[1, b]
L[a+1] ← ∞ ; R[b+1] ← ∞
i ← 1 ; j ← 1
for k = p to r
    if L[i] ≤ R[j]
        X[k] ← L[i]
        i ← i + 1
    else
        X[k] ← R[j]
        j ← j + 1
```

$p = 1$ $q = 4$ $r = 8$

X

1	2	5	9	3	4	7	10

$k = 1$

L

1	2	5	9	∞

$i = 1$

R

3	4	7	10	∞

$j = 1$

Divide-and-Conquer
(Sorting Problem)

Mergesort

```
Merge(X[1,n], p, q, r)

a ← q − p + 1
b ← r − q
let L[1, a + 1] and R[1, b + 1] be new arrays
copy X[p, q] to L[1, a]
copy X[q+1, r] to R[1, b]
L[a+1] ← ∞ ; R[b+1] ← ∞
i ← 1 ; j ← 1
for k = p to r
    if L[i] ≤ R[j]
        X[k] ← L[i]
        i ← i + 1
    else
        X[k] ← R[j]
        j ← j + 1
```

p = 1 q = 4 r = 8

X:

1	2	5	9	3	4	7	10

k = 1

L:

1	2	5	9	∞

i = 2

R:

3	4	7	10	∞

j = 1

Divide-and-Conquer
(Sorting Problem)

Mergesort

Merge(X[1 ,n], p, q, r)

```
a ← q – p + 1
b ← r – q
let L[1, a + 1] and R[1, b + 1] be new arrays
copy X[p, q] to L[1, a]
copy X[q+1, r] to R[1, b]
L[a+1] ← ∞ ; R[b+1] ← ∞
i ← 1 ; j ← 1
for k = p to r
    if L[i] ≤ R[j]
        X[k] ← L[i]
        i ← i + 1
    else
        X[k] ← R[j]
        j ← j + 1
```

$q = 4$

$p = 1$ $r = 8$

X

1	2	5	9	3	4	7	10

$k = 2$

L

1	2	5	9	∞

R

3	4	7	10	∞

$i = 2$ $j = 1$

Divide-and-Conquer
(Sorting Problem)

Mergesort

Merge(X[1 ,n], p, q, r)

```
a ← q – p + 1
b ← r – q
let L[1, a + 1] and R[1, b + 1] be new arrays
copy X[p, q] to L[1, a]
copy X[q+1, r] to R[1, b]
L[a+1] ← ∞ ; R[b+1] ← ∞
i ← 1 ; j ← 1
for k = p to r
    if L[i] ≤ R[j]
        X[k] ← L[i]
        i ← i + 1
    else
        X[k] ← R[j]
        j ← j + 1
```

p = 1 q = 4 r = 8

X:

1	2	5	9	3	4	7	10

k = 2

L:

1	2	5	9	∞

i = 2

R:

3	4	7	10	∞

j = 1

Divide-and-Conquer (Sorting Problem)

Mergesort

```
Merge(X[1,n], p, q, r)

a ← q – p + 1
b ← r – q
let L[1, a + 1] and R[1, b + 1] be new arrays
copy X[p, q] to L[1, a]
copy X[q+1, r] to R[1, b]
L[a+1] ← ∞ ; R[b+1] ← ∞
i ← 1 ; j ← 1
for k = p to r
    if L[i] ≤ R[j]
        X[k] ← L[i]
        i ← i + 1
    else
        X[k] ← R[j]
        j ← j + 1
```

$p = 1$ $q = 4$ $r = 8$

X

1	2	5	9	3	4	7	10

$k = 2$

L

1	2	5	9	∞

$i = 2$

R

3	4	7	10	∞

$j = 1$

Divide-and-Conquer
(Sorting Problem)

Mergesort

Merge(X[1 ,n], p, q, r)

```
a ← q - p + 1
b ← r - q
let L[1, a + 1] and R[1, b + 1] be new arrays
copy X[p, q] to L[1, a]
copy X[q+1, r] to R[1, b]
L[a+1] ← ∞ ; R[b+1] ← ∞
i ← 1 ; j ← 1
for k = p to r
    if L[i] ≤ R[j]
        X[k] ← L[i]
        i ← i + 1
    else
        X[k] ← R[j]
        j ← j + 1
```

q = 4

p = 1 r = 8

X:

1	2	5	9	3	4	7	10

k = 2

L:

1	2	5	9	∞

i = 3

R:

3	4	7	10	∞

j = 1

Divide-and-Conquer
(Sorting Problem)

Mergesort

Merge(X[1,n], p, q, r)

```
a ← q – p + 1
b ← r – q
let L[1, a + 1] and R[1, b + 1] be new arrays
copy X[p, q] to L[1, a]
copy X[q+1, r] to R[1, b]
L[a+1] ← ∞ ; R[b+1] ← ∞
i ← 1 ; j ← 1
for k = p to r
     if L[i] ≤ R[j]
          X[k] ← L[i]
          i ← i + 1
     else
          X[k] ← R[j]
          j ← j + 1
```

$p = 1$ $q = 4$ $r = 8$

1	2	5	9	3	4	7	10	X

$k = 3$

L	1	2	5	9	∞

3	4	7	10	∞	R

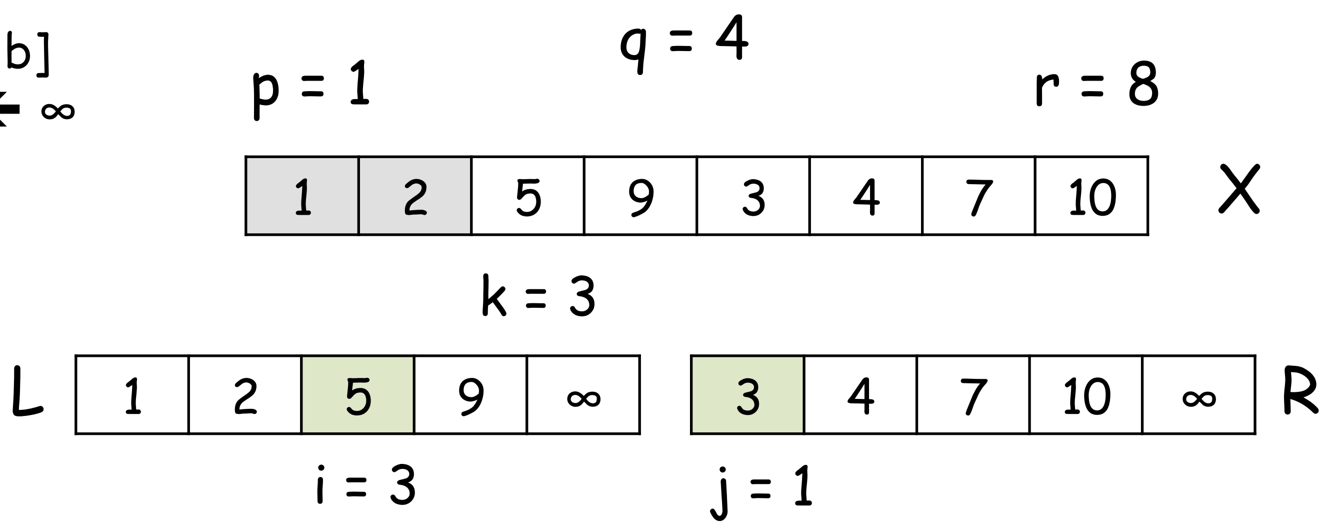
$i = 3$ $j = 1$

Divide-and-Conquer (Sorting Problem)

Mergesort

Merge(X[1 ,n], p, q, r)

```
a ← q – p + 1
b ← r – q
let L[1, a + 1] and R[1, b + 1] be new arrays
copy X[p, q] to L[1, a]
copy X[q+1, r] to R[1, b]
L[a+1] ← ∞ ; R[b+1] ← ∞
i ← 1 ; j ← 1
for k = p to r
    if L[i] ≤ R[j]
        X[k] ← L[i]
        i ← i + 1
    else
        X[k] ← R[j]
        j ← j + 1
```

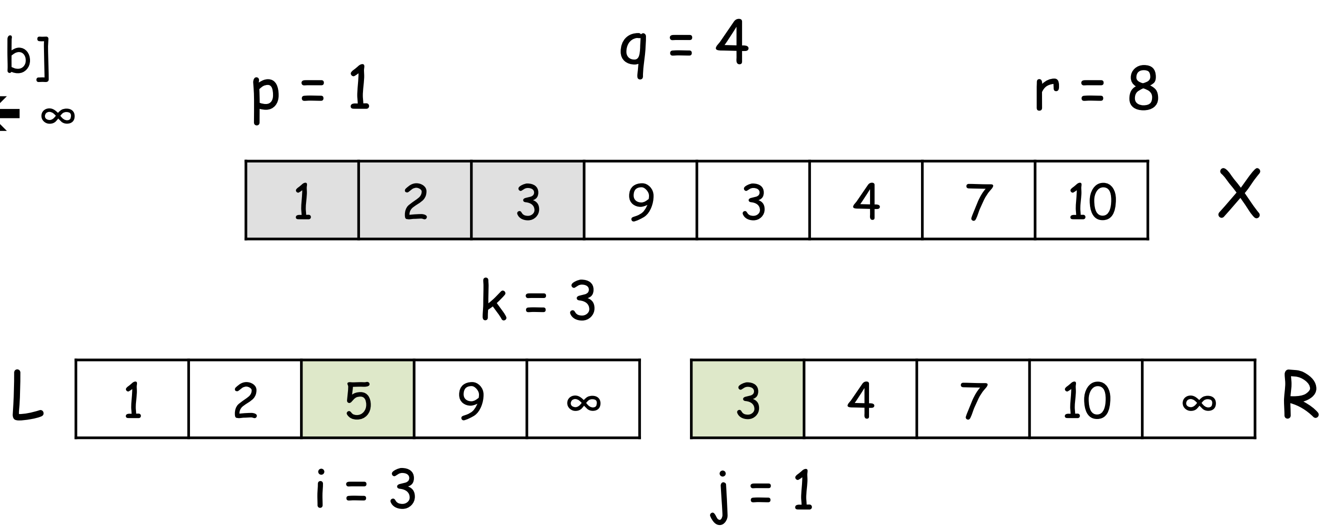


Divide-and-Conquer
(Sorting Problem)

Mergesort

```
Merge(X[1,n], p, q, r)

a ← q − p + 1
b ← r − q
let L[1, a + 1] and R[1, b + 1] be new arrays
copy X[p, q] to L[1, a]
copy X[q+1, r] to R[1, b]
L[a+1] ← ∞ ; R[b+1] ← ∞
i ← 1 ; j ← 1
for k = p to r
    if L[i] ≤ R[j]
        X[k] ← L[i]
        i ← i + 1
    else
        X[k] ← R[j]
        j ← j + 1
```



Divide-and-Conquer
(Sorting Problem)

Mergesort

Merge(X[1 ,n], p, q, r)

```
a ← q – p + 1
b ← r – q
let L[1, a + 1] and R[1, b + 1] be new arrays
copy X[p, q] to L[1, a]
copy X[q+1, r] to R[1, b]
L[a+1] ← ∞ ; R[b+1] ← ∞
i ← 1 ; j ← 1
for k = p to r
    if L[i] ≤ R[j]
        X[k] ← L[i]
        i ← i + 1
    else
        X[k] ← R[j]
        j ← j + 1
```

q = 4

p = 1 r = 8

X

1	2	3	9	3	4	7	10

k = 3

L

1	2	5	9	∞

R

3	4	7	10	∞

i = 3 j = 2

Divide-and-Conquer
(Sorting Problem)

Mergesort

```
Merge(X[1,n], p, q, r)

a ← q – p + 1
b ← r – q
let L[1, a + 1] and R[1, b + 1] be new arrays
copy X[p, q] to L[1, a]
copy X[q+1, r] to R[1, b]
L[a+1] ← ∞ ; R[b+1] ← ∞
i ← 1 ; j ← 1
for k = p to r
    if L[i] ≤ R[j]
        X[k] ← L[i]
        i ← i + 1
    else
        X[k] ← R[j]
        j ← j + 1
```

$p = 1$ $q = 4$ $r = 8$

X:

1	2	3	4	5	7	9	10

$k = 8$

L:

1	2	5	9	∞

$i = 5$

R:

3	4	7	10	∞

$j = 4$

Divide-and-Conquer
(Sorting Problem)

Mergesort

Merge(X[1 ,n], p, q, r)

```
a ← q - p + 1
b ← r - q
let L[1, a + 1] and R[1, b + 1] be new arrays
copy X[p, q] to L[1, a]
copy X[q+1, r] to R[1, b]
L[a+1] ← ∞ ; R[b+1] ← ∞
i ← 1 ; j ← 1
for k = p to r
    if L[i] ≤ R[j]
        X[k] ← L[i]
        i ← i + 1
    else
        X[k] ← R[j]
        j ← j + 1
```

$p = 1$ $q = 4$ $r = 8$

X

1	2	3	4	5	7	9	10

$k = 8$

L

1	2	5	9	∞

R

3	4	7	10	∞

$i = 5$ $j = 4$

Divide-and-Conquer
(Sorting Problem)

Mergesort

```
Merge(X[1,n], p, q, r)

a ← q – p + 1
b ← r – q
let L[1, a + 1] and R[1, b + 1] be new arrays
copy X[p, q] to L[1, a]
copy X[q+1, r] to R[1, b]
L[a+1] ← ∞ ; R[b+1] ← ∞
i ← 1 ; j ← 1
for k = p to r
    if L[i] ≤ R[j]
        X[k] ← L[i]
        i ← i + 1
    else
        X[k] ← R[j]
        j ← j + 1
```

p = 1 q = 4 r = 8

1	2	3	4	5	7	9	10	X

k = 8

L	1	2	5	9	∞

3	4	7	10	∞	R

i = 5 j = 4

Divide-and-Conquer (Sorting Problem)

Mergesort

```
Merge(X[1,n], p, q, r)

a ← q – p + 1
b ← r – q
let L[1, a + 1] and R[1, b + 1] be new arrays
copy X[p, q] to L[1, a]
copy X[q+1, r] to R[1, b]
L[a+1] ← ∞ ; R[b+1] ← ∞
i ← 1 ; j ← 1
for k = p to r
    if L[i] ≤ R[j]
        X[k] ← L[i]
        i ← i + 1
    else
        X[k] ← R[j]
        j ← j + 1
```

p = 1 q = 4 r = 8

X:

1	2	3	4	5	7	9	10

k = 8

L:

1	2	5	9	∞

i = 5

R:

3	4	7	10	∞

j = 5

Divide-and-Conquer
(Sorting Problem)

Mergesort

Merge(X[1 ,n], p, q, r)

- let n = r - p + 1

```
a ← q – p + 1
b ← r – q
let L[1, a + 1] and R[1, b + 1] be new arrays
copy X[p, q] to L[1, a]
copy X[q+1, r] to R[1, b]
L[a+1] ← ∞ ; R[b+1] ← ∞
i ← 1 ; j ← 1
for k = p to r
    if L[i] ≤ R[j]
        X[k] ← L[i]
        i ← i + 1
    else
        X[k] ← R[j]
        j ← j + 1
```

$\Theta(a)$

$\Theta(b)$

$\Theta(n)$

Divide-and-Conquer
(Sorting Problem)

Mergesort

Merge-Sort(X[1 ,n], p, r)

input: *an array of n orderable items*
output : *sorted array of n items*

```
if i < j
    q ← (i + j)/2
    Merge-Sort(X, p, q)
    Merge-Sort(X, q + 1, r)
    Merge(X, p, q, r)
```

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

$T(n) = 2.T(n/2) + f(n)$ if $n > 1$

Divide-and-Conquer
(Sorting Problem)

Mergesort

Merge-Sort(X[1 ,n], p, r)

input: *an array of n orderable items*
output : *sorted array of n items*

```
if i < j
    q ← (i + j)/2
    Merge-Sort(X, p, q)
    Merge-Sort(X, q + 1, r)
    Merge(X, p, q, r)
```

$T(n) = \theta(1)$ if $n = 1$

$T(n) = 2.T(n/2) + \theta(n)$ if $n > 1$

Divide-and-Conquer
(Sorting Problem)

Mergesort

Merge-Sort(X[1,n], p, r)

input: *an array of n orderable items*
output : *sorted array of n items*

```
if i < j
    q ← (i + j)/2
    Merge-Sort(X, p, q)
    Merge-Sort(X, q + 1, r)
    Merge(X, p, q, r)
```

$T(n) = \theta(1)$ if $n = 1$

$T(n) = 2.T(n/2) + \theta(n)$ if $n > 1$

from Master Theorem (the second case),

Divide-and-Conquer (Sorting Problem)

Mergesort

Merge-Sort(X[1,n], p, r)

input: *an array of n orderable items*
output : *sorted array of n items*

```
if i < j
    q ← (i + j)/2
    Merge-Sort(X, p, q)
    Merge-Sort(X, q + 1, r)
    Merge(X, p, q, r)
```

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

$T(n) = 2.T(n/2) + \Theta(n)$ if $n > 1$

from Master Theorem (the second case),

$f(n) = \Theta(n.\log^k n)$ for $k = 0$,

Divide-and-Conquer (Sorting Problem)

Mergesort

Merge-Sort(X[1,n], p, r)

input: *an array of n orderable items*
output : *sorted array of n items*

```
if i < j
    q ← (i + j)/2
    Merge-Sort(X, p, q)
    Merge-Sort(X, q + 1, r)
    Merge(X, p, q, r)
```

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

$T(n) = 2.T(n/2) + \Theta(n)$ if $n > 1$

from Master Theorem (the second case),

$f(n) = \Theta(n.\log^k n)$ for $k = 0$,

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

Divide-and-Conquer (Sorting Problem)

Mergesort

S	E	L	E	C	T	I	O	N

Merge-Sort(X[1 ,n], p, r)

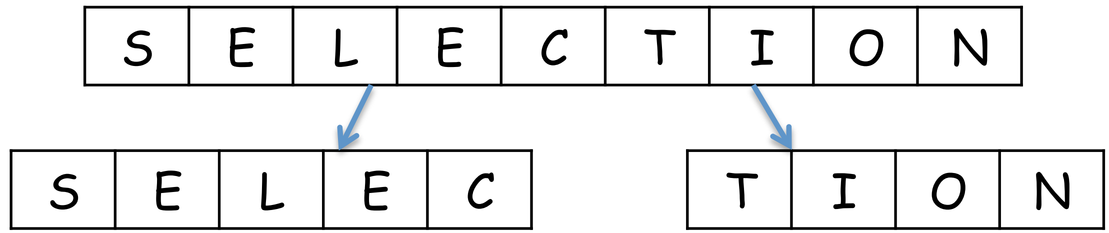
```
if i < j
    q ← (i + j)/2
    Merge-Sort(X, p, q)
    Merge-Sort(X, q + 1, r)
    Merge(X, p, q, r)
```

Divide-and-Conquer (Sorting Problem)

Mergesort

Merge-Sort(X[1,n], p, r)

```
if i < j
    q ← (i + j)/2
    Merge-Sort(X, p, q)
    Merge-Sort(X, q + 1, r)
    Merge(X, p, q, r)
```

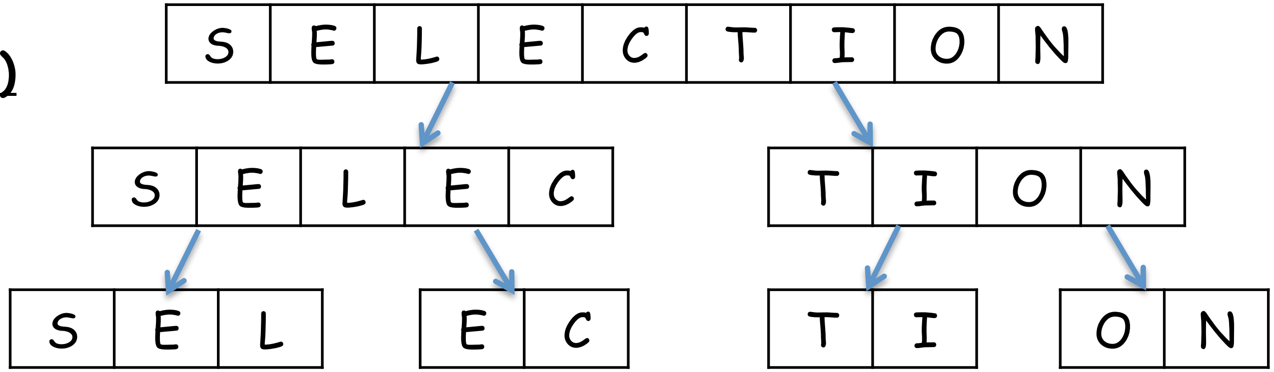


Divide-and-Conquer
(Sorting Problem)

Mergesort

Merge-Sort(X[1,n], p, r)

if i < j
    q ← (i + j)/2
    Merge-Sort(X, p, q)
    Merge-Sort(X, q + 1, r)
    Merge(X, p, q, r)

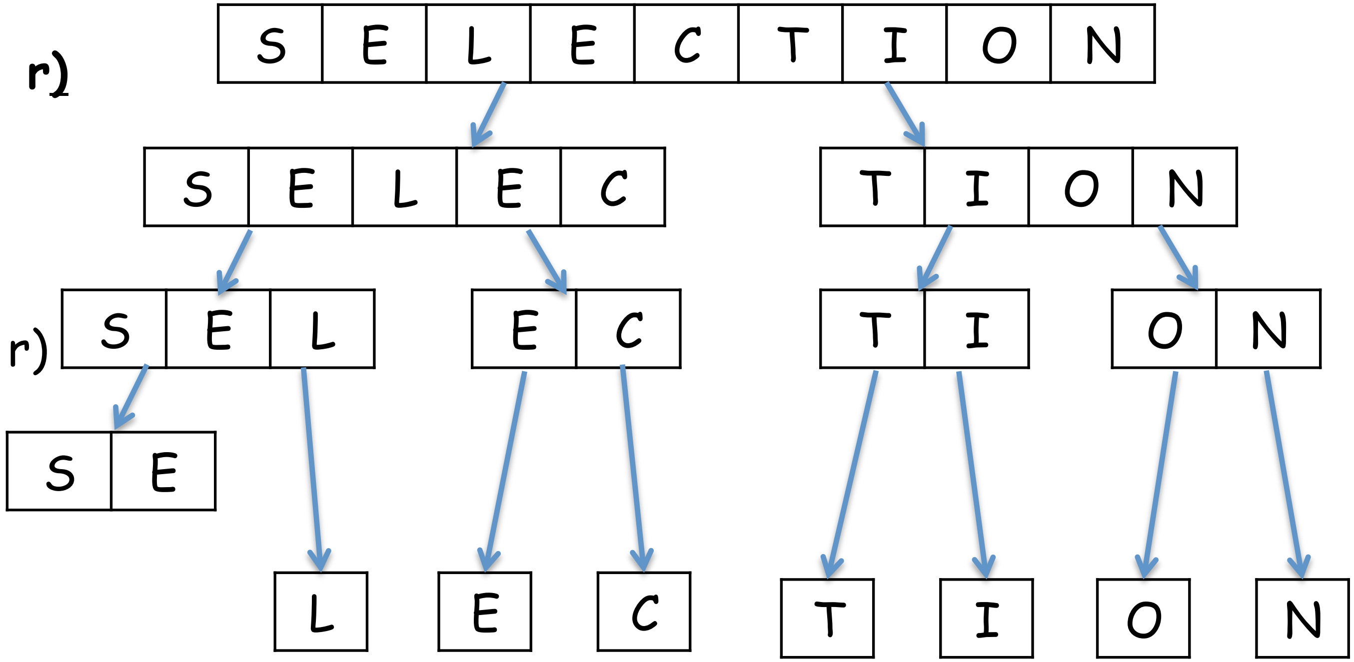


Divide-and-Conquer
(Sorting Problem)

Mergesort

Merge-Sort(X[1 ,n], p, r)

if i < j
    q ← (i + j)/2
    Merge-Sort(X, p, q)
    Merge-Sort(X, q + 1, r)
    Merge(X, p, q, r)

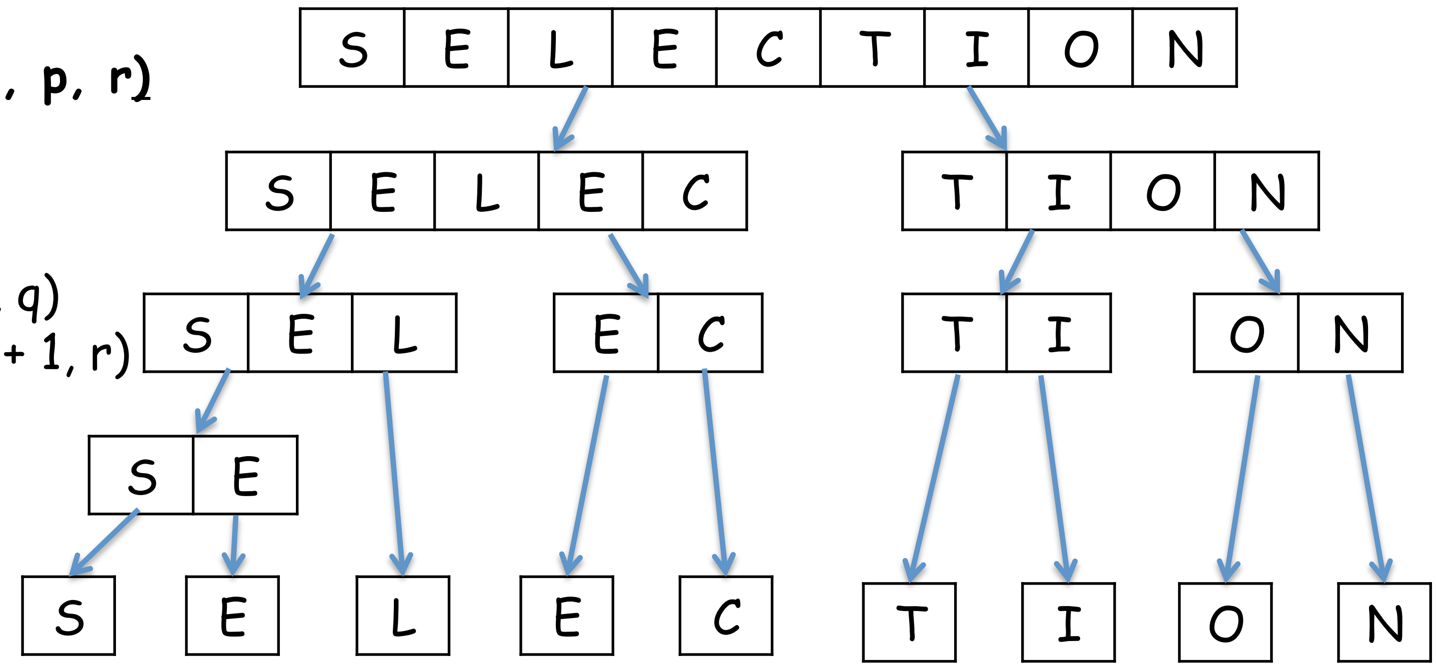


Divide-and-Conquer
(Sorting Problem)

Mergesort

Merge-Sort(X[1 ,n], p, r)

if i < j
  q ← (i + j)/2
  Merge-Sort(X, p, q)
  Merge-Sort(X, q + 1, r)
  Merge(X, p, q, r)

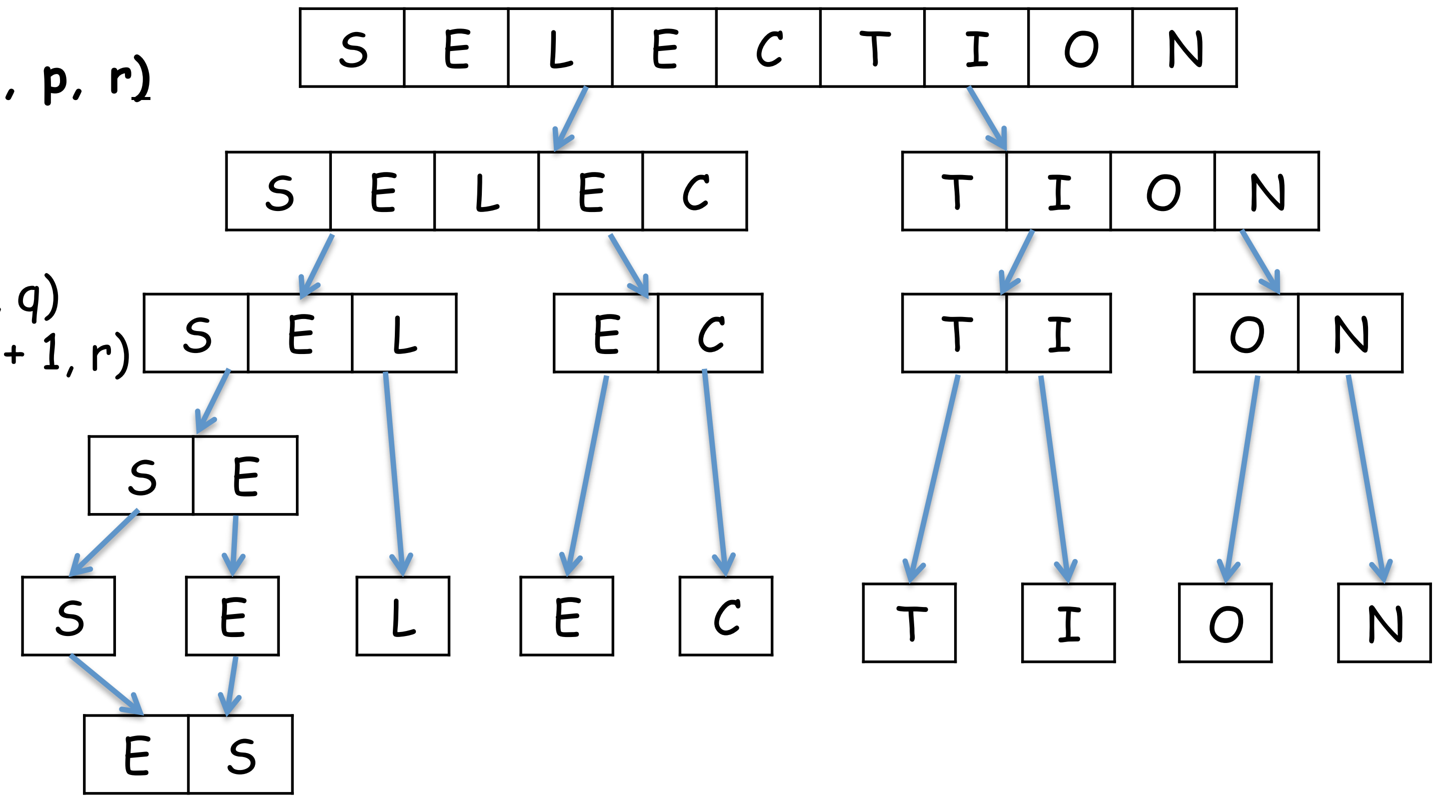


Divide-and-Conquer
(Sorting Problem)

Mergesort

Merge-Sort(X[1,n], p, r)

if i < j
    q ← (i + j)/2
    Merge-Sort(X, p, q)
    Merge-Sort(X, q + 1, r)
    Merge(X, p, q, r)

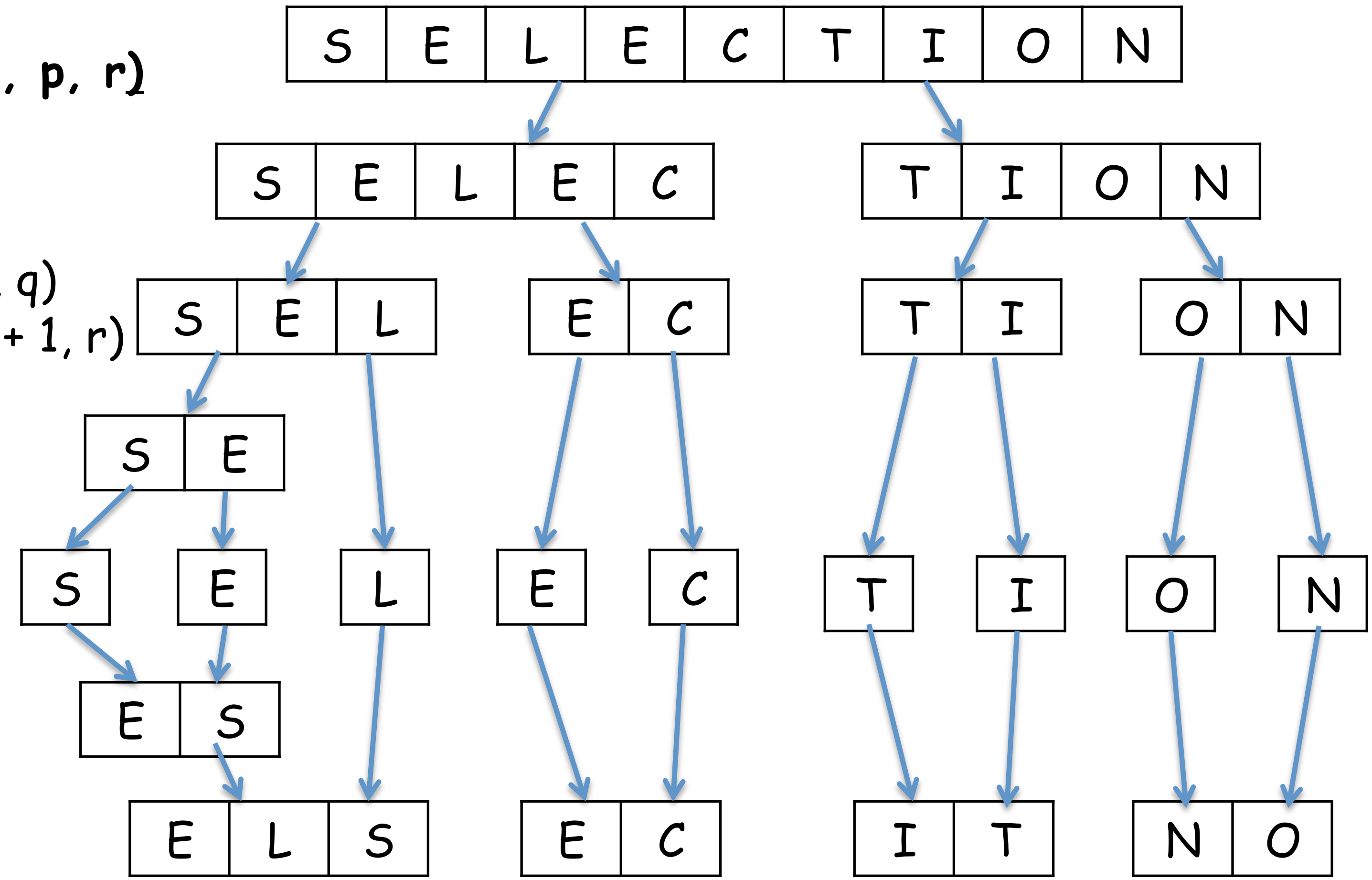


Divide-and-Conquer
(Sorting Problem)

Mergesort

Merge-Sort(X[1,n], p, r)

if i < j
    q ← (i + j)/2
    Merge-Sort(X, p, q)
    Merge-Sort(X, q + 1, r)
    Merge(X, p, q, r)

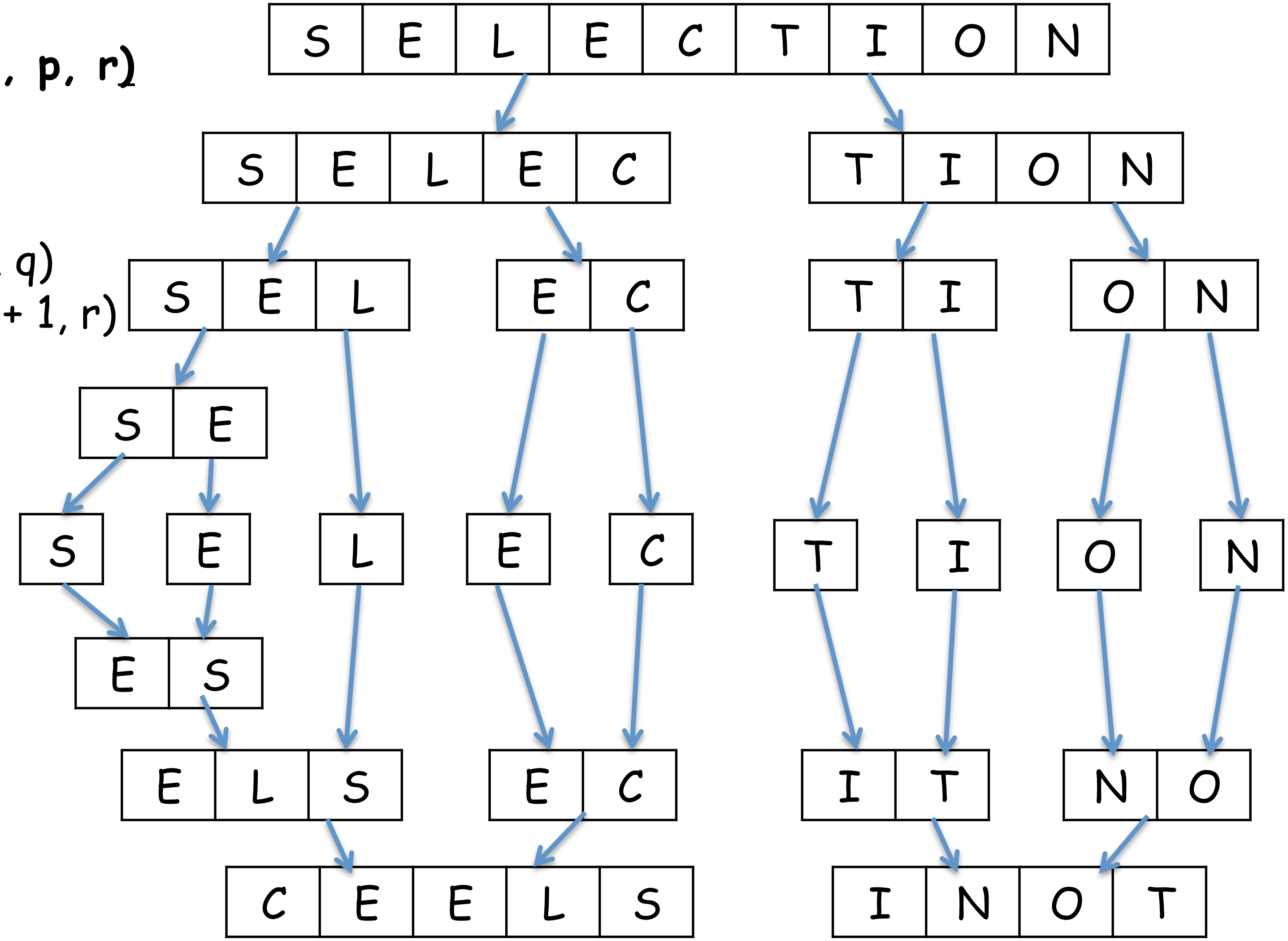


Divide-and-Conquer
(Sorting Problem)

Mergesort

Merge-Sort(X[1,n], p, r)

if i < j
    q ← (i + j)/2
    Merge-Sort(X, p, q)
    Merge-Sort(X, q + 1, r)
    Merge(X, p, q, r)

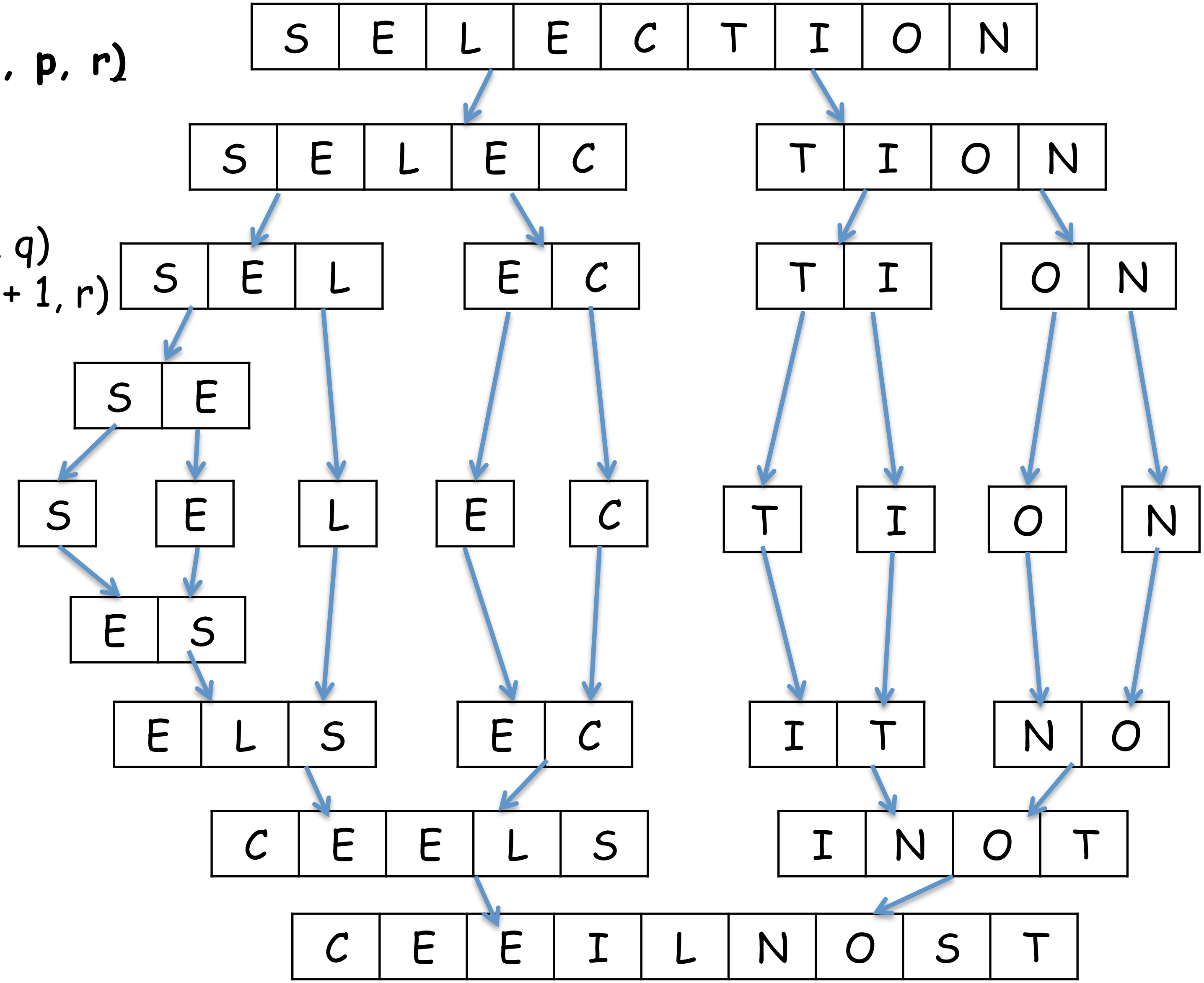


Divide-and-Conquer
(Sorting Problem)

Mergesort

Merge-Sort(X[1,n], p, r)

if i < j
    q ← (i + j)/2
    Merge-Sort(X, p, q)
    Merge-Sort(X, q + 1, r)
    Merge(X, p, q, r)



Divide-and-Conquer
(Sorting Problem)

Quicksort
(divide the elements according to their value)

- given an array of n orderable items [a_1, a_2,…, a_n], reorder the items as [a_1', a_2',…, a_n'] such that $a_1' \le a_2' \le \ldots \le a_n'$

Divide-and-Conquer (Sorting Problem)

Quicksort

(divide the elements according to their value)

- given an array of n orderable items $[a_1, a_2,\dots, a_n]$, reorder the items as $[a_1', a_2',\dots, a_n']$ such that $a_1' \le a_2' \le \dots \le a_n'$
- divide the given array into two parts such that the elements in the left part less than a certain element of the array (pivot) and the elements in the right part greater than the pivot, sort each of them recursively

Divide-and-Conquer (Sorting Problem)

Quicksort
(divide the elements according to their value)

- given an array of n orderable items $[a_1, a_2,\ldots, a_n]$, reorder the items as $[a_1', a_2',\ldots, a_n']$ such that $a_1' \leq a_2' \leq \ldots \leq a_n'$
- divide the given array into two parts such that the elements in the left part less than a certain element of the array (pivot) and the elements in the right part greater than the pivot, sort each of them recursively

  (making effort on dividing rather than merging)

Divide-and-Conquer
(Sorting Problem)

Quicksort
(divide the elements according to their value)

- given an array of n orderable items $[a_1, a_2,\ldots, a_n]$, reorder the items as $[a_1', a_2',\ldots, a_n']$ such that $a_1' \leq a_2' \leq \ldots \leq a_n'$
- divide the given array into two parts such that the elements in the left part less than a certain element of the array (pivot) and the elements in the right part greater than the pivot, sort each of them recursively

  (making effort on dividing rather than merging)

Quick-Sort(X[1,n],p,r)
input: *an array of n orderable items*
output : *sorted array of n items*

```
if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)
```

Divide-and-Conquer
(Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

```
if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)
```

Lomuto-Partition(X,p,r)

input: *an array of n orderable items*
output : *the partition of the array and new position for pivot*

k ← a_p ; s ← p
for i = p + 1 to r
    if a_i < k
        s ← s + 1 ; swap(a_s, a_i)
swap(a_p, a_s)
return s

Divide-and-Conquer (Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)

7	2	5	8	9	3

Lomuto-Partition(X,p,r)

input: *an array of n orderable items*
output : *the partition of the array and new position for pivot*

$k \leftarrow a_p$; $s \leftarrow p$
for $i = p + 1$ to r
    if $a_i < k$
        $s \leftarrow s + 1$; swap(a_s, a_i)
swap(a_p, a_s)
return s

Divide-and-Conquer (Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

```
if p < r
   s ← Partition(X,p,r)
   Quick-Sort(X,p,s-1)
   Quick-Sort(X,s+1,r)
```

Lomuto-Partition(X,p,r)

input: *an array of n orderable items*
output : *the partition of the array and new position for pivot*

for $i = p + 1$ to r

$k \leftarrow a_p$; $s \leftarrow p$
for $i = p + 1$ to r
  if $a_i < k$
    $s \leftarrow s + 1$; swap(a_s, a_i)
swap(a_p, a_s)
return s

Quick-Sort(X, 1, 6)

7	2	5	8	9	3

Divide-and-Conquer
(Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)

Quick-Sort(X, 1, 6)

7	2	5	8	9	3

Lomuto(X,1,6)

Lomuto-Partition(X,p,r)

input: *an array of n orderable items*
output : *the partition of the array and new position for pivot*

k ← a_p ; s ← p
for i = p + 1 to r
    if a_i < k
        s ← s + 1 ; swap(a_s, a_i)
swap(a_p, a_s)
return s

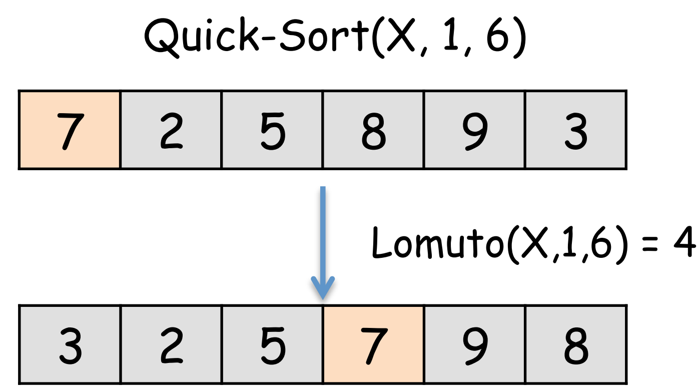
Divide-and-Conquer
(Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)



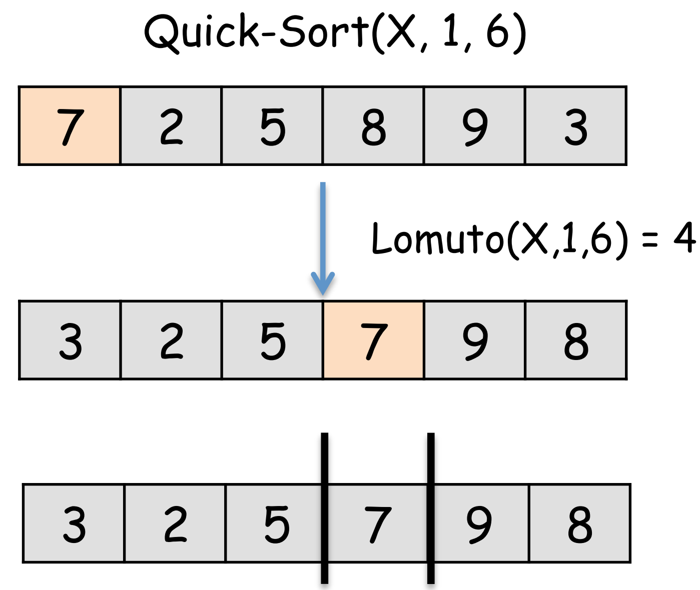
Lomuto-Partition(X,p,r)

input: *an array of n orderable items*
output : *the partition of the array and new position for pivot*

k ← a_p ; s ← p
for i = p + 1 to r
    if a_i < k
        s ← s + 1 ; swap(a_s, a_i)
swap(a_p, a_s)
return s

Divide-and-Conquer
(Sorting Problem)

Quicksort



Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

if p < r
  s ← Partition(X,p,r)
  Quick-Sort(X,p,s-1)
  Quick-Sort(X,s+1,r)

Lomuto-Partition(X,p,r)

input: *an array of n orderable items*
output : *the partition of the array and new position for pivot*

k ← a_p ; s ← p
for i = p + 1 to r
  if a_i < k
    s ← s + 1 ; swap(a_s, a_i)
swap(a_p, a_s)
return s

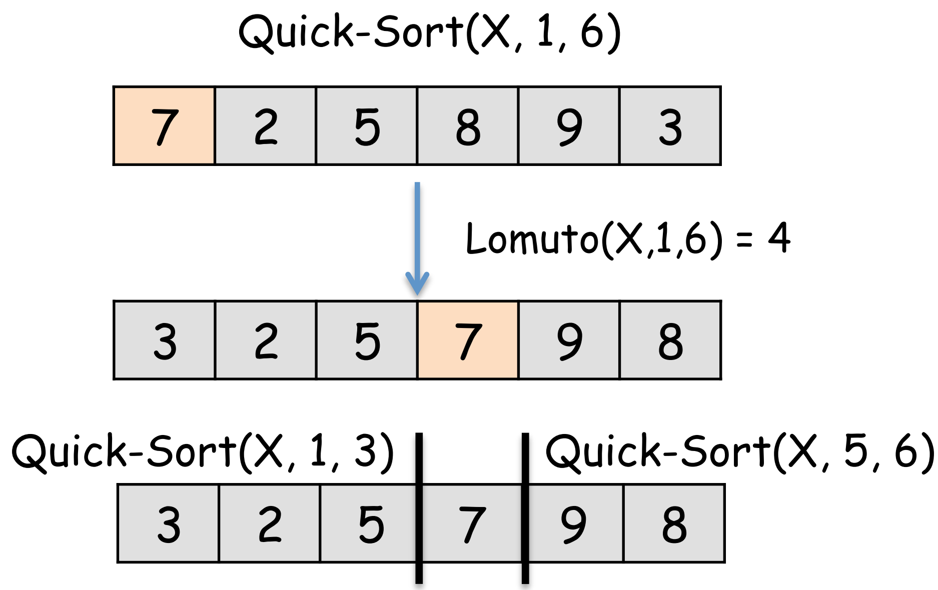
Divide-and-Conquer
(Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

```
if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)
```



Lomuto-Partition(X,p,r)

input: *an array of n orderable items*
output : *the partition of the array and new position for pivot*

k ← a_p ; s ← p
for i = p + 1 to r
    if $a_i < k$
        s ← s + 1 ; swap(a_s, a_i)
swap(a_p, a_s)
return s

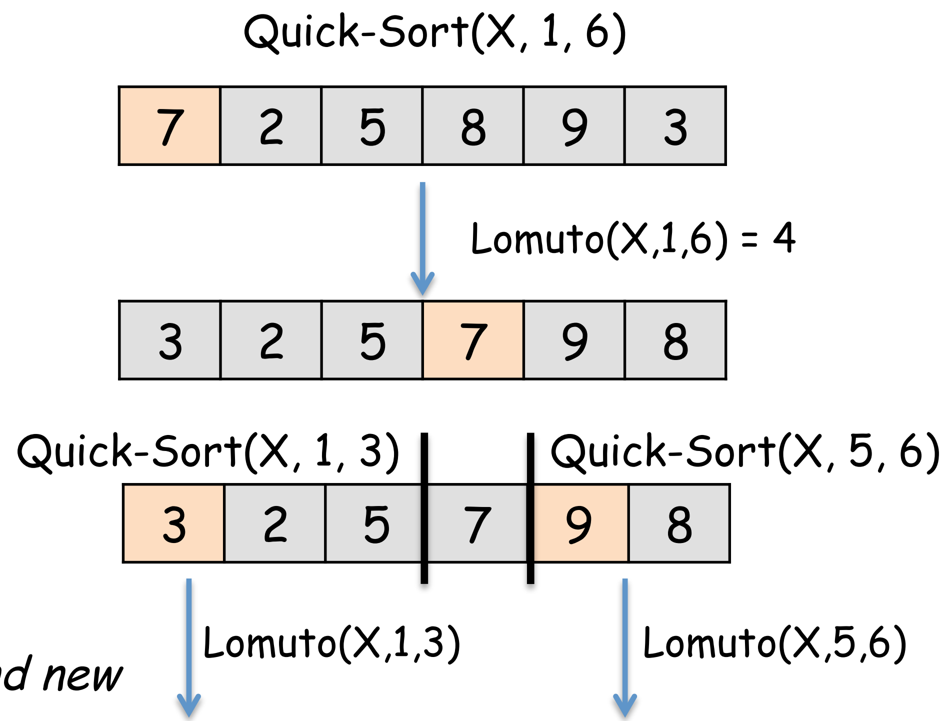
Divide-and-Conquer
(Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)



Lomuto-Partition(X,p,r)

input: *an array of n orderable items*
output : *the partition of the array and new position for pivot*

k ← a_p ; s ← p
for i = p + 1 to r
    if a_i < k
        s ← s + 1 ; swap(a_s, a_i)
swap(a_p, a_s)
return s

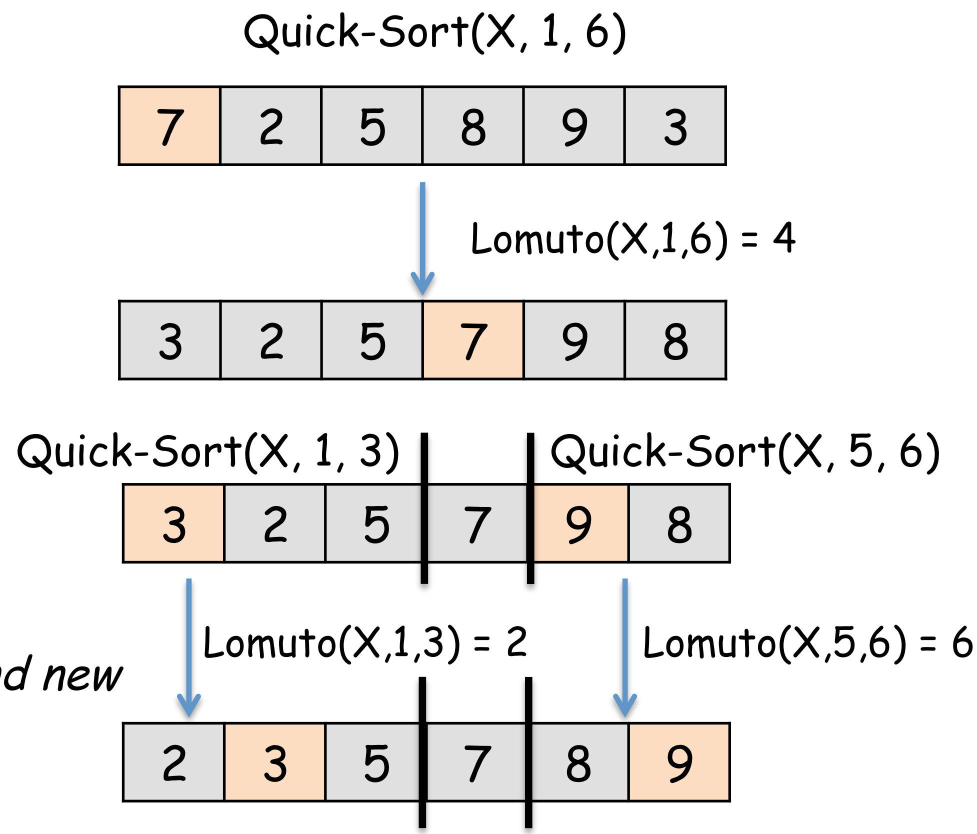
Divide-and-Conquer
(Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

```
if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)
```



Lomuto-Partition(X,p,r)

input: *an array of n orderable items*
output : *the partition of the array and new position for pivot*

```
k ← a_p ; s ← p
for i = p + 1 to r
     if a_i < k
         s ← s + 1 ; swap(a_s, a_i)
swap(a_p, a_s)
return s
```

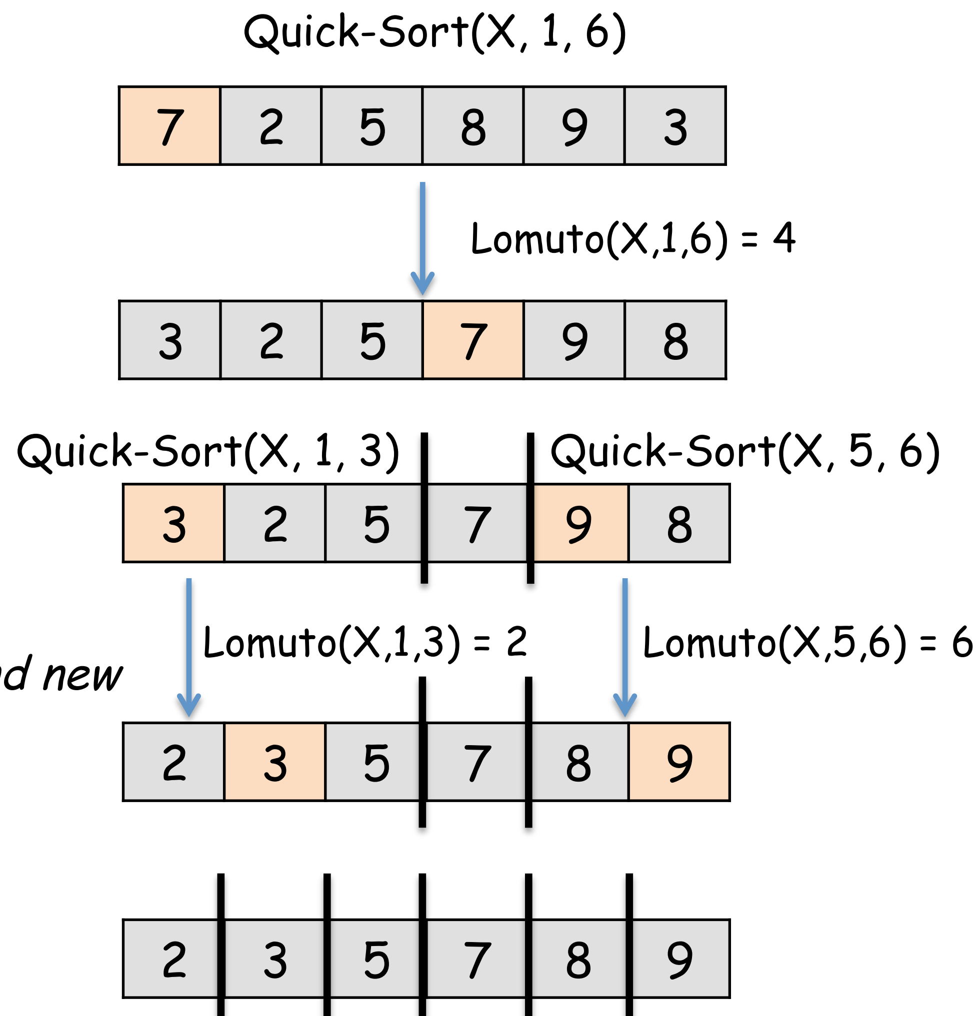
Divide-and-Conquer
(Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)



Lomuto-Partition(X,p,r)

input: *an array of n orderable items*
output : *the partition of the array and new position for pivot*

k ← a_p ; s ← p
for i = p + 1 to r
    if a_i < k
        s ← s + 1 ; swap(a_s, a_i)
swap(a_p, a_s)
return s

Divide-and-Conquer (Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

```
if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)
```

Divide-and-Conquer
(Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

```
if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)
```

- the running time of the algorithm depends on whether the partitioning is balanced or unbalanced

Divide-and-Conquer
(Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

```
if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)
```

- the running time of the algorithm depends on whether the partitioning is balanced or unbalanced
- if the partitioning is balanced, the algorithm runs asymptotically as fast as merge sort

Divide-and-Conquer (Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

```
if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)
```

- the running time of the algorithm depends on whether the partitioning is balanced or unbalanced

- if the partitioning is balanced, the algorithm runs asymptotically as fast as merge sort

  if it is unbalanced, it can run asymptotically as slowly as insertion sort

Divide-and-Conquer
(Sorting Problem)

Quicksort

Quick-Sort(X[1 ,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

```
if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)
```

Divide-and-Conquer
(Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

```
if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)
```

Worst-Case Partitioning

Divide-and-Conquer
(Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

```
if p < r
   s ← Partition(X,p,r)
   Quick-Sort(X,p,s-1)
   Quick-Sort(X,s+1,r)
```

Worst-Case Partitioning

- assume the algorithm takes a sorted array as input, i.e. at each step, partitioning procedure creates one subproblem with 0 element and one subproblem with one less element

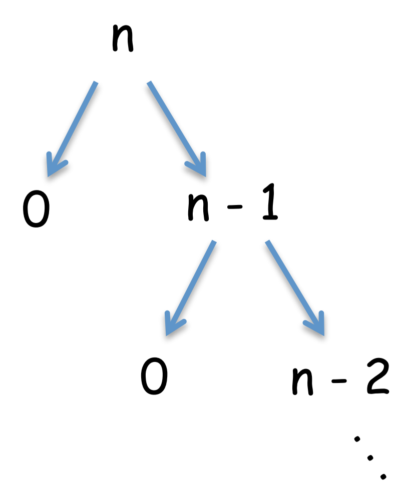
Divide-and-Conquer
(Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

```
if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)
```



Worst-Case Partitioning

- assume the algorithm takes a sorted array as input, i.e. at each step, partitioning procedure creates one subproblem with 0 element and one subproblem with one less element

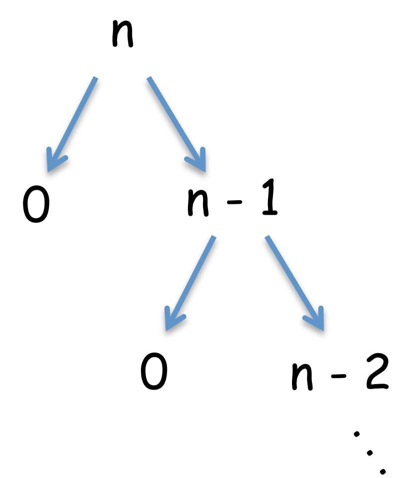
Divide-and-Conquer
(Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

```
if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)
```



Worst-Case Partitioning

- assume the algorithm takes a sorted array as input, i.e. at each step, partitioning procedure creates one subproblem with 0 element and one subproblem with one less element

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

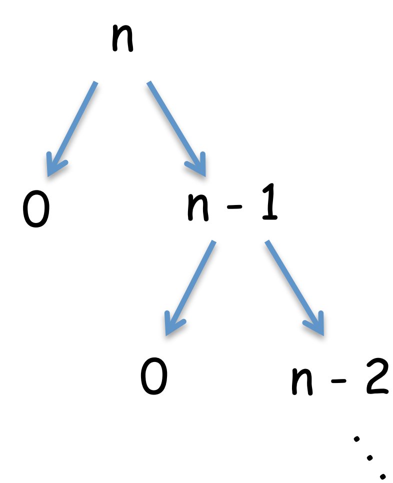
Divide-and-Conquer
(Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

```
if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)
```



Worst-Case Partitioning

- assume the algorithm takes a sorted array as input, i.e. at each step, partitioning procedure creates one subproblem with 0 element and one subproblem with one less element

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

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

Divide-and-Conquer
(Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

```
if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)
```

Balance Partitioning

Divide-and-Conquer
(Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

```
if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)
```

Balance Partitioning

- at each step, partitioning procedure creates one subproblem with n/10 – 1 elements and one subproblem with 9n/10 elements

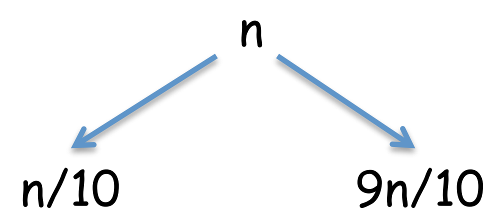
Divide-and-Conquer
(Sorting Problem)

Quicksort

<u>**Quick-Sort(X[1,n],p,r)**</u>

input: *an array of n orderable items*
output : *sorted array of n items*

```
if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)
```



<u>Balance Partitioning</u>

- at each step, partitioning procedure creates one subproblem with n/10 – 1 elements and one subproblem with 9n/10 elements

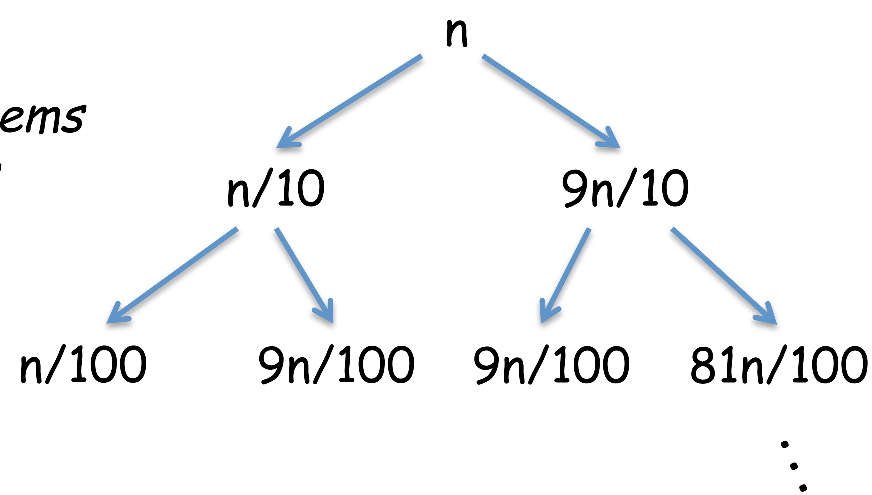
Divide-and-Conquer
(Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)



Balance Partitioning

- at each step, partitioning procedure creates one subproblem with n/10 – 1 elements and one subproblem with 9n/10 elements

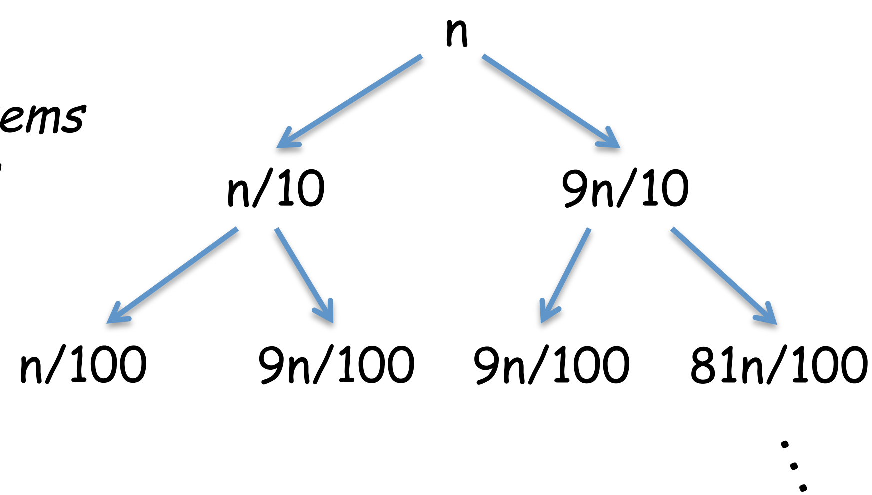
Divide-and-Conquer
(Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

```
if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)
```



Balance Partitioning

- at each step, partitioning procedure creates one subproblem with $n/10 - 1$ elements and one subproblem with $9n/10$ elements

$$T(n) = T(n/10) + T(9n/10) + O(n)$$

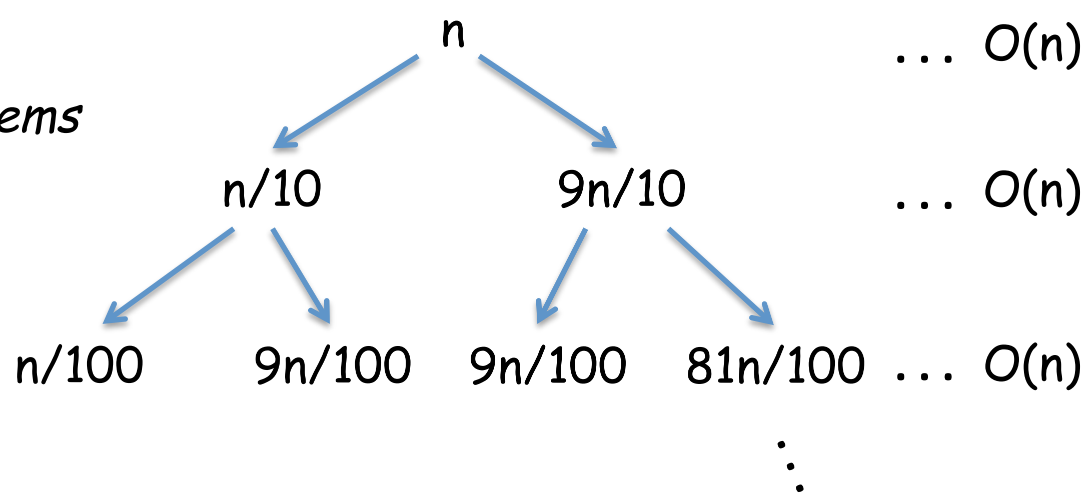
Divide-and-Conquer
(Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

```
if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)
```



Balance Partitioning

- at each step, partitioning procedure creates one subproblem with $n/10 - 1$ elements and one subproblem with $9n/10$ elements

$$T(n) = T(n/10) + T(9n/10) + O(n)$$

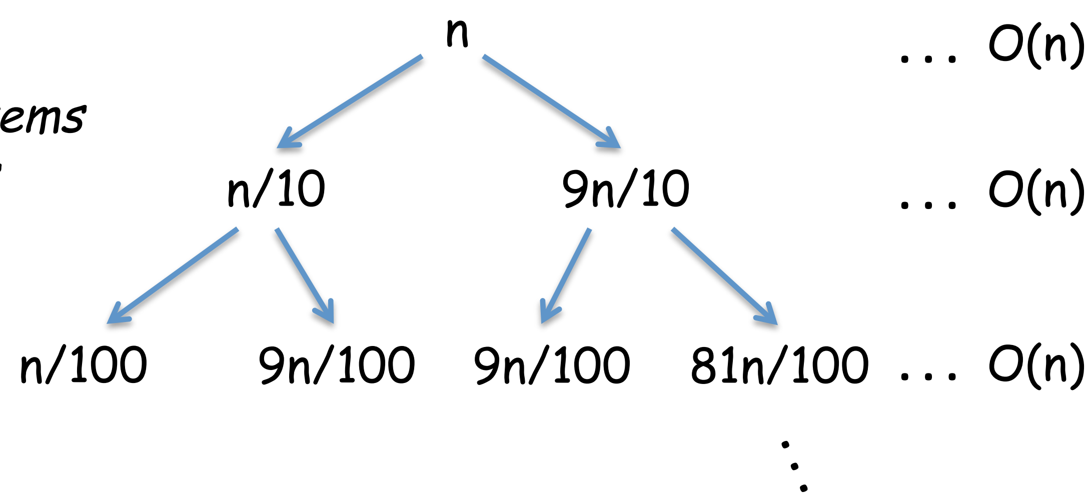
Divide-and-Conquer
(Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

```
if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)
```



Balance Partitioning

- at each step, partitioning procedure creates one subproblem with $n/10 - 1$ elements and one subproblem with $9n/10$ elements

$T(n) = T(n/10) + T(9n/10) + O(n)$

- $(9/10)^i.n = 1$

  $i = \log_{10/9} n$

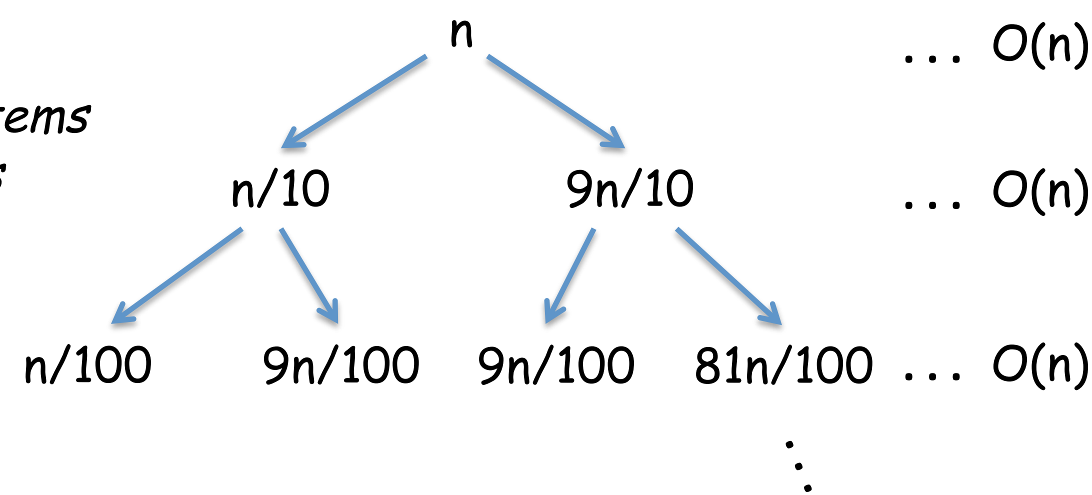
Divide-and-Conquer
(Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

```
if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)
```



Balance Partitioning

- at each step, partitioning procedure creates one subproblem with n/10 – 1 elements and one subproblem with 9n/10 elements

$T(n) = T(n/10) + T(9n/10) + O(n)$

$T(n) = O(n\log_{10/9}n)$

- $(9/10)^i.n = 1$

  $i = \log_{10/9} n$

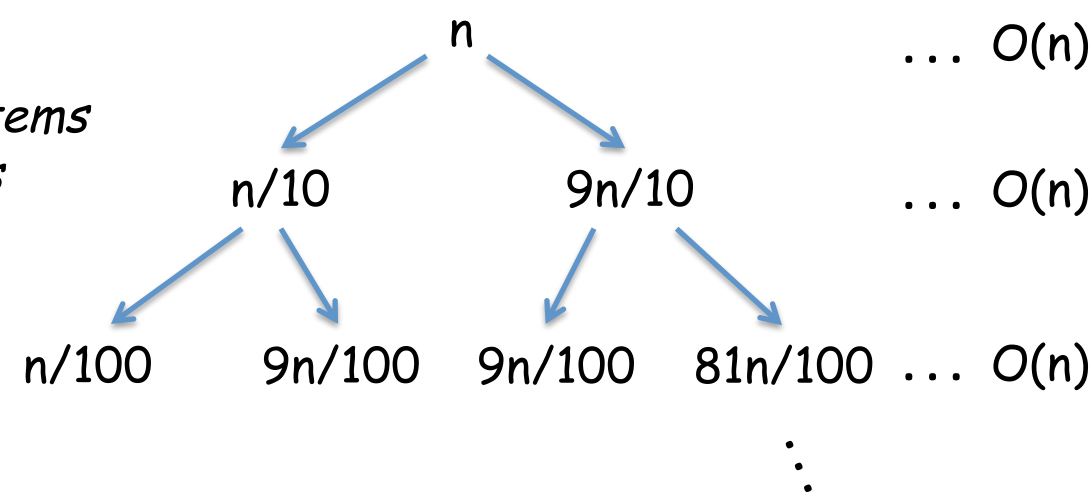
Divide-and-Conquer
(Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

```
if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)
```



Balance Partitioning

- at each step, partitioning procedure creates one subproblem with n/10 – 1 elements and one subproblem with 9n/10 elements

$T(n) = T(n/10) + T(9n/10) + O(n)$

$T(n) = O(n\log_{10/9}n) \approx O(n\log n)$ $(\log_{10/9}n > \log n)$

- $(9/10)^i.n = 1$

  $i = \log_{10/9} n$

Divide-and-Conquer
(Sorting Problem)

Quicksort

Quick-Sort(X[1 ,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

```
if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)
```

Best-Case Partitioning

Divide-and-Conquer
(Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

```
if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)
```

Best-Case Partitioning

- at each step, partitioning procedure creates one subproblem with $n/2 - 1$ elements and one subproblem with $n/2$ elements

Divide-and-Conquer
(Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)

n

n/2 - 1   n/2

Best-Case Partitioning

- at each step, partitioning procedure creates one subproblem with n/2 – 1 elements and one subproblem with n/2 elements

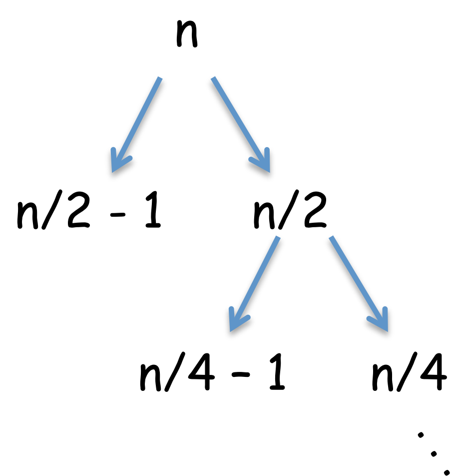
Divide-and-Conquer
(Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

```
if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)
```



Best-Case Partitioning

- at each step, partitioning procedure creates one subproblem with n/2 – 1 elements and one subproblem with n/2 elements

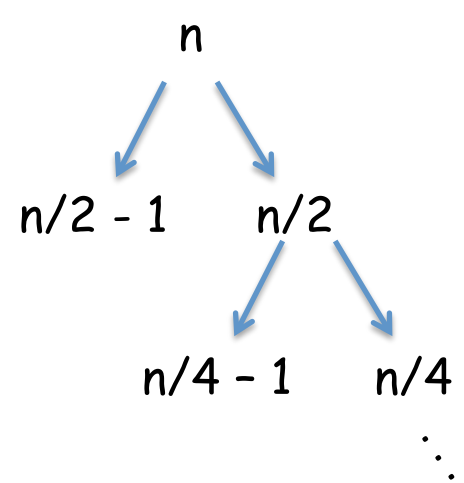
Divide-and-Conquer
(Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

```
if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)
```



Best-Case Partitioning

- at each step, partitioning procedure creates one subproblem with $n/2 - 1$ elements and one subproblem with $n/2$ elements

$$T(n) = 2T(n/2) + O(n)$$

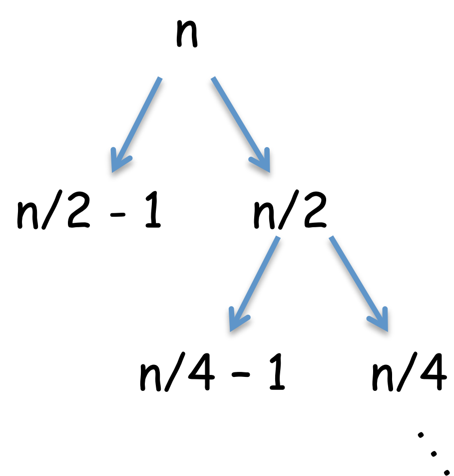
Divide-and-Conquer
(Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

```
if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)
```



Best-Case Partitioning

- at each step, partitioning procedure creates one subproblem with $n/2 - 1$ elements and one subproblem with $n/2$ elements

$$T(n) = 2T(n/2) + O(n)$$

$$T(n) = O(n\log n)$$

Divide-and-Conquer
(Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

```
if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)
```

Divide-and-Conquer
(Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)

Average-Case Partitioning

Divide-and-Conquer
(Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)

Average-Case Partitioning

- assume you run Quicksort on a random input, the partitioning is highly unlikely to happen in the same way at every level

  i.e. some splits will be well balanced (having constant proportionality), some will be unbalanced

Divide-and-Conquer (Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

```
if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)
```

Average-Case Partitioning

- assume you run Quicksort on a random input, the partitioning is highly unlikely to happen in the same way at every level

  i.e. some splits will be well balanced (having constant proportionality), some will be unbalanced

- in the corresponding recursion tree, the good and bad splits distributed randomly

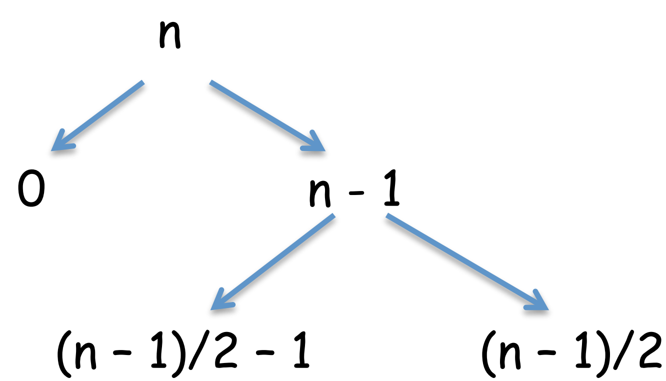
Divide-and-Conquer
(Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

```
if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)
```



Average-Case Partitioning

- assume you run Quicksort on a random input, the partitioning is highly unlikely to happen in the same way at every level

  i.e. some splits will be well balanced (having constant proportionality), some will be unbalanced

- in the corresponding recursion tree, the good and bad splits distributed randomly

  assume a bad split followed by a good split, and a good split followed by a bad one

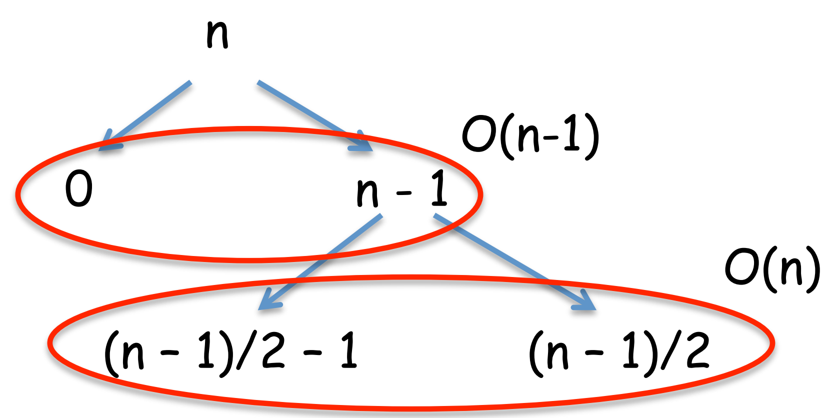
Divide-and-Conquer
(Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

```
if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)
```



Average-Case Partitioning

- assume you run Quicksort on a random input, the partitioning is highly unlikely to happen in the same way at every level

  i.e. some splits will be well balanced (having constant proportionality), some will be unbalanced

- in the corresponding recursion tree, the good and bad splits distributed randomly

  assume a bad split followed by a good split, and a good split followed by a bad one

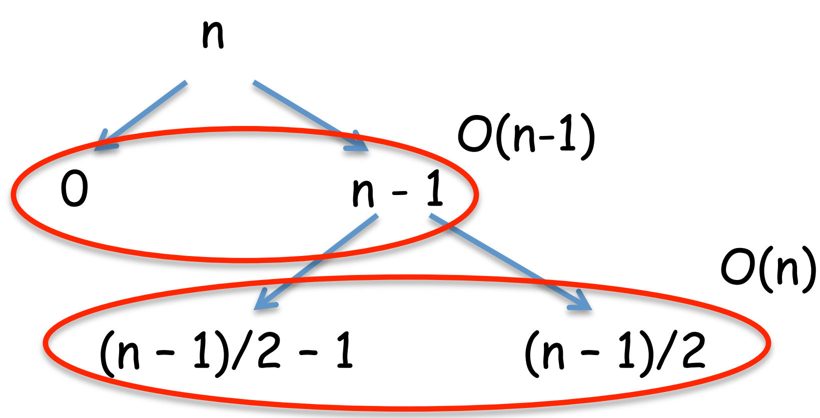
Divide-and-Conquer
(Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

```
if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)
```



Average-Case Parti

- combined partitioning cost will be O(n) + O(n-1) = O(n)

- assume you run Quicksort on a random input, the partitioning is highly unlikely to happen in the same way at every level

  i.e. some splits will be well balanced (having constant proportionality), some will be unbalanced

- in the corresponding recursion tree, the good and bad splits distributed randomly

  assume a bad split followed by a good split, and a good split followed by a bad one

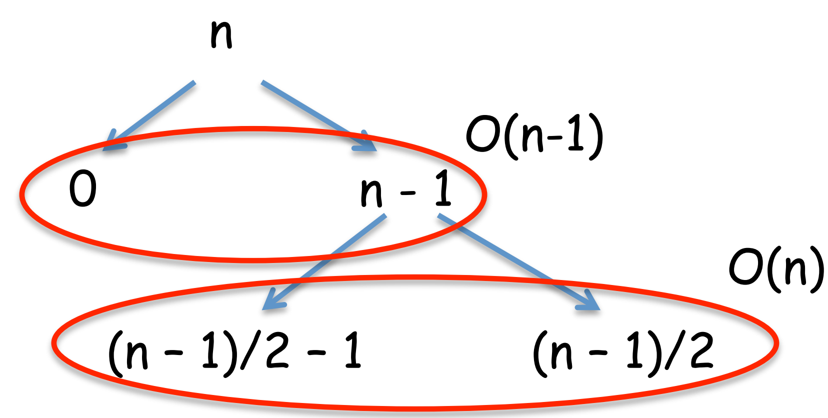
Divide-and-Conquer
(Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

if p < r
  s ← Partition(X,p,r)
  Quick-Sort(X,p,s-1)
  Quick-Sort(X,s+1,r)



Average-Case Parti

- assume you run
  unlikely to happe

  i.e. some splits w
  will be unbalance

- combined partitioning cost will be O(n) + O(n-1) = O(n)
- O(n) cost of bad split can be absorbed into O(n) cost of good split

- in the corresponding recursion tree, the good and bad splits distributed randomly

  assume a bad split followed by a good split, and a good split followed by a bad one

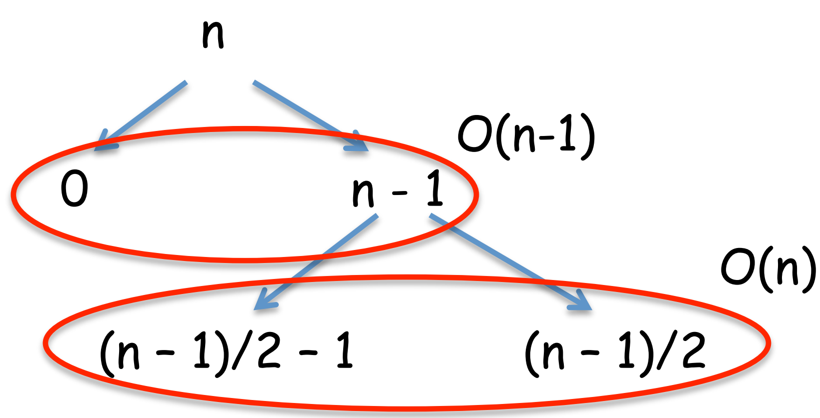
Divide-and-Conquer
(Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

```
if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)
```



Average-Case Parti

- assume you run
  unlikely to happe

  i.e. some splits w
  will be unbalance

- in the correspon
  randomly

  assume a bad split followed by a good split, and a good split followed by a bad one

- combined partitioning cost will be $O(n) + O(n-1) = O(n)$
- $O(n)$ cost of bad split can be absorbed into $O(n)$ cost of good split
- thus, a bad split and a following good split yield a good split

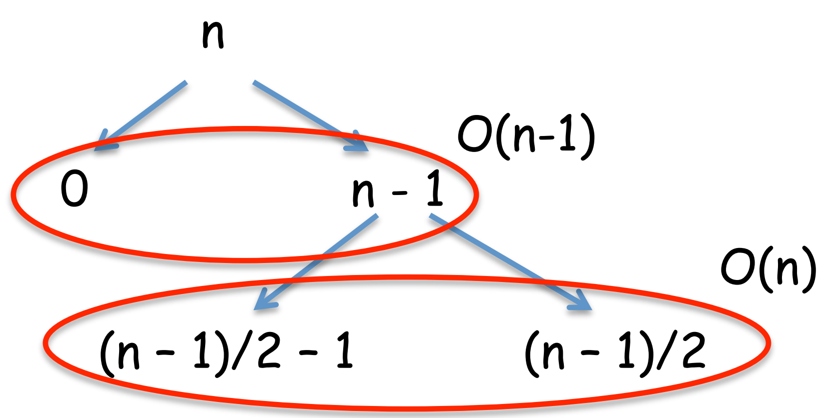
Divide-and-Conquer
(Sorting Problem)

Quicksort

Quick-Sort(X[1,n],p,r)

input: *an array of n orderable items*
output : *sorted array of n items*

if p < r
    s ← Partition(X,p,r)
    Quick-Sort(X,p,s-1)
    Quick-Sort(X,s+1,r)



Average-Case Parti

- assume you run
unlikely to happe

i.e. some splits w
will be unbalance

- in the correspon
randomly

assume a bad split follow   a
bad one

- combined partitioning cost will be $O(n) + O(n\text{-}1) = O(n)$
- $O(n)$ cost of bad split can be absorbed into $O(n)$ cost of good split
- thus, a bad split and a following good split yield a good split

$T_{avg}(n) \approx 1.39n\log n$ (check pg 180)

Divide-and-Conquer

Matrix Multiplication

Divide-and-Conquer

Matrix Multiplication

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$$

Divide-and-Conquer

Matrix Multiplication

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$$

- $c_{11} = a_{11} \times b_{11} + a_{12} \times b_{21}$, $c_{12} = a_{11} \times b_{12} + a_{12} \times b_{22}$

  $c_{21} = a_{21} \times b_{11} + a_{21} \times b_{21}$, $c_{22} = 2_{11} \times b_{12} + a_{22} \times b_{22}$

Divide-and-Conquer

Matrix Multiplication

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$$

- $c_{11} = a_{11} \times b_{11} + a_{12} \times b_{21}$, $c_{12} = a_{11} \times b_{12} + a_{12} \times b_{22}$

  $c_{21} = a_{21} \times b_{11} + a_{21} \times b_{21}$, $c_{22} = 2_{11} \times b_{12} + a_{22} \times b_{22}$

- thus, brute-force algorithm applies 8 multiplications and four additions

Divide-and-Conquer

Matrix Multiplication

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$$

- $c_{11} = a_{11} \times b_{11} + a_{12} \times b_{21}$, $c_{12} = a_{11} \times b_{12} + a_{12} \times b_{22}$

  $c_{21} = a_{21} \times b_{11} + a_{21} \times b_{21}$, $c_{22} = 2_{11} \times b_{12} + a_{22} \times b_{22}$

- thus, brute-force algorithm applies 8 multiplications and four additions

- Strassen (1969) developed an algorithm that applies 7 multiplications and 18 addition/subtractions

Divide-and-Conquer

Matrix Multiplication
(Strassen's Algorithm)

Divide-and-Conquer

Matrix Multiplication (Strassen's Algorithm)

- The algorithm employs the following formula

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$$

Divide-and-Conquer

Matrix Multiplication
(Strassen's Algorithm)

- The algorithm employs the following formula

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} m_1+m_4-m_5+m_7 & m_3+m_5 \\ m_2+m_4 & m_1+m_3-m_2+m_6 \end{pmatrix}$$

Divide-and-Conquer

Matrix Multiplication (Strassen's Algorithm)

- The algorithm employs the following formula

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} m_1+m_4-m_5+m_7 & m_3+m_5 \\ m_2+m_4 & m_1+m_3-m_2+m_6 \end{pmatrix}$$

where $m_1 = (a_{11} + a_{22}) \times (b_{11} + b_{22})$, $m_2 = (a_{21} + a_{22}) \times b_{11}$,

$m_3 = a_{11} \times (b_{12} - b_{22})$, $m_4 = a_{22} \times (b_{21} - b_{11})$,

$m_5 = (a_{11} + a_{12}) \times b_{22}$, $m_6 = (a_{21} - a_{11}) \times (b_{11} + b_{12})$,

$m_7 = (a_{12} - a_{22}) \times (b_{12} + b_{22})$

Divide-and-Conquer

Matrix Multiplication
(Strassen's Algorithm)

- let A and B be two nxn matrices where n is a power of 2

Divide-and-Conquer

Matrix Multiplication (Strassen's Algorithm)

- let A and B be two nxn matrices where n is a power of 2

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} & \\ & \end{pmatrix}$$

Divide-and-Conquer

Matrix Multiplication
(Strassen's Algorithm)

- let A and B be two nxn matrices where n is a power of 2
- divide A and B into four n/2 x n/2 sub-matrices

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} \quad \end{pmatrix}$$

Divide-and-Conquer

Matrix Multiplication (Strassen's Algorithm)

- let A and B be two nxn matrices where n is a power of 2
- divide A and B into four n/2 x n/2 sub-matrices

$$\left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right)\left(\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right) = \left(\qquad\qquad\qquad\right)$$

Divide-and-Conquer

Matrix Multiplication (Strassen's Algorithm)

- let A and B be two nxn matrices where n is a power of 2
- divide A and B into four n/2 x n/2 sub-matrices

$$\left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right)\left(\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right) = \begin{pmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 + M_3 - M_2 + M_6 \end{pmatrix}$$

Divide-and-Conquer

Matrix Multiplication (Strassen's Algorithm)

- let A and B be two nxn matrices where n is a power of 2
- divide A and B into four n/2 x n/2 sub-matrices

$$\left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right)\left(\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right) = \begin{pmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 + M_3 - M_2 + M_6 \end{pmatrix}$$

where $M_1 = (A_{11} + A_{22}) \times (B_{11} + B_{22})$, $M_2 = (A_{21} + A_{22}) \times B_{11}$,

$M_3 = A_{11} \times (B_{12} - B_{22})$, $M_4 = A_{22} \times (B_{21} - B_{11})$,

$M_5 = (A_{11} + A_{12}) \times B_{22}$, $M_6 = (A_{21} - A_{11}) \times (B_{11} + B_{12})$,

$M_7 = (A_{12} - A_{22}) \times (B_{12} + B_{22})$

Divide-and-Conquer

Matrix Multiplication (Strassen's Algorithm)

- let A and B be two nxn matrices where n is a power of 2
- divide A and B into four n/2 x n/2 sub-matrices

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 + M_3 - M_2 + M_6 \end{pmatrix}$$

where

- Strassen's algorithm applies 7 multiplication and 18 additions and subtractions on the sub-matrices

$M_3 = A_{11} \times (B_{12} - B_{22})$, $M_4 = A_{22} \times (B_{21} - B_{11})$,

$M_5 = (A_{11} + A_{12}) \times B_{22}$, $M_6 = (A_{21} - A_{11}) \times (B_{11} + B_{12})$,

$M_7 = (A_{12} - A_{22}) \times (B_{12} + B_{22})$

Divide-and-Conquer

Matrix Multiplication (Strassen's Algorithm)

- let A and B be two nxn matrices where n is a power of 2
- divide A and B into four n/2 x n/2 sub-matrices

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 + M_3 - M_2 + M_6 \end{pmatrix}$$

where

- Strassen's algorithm applies 7 multiplication and 18 additions and subtractions on the sub-matrices
- $T(n) = 7T(n/2) + f(n)$

$$M_5 = (A_{11} + A_{12}) \times B_{22}, \quad M_6 = (A_{21} - A_{11}) \times (B_{11} + B_{12}),$$

$$M_7 = (A_{12} - A_{22}) \times (B_{12} + B_{22})$$

Divide-and-Conquer

Matrix Multiplication (Strassen's Algorithm)

- let A and B be two nxn matrices where n is a power of 2
- divide A and B into four n/2 x n/2 sub-matrices

$$\left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right)\left(\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right) = \begin{pmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 + M_3 - M_2 + M_6 \end{pmatrix}$$

where

- Strassen's algorithm applies 7 multiplication and 18 additions and subtractions on the sub-matrices
- $T(n) = 7T(n/2) + f(n)$
- each sub-matrix contains $(n/4)^2$ entries, thus $f(n) = O(n^2)$

$M_5 = (A_{11} + A_{12}) \times B_{22}$, $M_6 = (A_{21} - A_{11}) \times (B_{11} + B_{12})$,

$M_7 = (A_{12} - A_{22}) \times (B_{12} + B_{22})$

Divide-and-Conquer

Matrix Multiplication (Strassen's Algorithm)

- let A and B be two nxn matrices where n is a power of 2
- divide A and B into four n/2 x n/2 sub-matrices

$$\left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right) \left(\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right) = \begin{pmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 + M_3 - M_2 + M_6 \end{pmatrix}$$

where

- Strassen's algorithm applies 7 multiplication and 18 additions and subtractions on the sub-matrices
- $T(n) = 7T(n/2) + f(n)$
- each sub-matrix contains $(n/4)^2$ entries, thus $f(n) = O(n^2)$
- from Master Theorem (the first case),

$$M_7 = (A_{12} - A_{22}) \times (B_{12} + B_{22})$$

Divide-and-Conquer

Matrix Multiplication (Strassen's Algorithm)

- let A and B be two nxn matrices where n is a power of 2
- divide A and B into four n/2 x n/2 sub-matrices

$$\left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right)\left(\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right) = \begin{pmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 + M_3 - M_2 + M_6 \end{pmatrix}$$

where

- Strassen's algorithm applies 7 multiplication and 18 additions and subtractions on the sub-matrices
- $T(n) = 7T(n/2) + f(n)$
- each sub-matrix contains $(n/4)^2$ entries, thus $f(n) = O(n^2)$
- from Master Theorem (the first case),

$$T(n) = \Theta(n^{log_2 7}) \approx \Theta(n^{2.8})$$