Divide and Conquer-1

Department of Computer Science, Tsinghua University

Review

- Incremental Approach
  - Solving the problem by incrementally growing the solution.
- Correctness
  - Setup loop invariant (mathematical induction).
- Efficiency
  - Determined by loops, easy to be optimized by compilers.
- Design
  - Pre-condition, Post-condition
  - Loop invariant inspired by post-condition can help your algorithm design as well.

Design Paradigm

Pro.:

- D&C is another powerful technique for algorithm design.
- D&C algorithms are ***recursive*** in nature, and can be analyzed using recurrences and the master method.

Paradigm:

1. Divide the problem (instance) into smaller problems of the ***same*** problem.

2. Conquer the subproblems by solving them ***recursively*** and ***solve them directly*** if they are small enough.

3. Combine subproblem solutions to solve the problem.

Design Paradigm

Paradigm:

1. Divide the problem (instance) into subproblems.

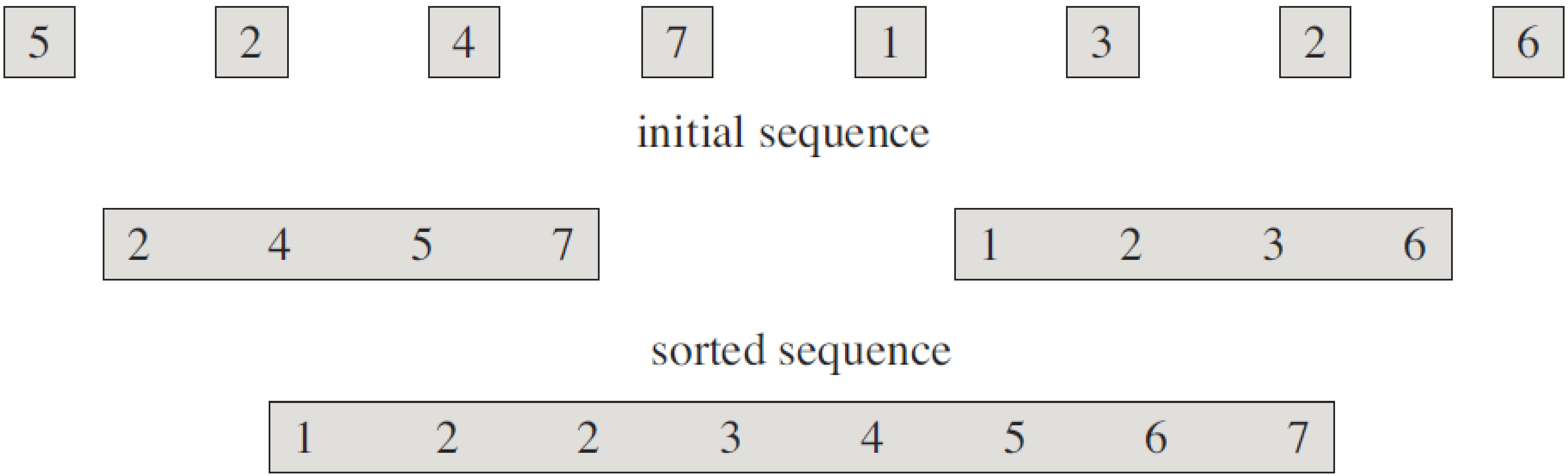
2. Conquer the subproblems by solving them recursively.

3. Combine subproblem solutions to solve the problem.

DC(P)
if P is not small enough
    Divide P into subproblems P_1, P_2, …, P_m;
    S_1 = DC(P_1); S_2 = DC(P_2);…;S_m = DC(P_m);
    Combine solutions S_1, S_2, … S_m to S;
else S=solve(P);
return S;

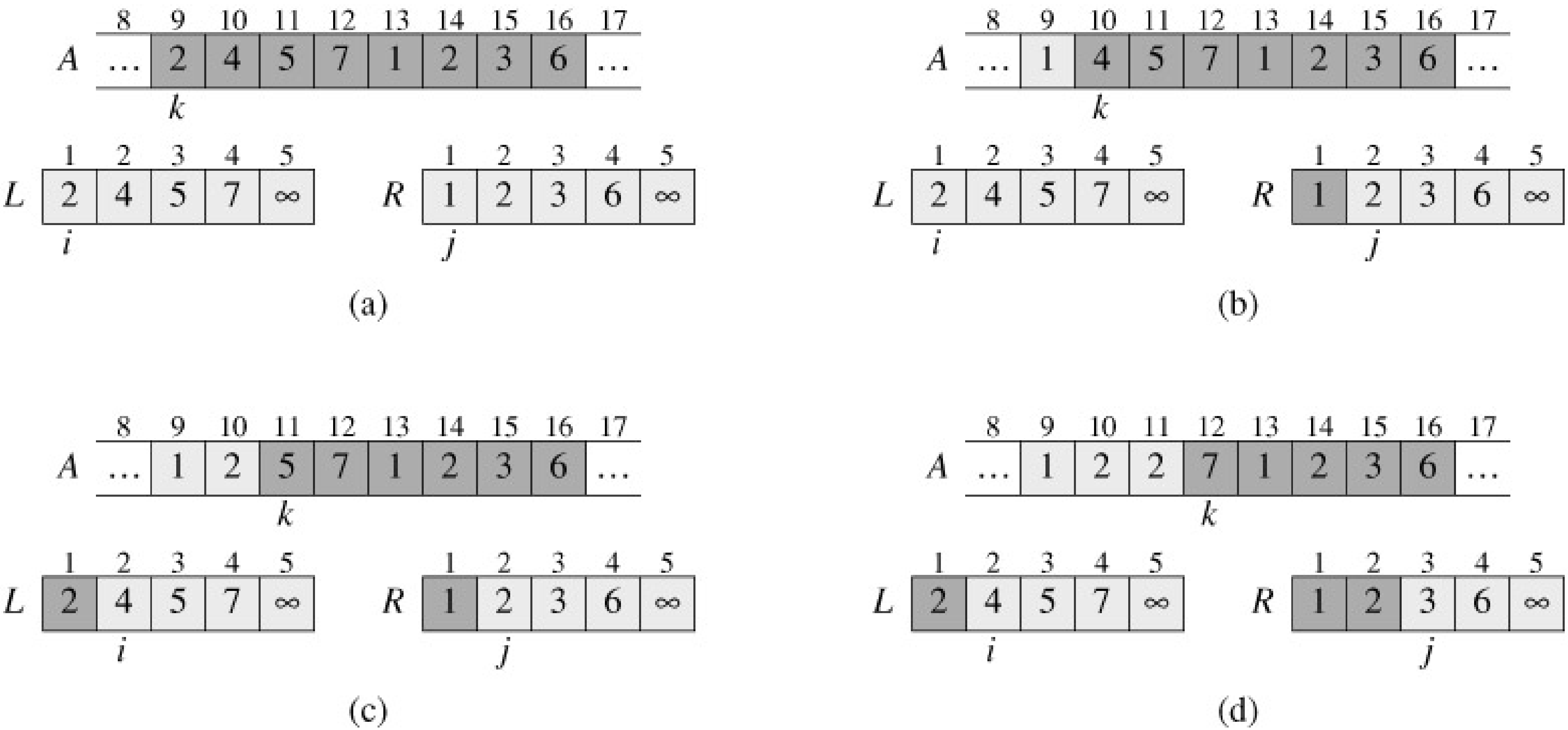
$$T(n) = \sum_{i=1}^{m} T(|P_i|) + D(n) + C(n)$$

Example 1: Merge Sort



1. **Divide:** *equally partition the array into two sub-arrays. (trivial)*
2. **Conquer:** *recursively sort two sub-arrays.*
3. **Combine:** *merge two sorted sub-arrays into one sorted array. (linear time)*

Merge


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
R 1 2 3 6 ∞
j
(a)
A … 1 4 5 7 1 2 3 6 …
(b)
A … 1 2 5 7 1 2 3 6 …
(c)
A … 1 2 2 7 1 2 3 6 …
(d)

Merge

```
MERGE(A, p, q, r)
n1 = q - p + 1
n2 = r - q
//create arrays L[1..n1 + 1] and R[1..n2 + 1]
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
```

D&C Algorithm

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

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

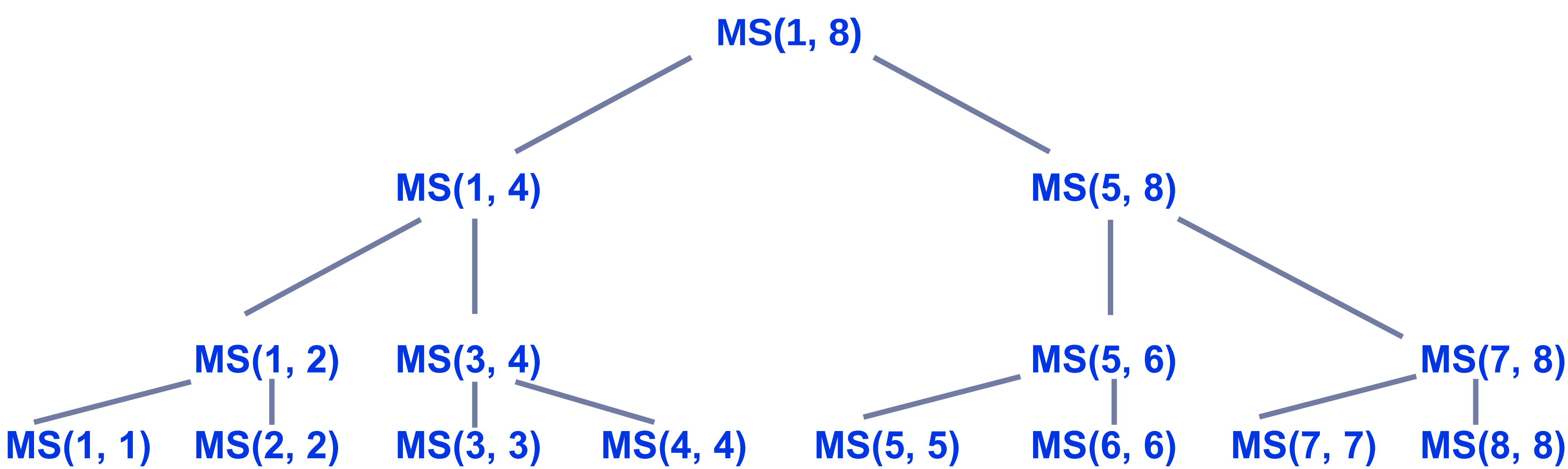
 # subproblems

Subproblem size

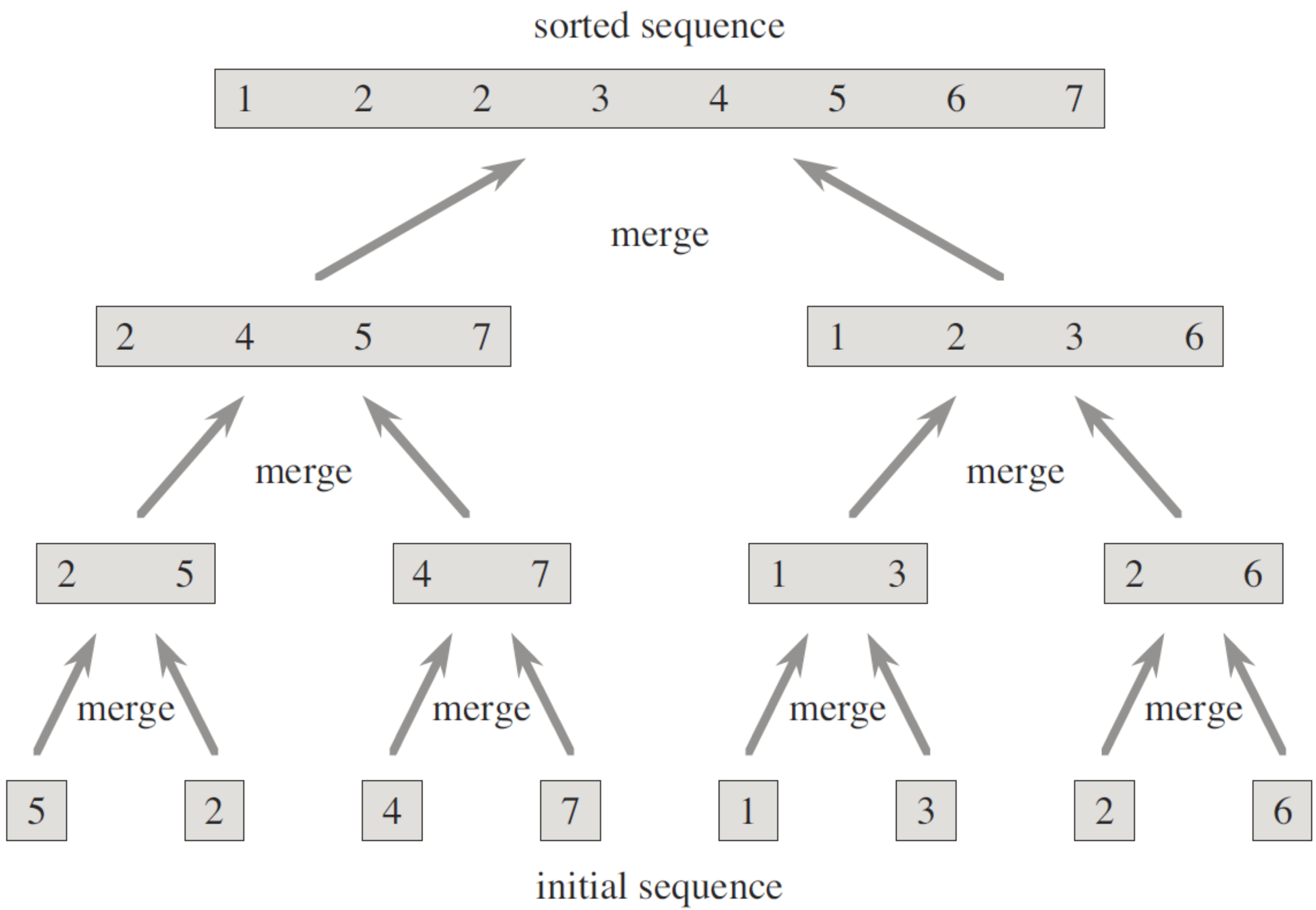
Time for the divide and combine steps

D&C Algorithm

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



D&C Algorithm



In-class Exercise

- Inversion:
  - Let $A[1..n]$ be an array of n distinct numbers. If $i < j$ and $A[i] > A[j]$, then the pair (i, j) is called an inversion of A.
  - Use global variable $count$ to denote the number of inversions. How to modify function MERGE to compute $count$?

```
MERGE(A, p, q, r)
n1 = q - p + 1
n2 = r - q
//create arrays L[1..n1 + 1] and R[1..n2 + 1]
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
```

Design Points

- Correctness:
  - ***Tip1:*** the divide step: subproblems must be the SAME problem.
  - ***Tip2:*** the divide and combine steps (incremental): can be verified by loop invariant analysis.
  - ***Tip3:*** the whole D&C algorithm: mathematical induction on the recursive structure.
- Efficiency:
  - number of subproblems
  - size of subproblems
  - time for the divide and combine steps.

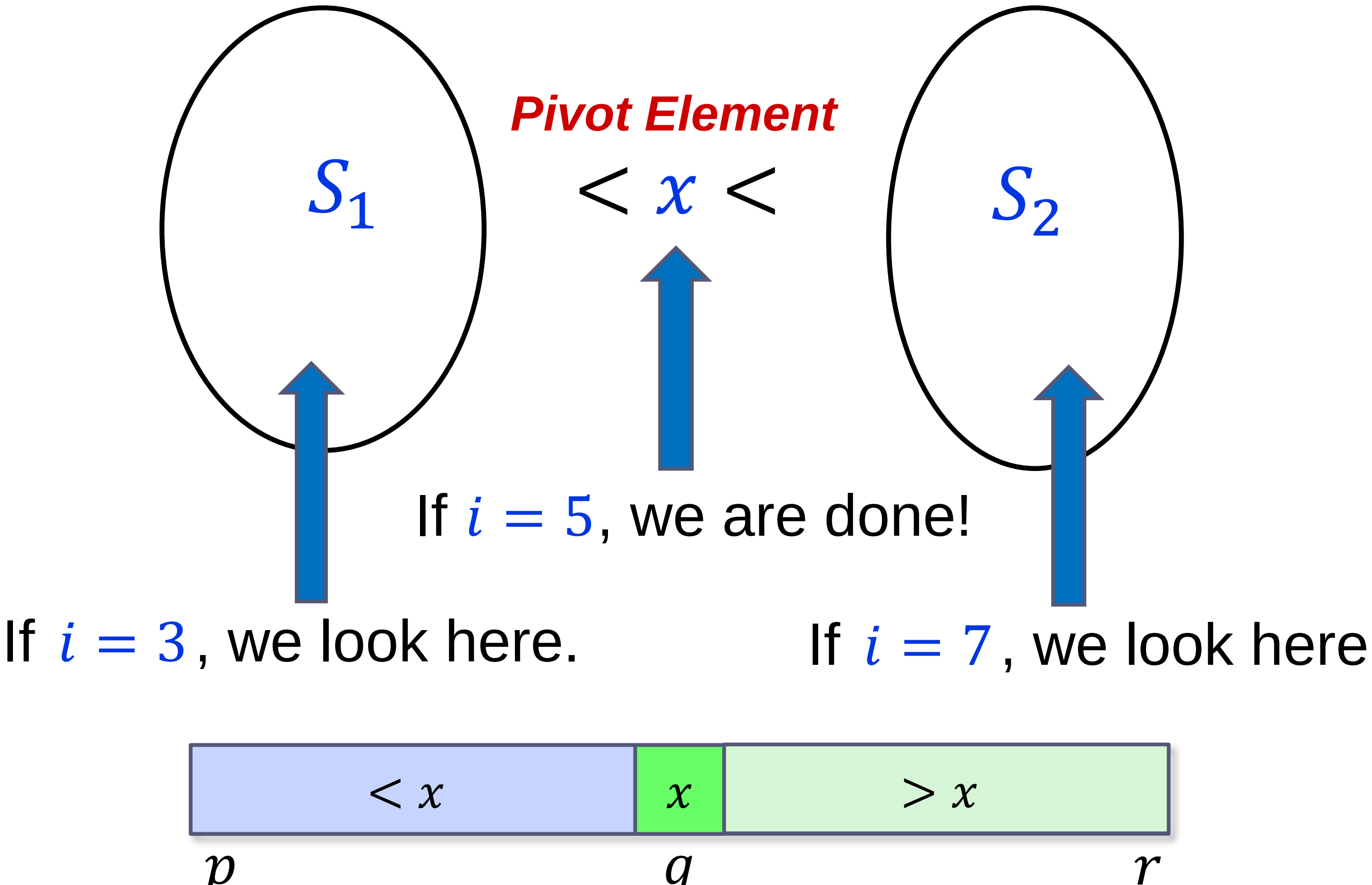
Order Statistics

- Find the i^{th} order statistic
- **Input:** Given an array A of n distinct numbers and an integer i, with $1 \leq i \leq n$.
- **Output:** The element $x \in A$ that is larger than exactly $i - 1$ other elements of A.

- Partition the input array differently!

Example 2: SELECT

Suppose S_1 has 4 elements, S_2 has 6 elements:



Partition

PARTITION (A, p, r) ▷ $A[p..r]$

$x = A[p]$ ▷ pivot = $A[p]$

$i = p$

for $j = p + 1$ **to** r

  if $A[j] < x$

    $i = i + 1$

    exchange$(A[i], A[j])$

exchange$(A[p], A[i])$

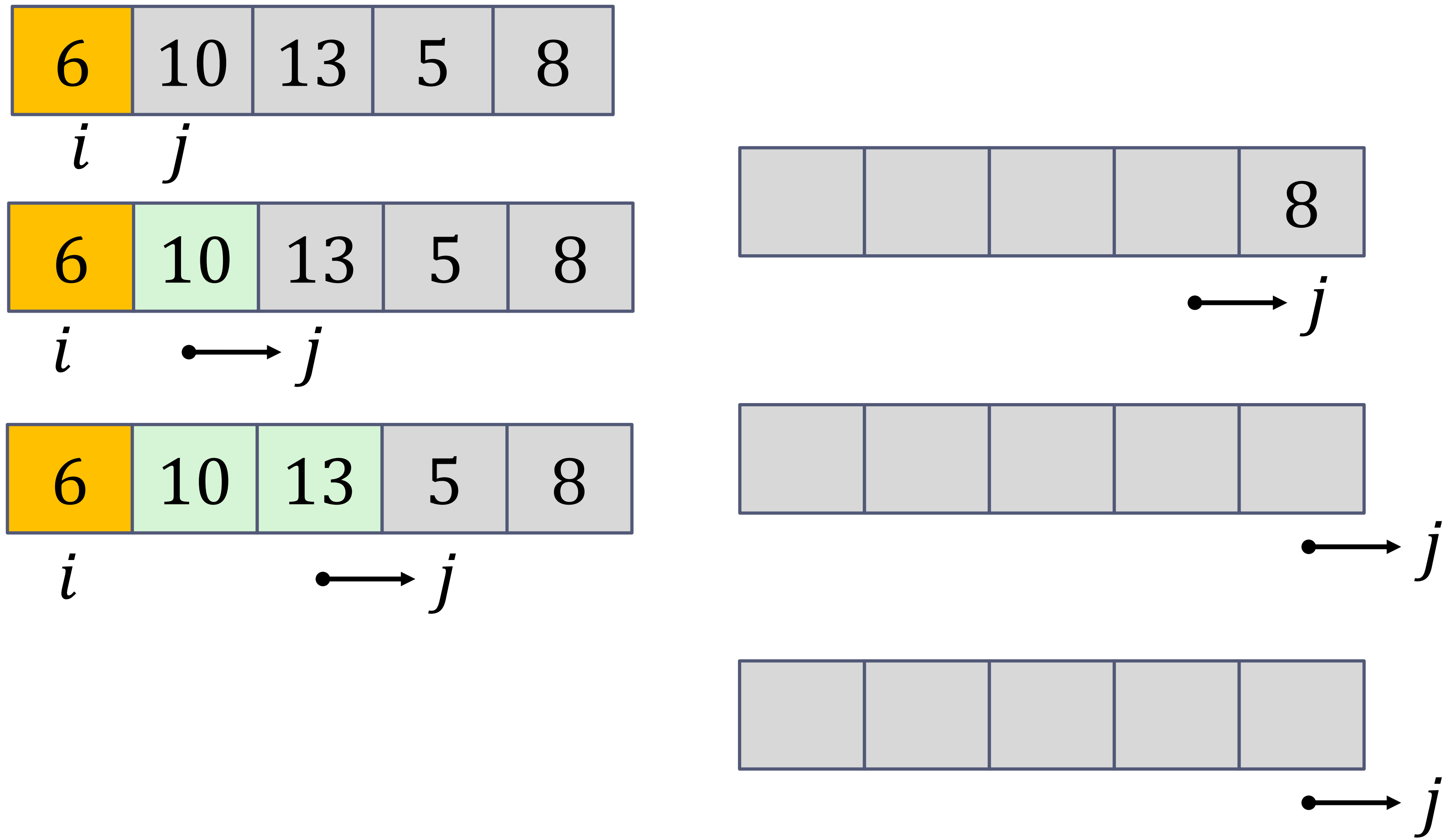
return i

i : pointing to the last value that is smaller than $A[p]$.

invariant:

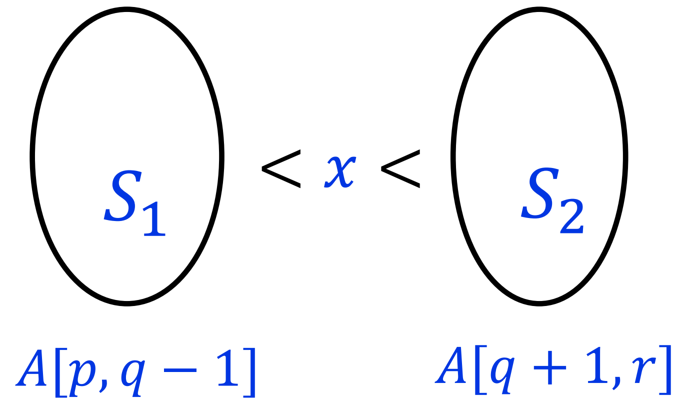
x	$< x$	$> x$	?
p	i	j	r

Exercise



SELECT

```
SELECT(A, p, r, i)
   if p == r
        return A[p]
   q = PARTITION(A, p, r)
   k = q - p + 1
    if i == k
        return A[q]
   elseif i < k
        return SELECT(A, p, q - 1, i)
    else return SELECT(A, q + 1, r, i - k)
```



Initial call: SELECT($A, 1, n, i$)

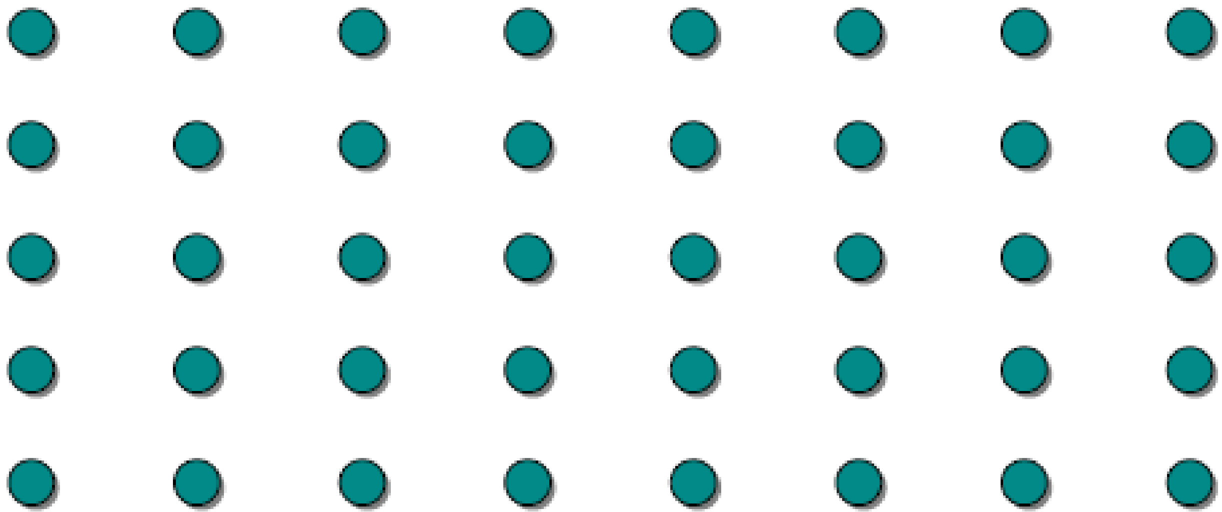
Worst-case

- The input array is sorted.
- Partition around the min or max element.
- One side of the partition always has no elements.

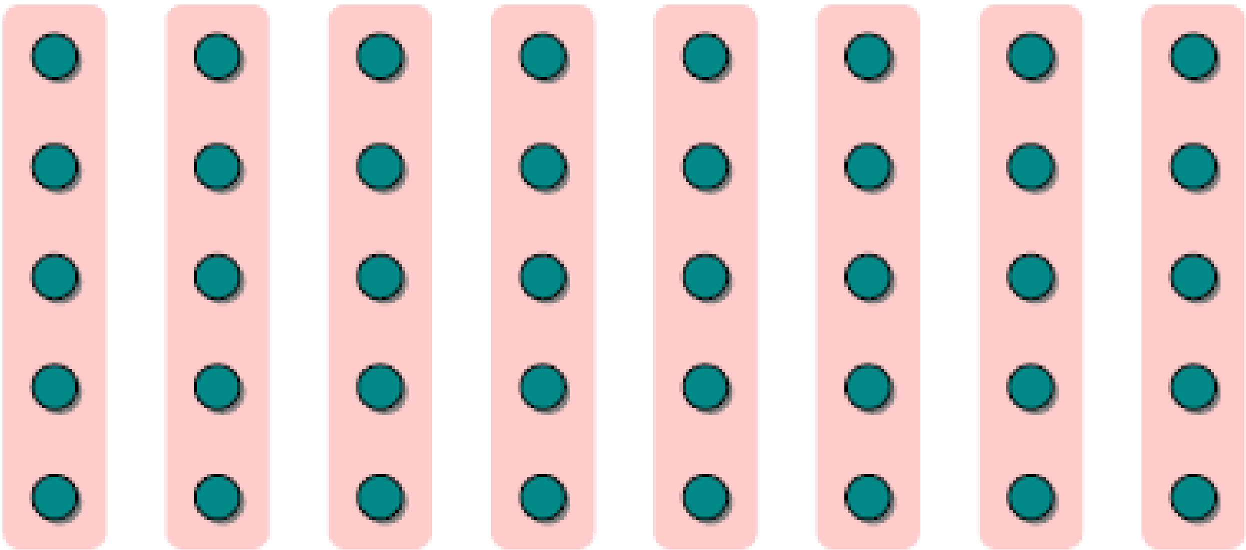
$$
\begin{aligned}
T(n) &= T(n-1) + cn \\
&= T(n-2) + c(n-1) + cn \\
&= \cdots \\
&= c\sum_{i=1}^{n} i \\
&= \theta(n^2)
\end{aligned}
$$

(arithmetic series)

Choosing the pivot

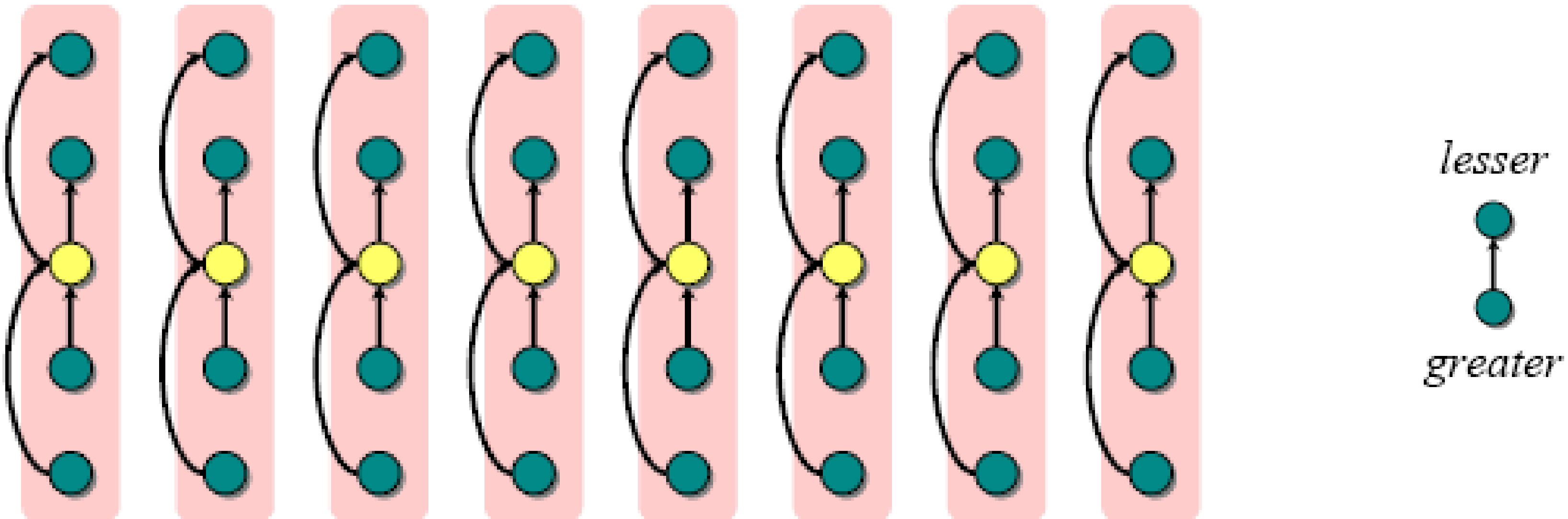


Choosing the pivot



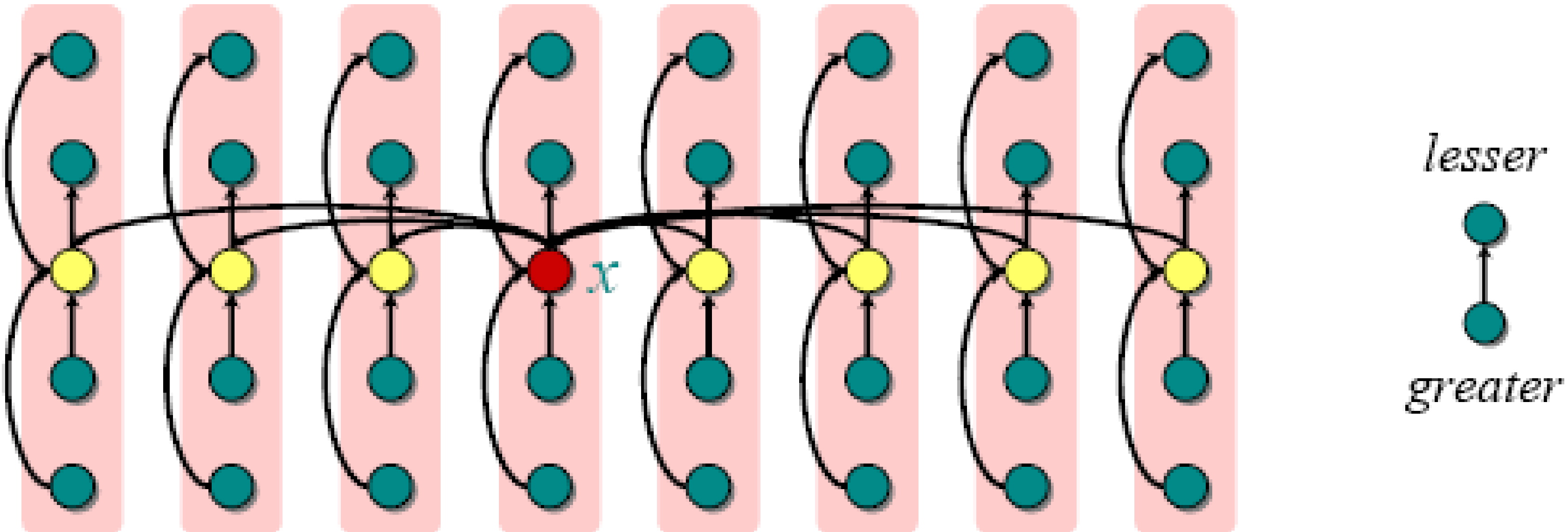
1.1 Divide the n elements into groups of 5.

Choosing the pivot



1.2 Divide the n elements into groups of 5. Find the median of each 5-element group by insertion sort and picking the median (yellow) from the sorted list.

Choosing the pivot



2. Recursively **SELECT** the median x of the $\lfloor n/5 \rfloor$ group medians to be the pivot.

Linear-time SELECT

SELECT (A, p, r, i)

if $(r - p + 1) < 100$ **return** DirectSelect(A, p, r, i)

Divide the elements in $A[p..r]$ into groups of 5. Find the median (group medians) of each 5-element group by insertion sort and save them in B.

$x =$ SELECT$\left(B, 1, \left\lfloor \frac{n}{5} \right\rfloor, \left\lfloor \frac{n}{10} \right\rfloor\right)$

$q =$ PARTITION(A, p, r, x)

$k = q - p + 1$

if $i == k$

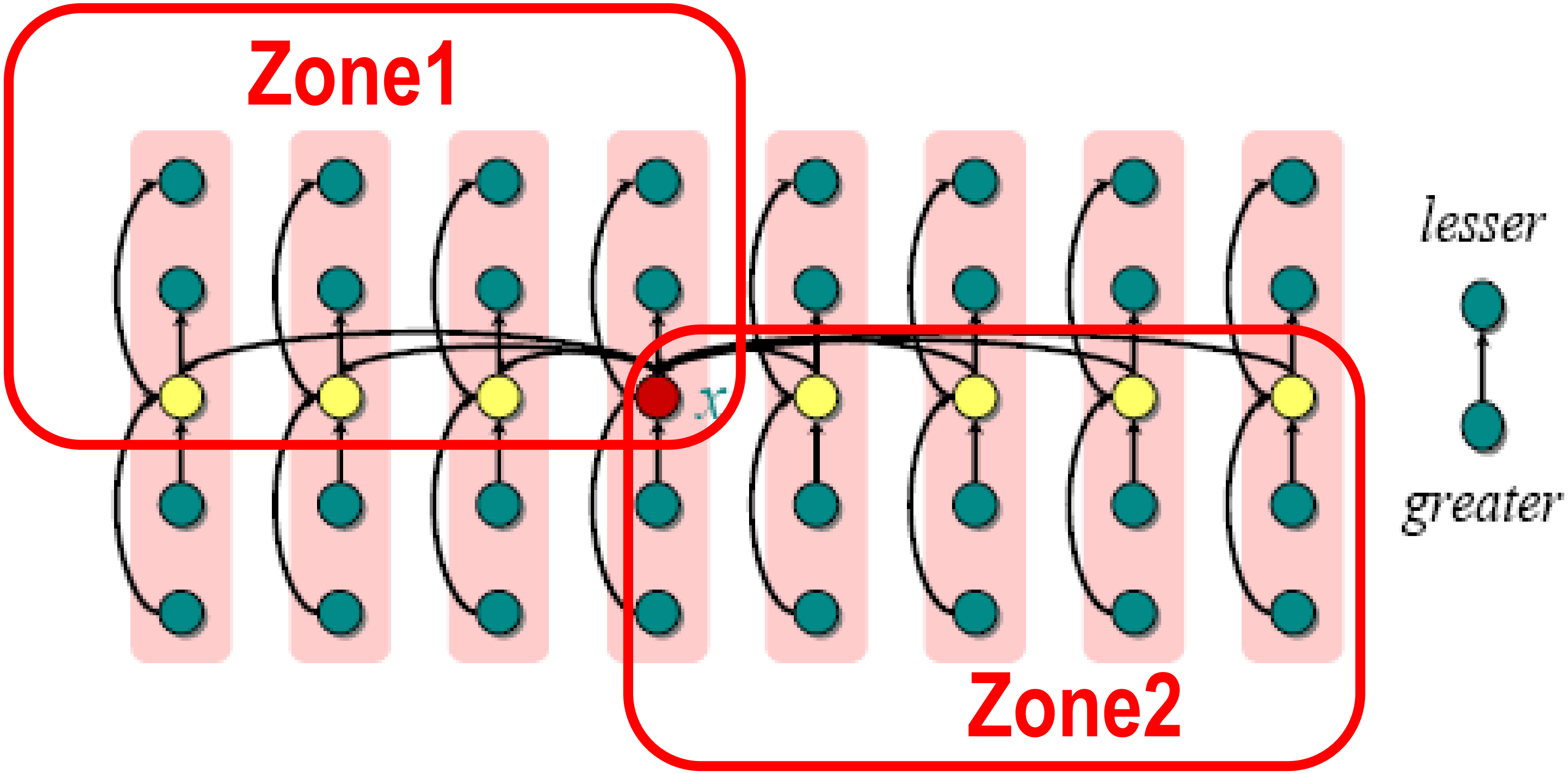
    return $A[q]$

else $i < k$

    return SELECT$(A, p, q - 1, i)$

else return SELECT$(A, q + 1, r, i - k)$

Analysis



Partition the input array around x into S_1, S_2, whose sizes are at most $7\lfloor n/10 \rfloor$.

Recurrence

$T(n) \leq$

$O(n)$

SELECT(A, p, r, i)

1. Divide the elements in $A[p..r]$ into groups of 5. Find the median (group medians) of each 5-element group by insertion sort and save them in B.

$T(n/5)$

2. $x =$ SELECT$(B, 1, \left\lceil \frac{n}{5} \right\rceil, \lfloor \frac{n}{10} \rfloor)$

$O(n)$

3. $q =$ PARTITION(A, p, r, x)

$T(7n/10)$

4. **if** $i == k$ **return** $A[q]$
   else $i < k$
   return SELECT$(A, p, q-1, i)$
   else return SELECT$(A, q+1, r, i-k)$

Group Discussion

- Q1: ***Implementation:*** When can D&C algorithms be implemented iteratively? And when recursively?
- Q2: ***Efficiency:*** How to design D&C algorithms in order to achieve linear algorithms?

Summary

- D&C is another powerful technique for algorithm design.
- D&C algorithms are ***recursive*** in nature, and can be analyzed using recurrences and the master method.
- D&C algorithms can be implemented iteratively or recursively.
- The ***efficiency*** of D&C algorithms depends on the number of subproblems, the size of subproblems, and time for the divide and combine steps.