CS 310: Algorithms

Lecture 10

Instructor: Naveed Anwar Bhatti

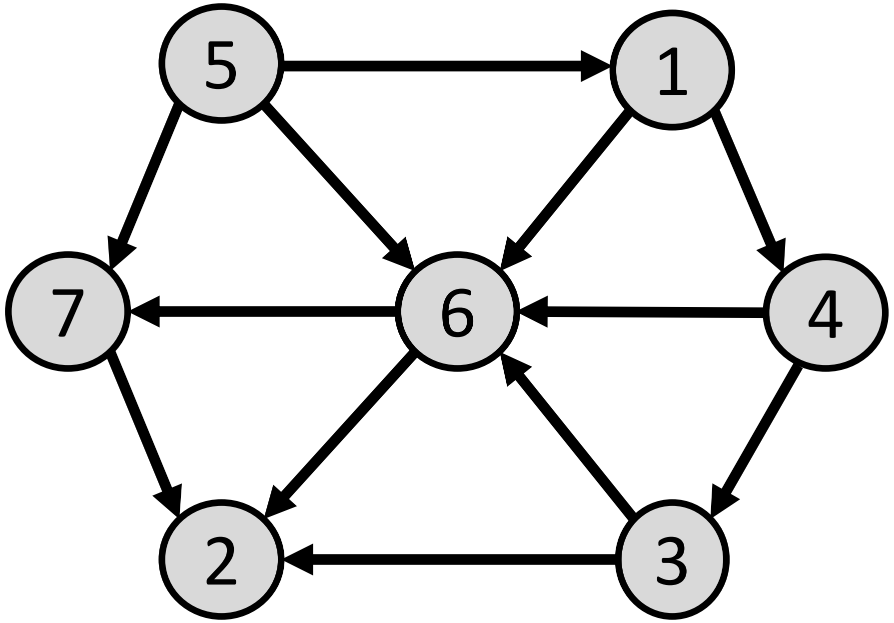
Administrivia

Quiz 3 on Monday

3 grace days

Topological Ordering: Live Poll 1

Select the correct topological order:



A. Topological Order is: 1-2-3-4-5-6-7
B. Topological Order is: 2-3-1-4-5-6-7
C. Topological Order is: 5-1-4-3-6-7-2
D. Topological Order is: 2-3-6-5-1-4-7
E. Topological Order does not exist

Scan the QR code to vote or go to https://forms.office.com/r/Rh8UH9t7PB

Topological Ordering: Live Poll 1
Only people in my organization can respond, Record name

1. Select the correct topological order:

Topological Order is: 1-2-3-4-5-6-7	0%
Topological Order is: 2-3-1-4-5-6-7	0%
Topological Order is: 5-1-4-3-6-7-2	100%
Topological Order is: 2-3-6-5-1-4-7	0%
Topological Order does not exist	0%

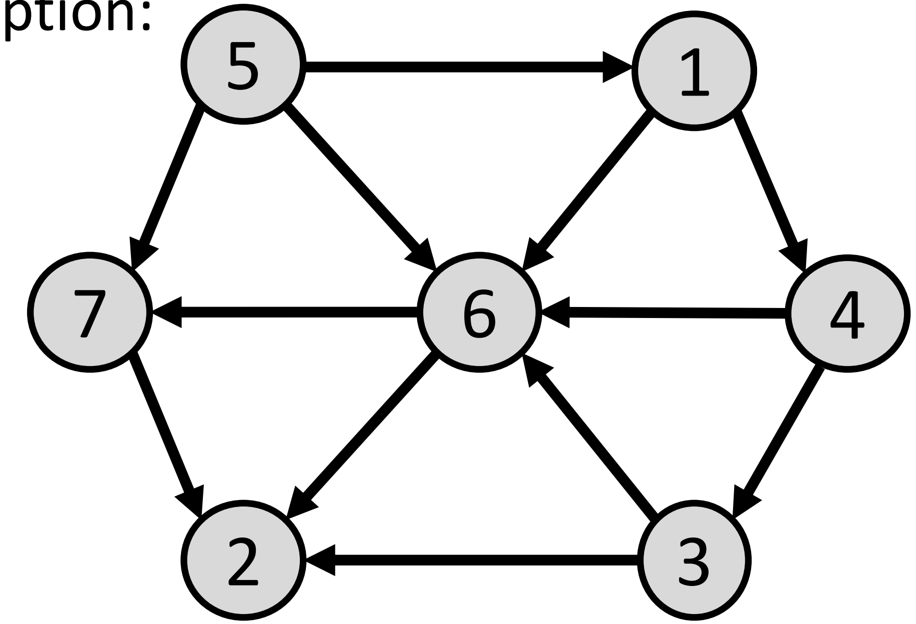
32 responses

1/1

Scan the QR code to vote or go to https://forms.office.com/r/Rh8UH9t7PB

Topological Ordering: Live Poll 1

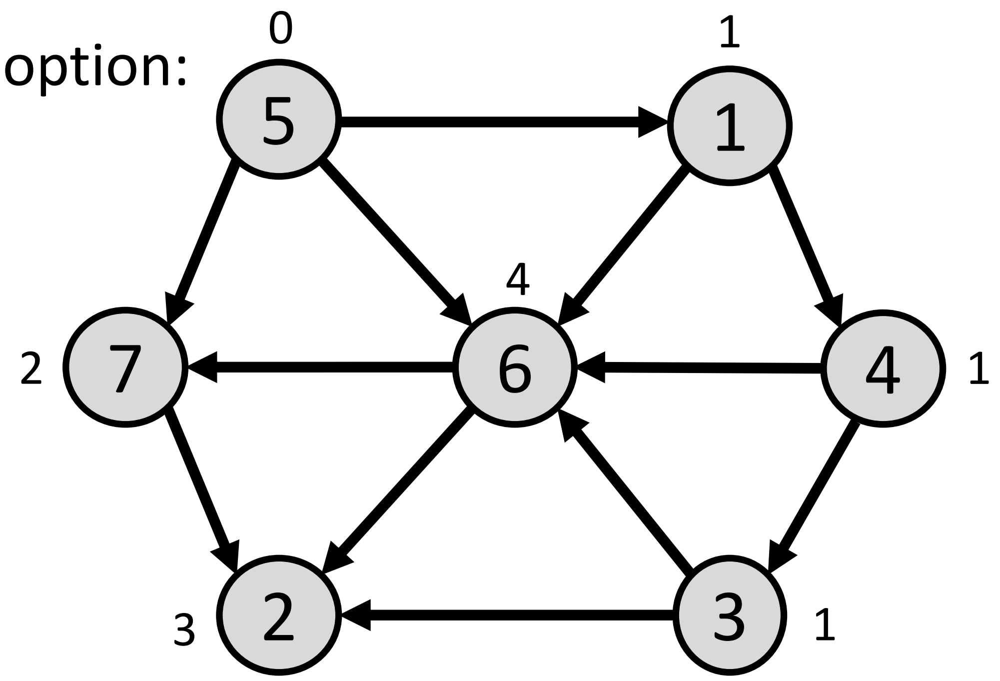
Select the correct option:



A. Topological Order is: 1-2-3-4-5-6-7
B. Topological Order is: 2-3-1-4-5-6-7
C. Topological Order is: 5-1-4-3-6-7-2
D. Topological Order is: 2-3-6-5-1-4-7
E. Topological Order does not exist

Topological Ordering: Live Poll 1

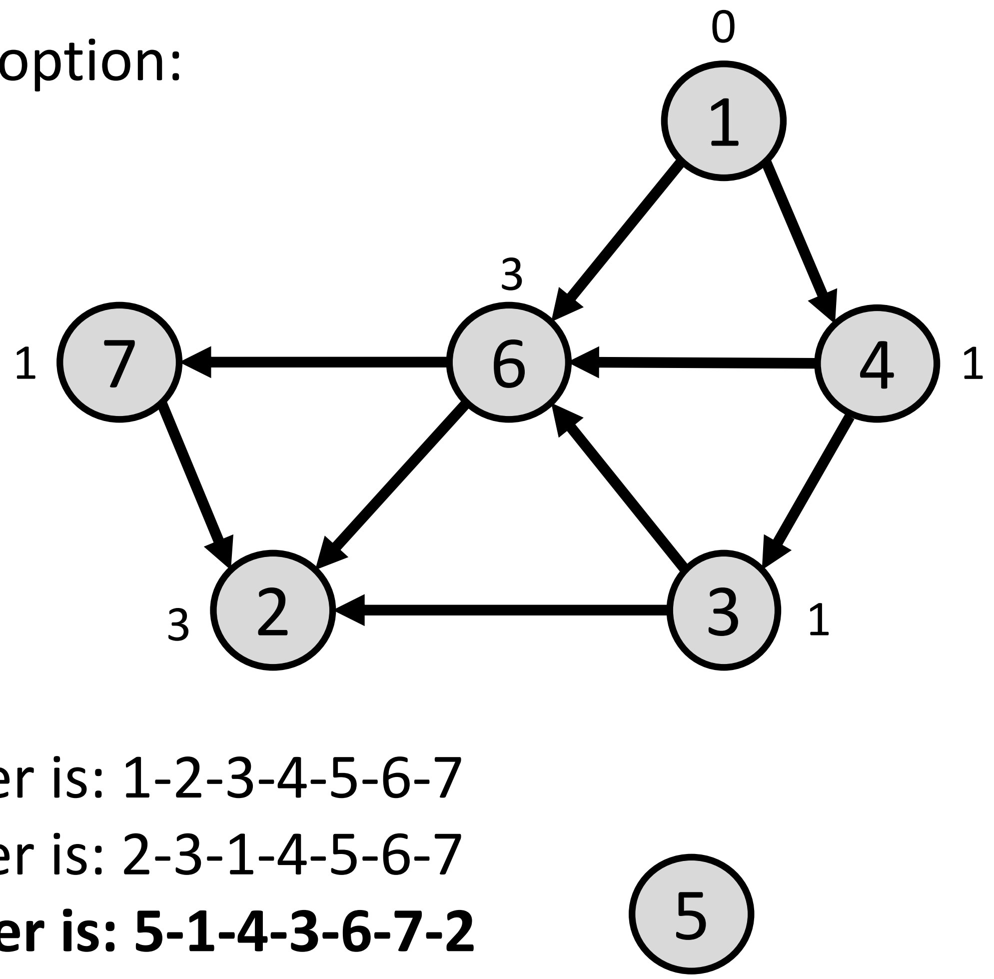
Select the correct option:



A. Topological Order is: 1-2-3-4-5-6-7
B. Topological Order is: 2-3-1-4-5-6-7
C. Topological Order is: 5-1-4-3-6-7-2
D. Topological Order is: 2-3-6-5-1-4-7
E. Topological Order does not exist

Topological Ordering: Live Poll 1

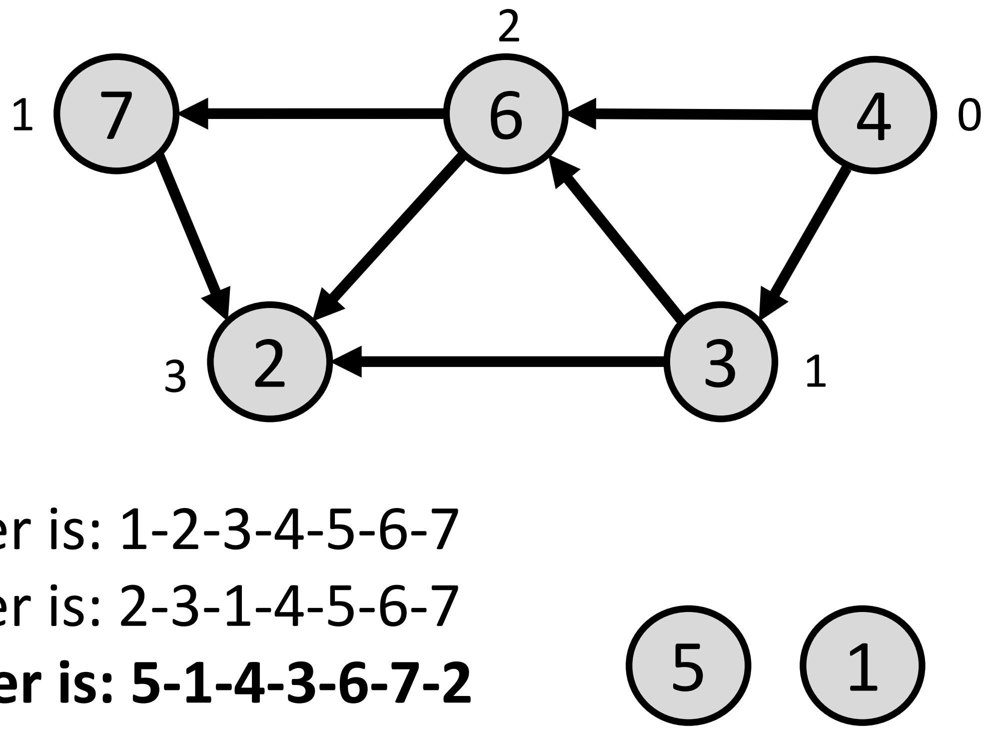
Select the correct option:



A. Topological Order is: 1-2-3-4-5-6-7
B. Topological Order is: 2-3-1-4-5-6-7
C. Topological Order is: 5-1-4-3-6-7-2
D. Topological Order is: 2-3-6-5-1-4-7
E. Topological Order does not exist

Topological Ordering: Live Poll 1

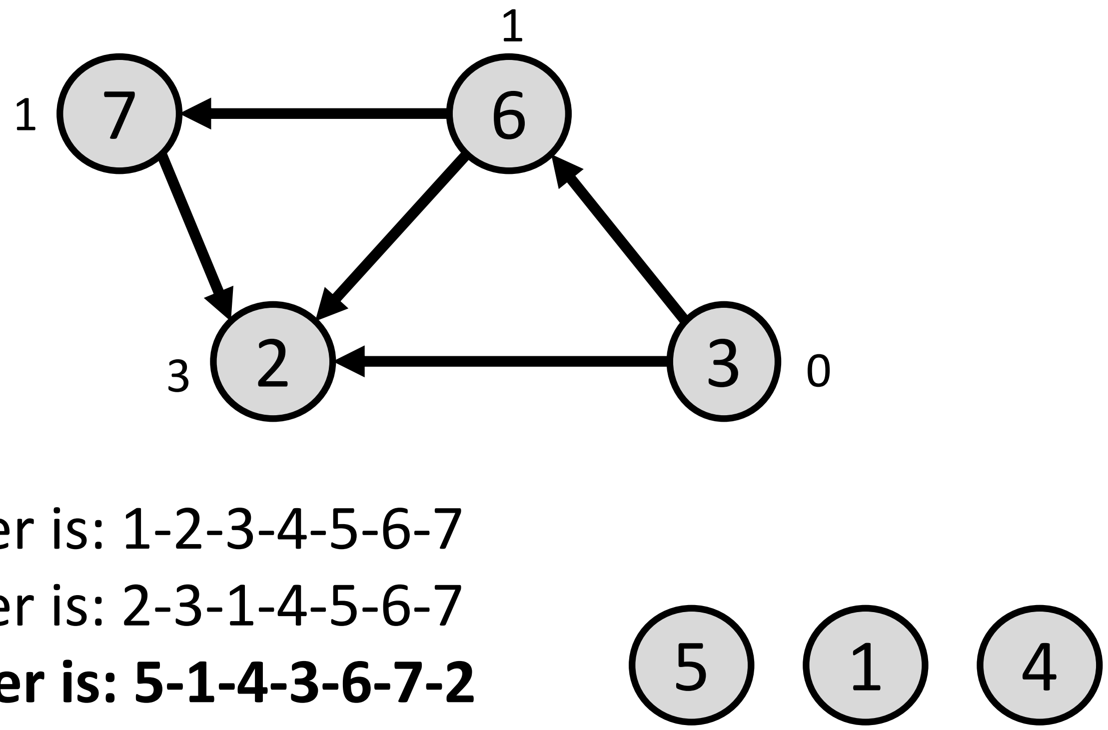
Select the correct option:



A. Topological Order is: 1-2-3-4-5-6-7
B. Topological Order is: 2-3-1-4-5-6-7
C. Topological Order is: 5-1-4-3-6-7-2
D. Topological Order is: 2-3-6-5-1-4-7
E. Topological Order does not exist

Topological Ordering: Live Poll 1

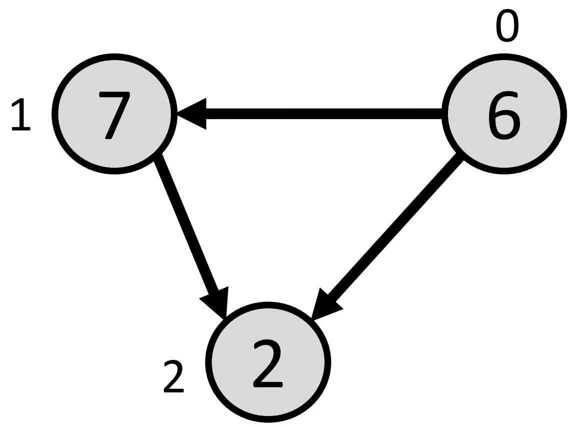
Select the correct option:



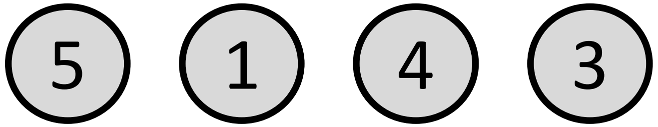
A. Topological Order is: 1-2-3-4-5-6-7
B. Topological Order is: 2-3-1-4-5-6-7
C. Topological Order is: 5-1-4-3-6-7-2
D. Topological Order is: 2-3-6-5-1-4-7
E. Topological Order does not exist

Topological Ordering: Live Poll 1

Select the correct option:

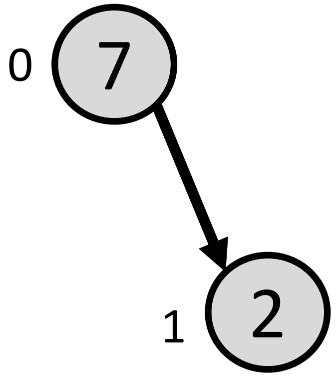


A. Topological Order is: 1-2-3-4-5-6-7
B. Topological Order is: 2-3-1-4-5-6-7
C. Topological Order is: 5-1-4-3-6-7-2
D. Topological Order is: 2-3-6-5-1-4-7
E. Topological Order does not exist



Topological Ordering: Live Poll 1

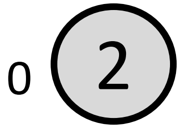
Select the correct option:



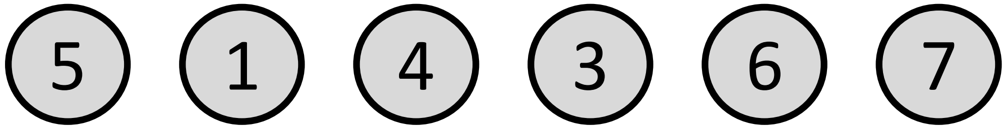
A. Topological Order is: 1-2-3-4-5-6-7
B. Topological Order is: 2-3-1-4-5-6-7
C. Topological Order is: 5-1-4-3-6-7-2
D. Topological Order is: 2-3-6-5-1-4-7
E. Topological Order does not exist

Topological Ordering: Live Poll 1

Select the correct option:



A. Topological Order is: 1-2-3-4-5-6-7
B. Topological Order is: 2-3-1-4-5-6-7
C. Topological Order is: 5-1-4-3-6-7-2
D. Topological Order is: 2-3-6-5-1-4-7
E. Topological Order does not exist



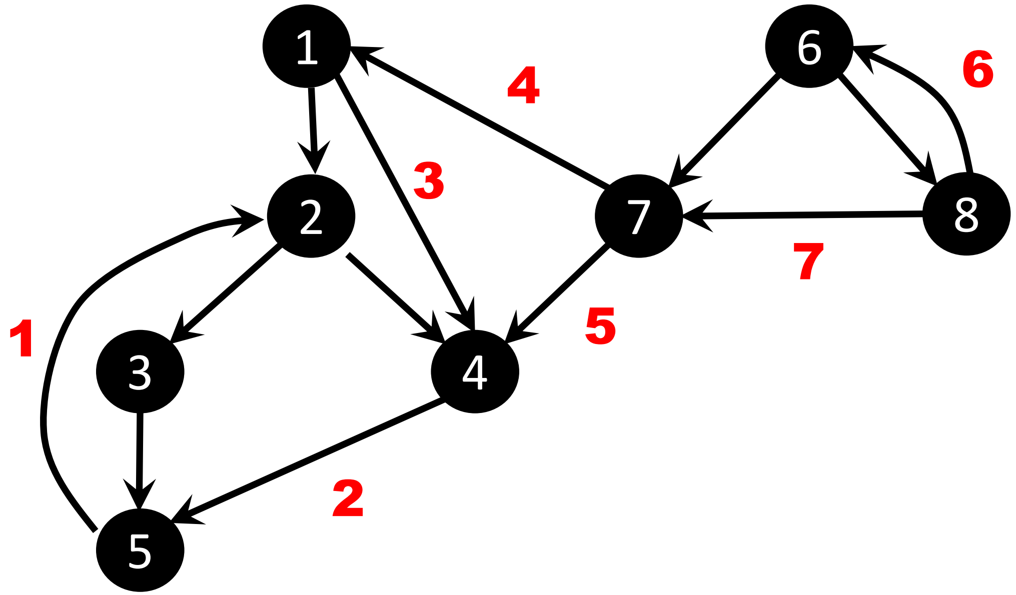
Topological Ordering: Live Poll 1

Select the correct option:

A. Topological Order is: 1-2-3-4-5-6-7
B. Topological Order is: 2-3-1-4-5-6-7
C. Topological Order is: 5-1-4-3-6-7-2
D. Topological Order is: 2-3-6-5-1-4-7
E. Topological Order does not exist

DFS: Live Poll 2

Label the numbered edges as ***Back***, ***Forward***, or ***Cross*** edges



A. B, F, F, C, C, B, F
B. B, C, C, F, C, F, C
C. B, C, B, F, C, F, C
D. B, C, F, C, C, B, C
E. B, C, F, B, C, F, C

Scan the QR code to vote or go to https://forms.office.com/r/MRrtZ1bsbn

DFS: Live Poll 2

Only people in my organization can respond, Record name

1. Label the numbered edges as Back, Forward, or Cross edges

B, F, F, C, C, B, F	24%
B, C, C, F, C, F, C	22%
B, C, B, F, C, F, C	2%
B, C, F, C, C, B, C	46%
B, C, F, B, C, F, C	5%

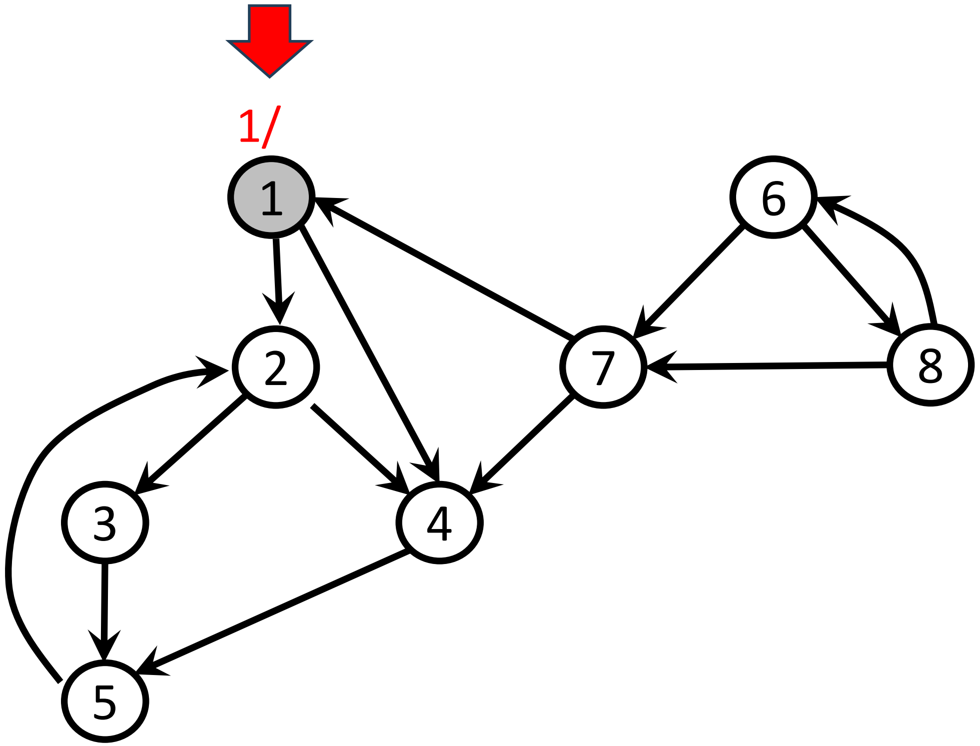
41 responses

1/1

Scan the QR code to vote or go to https://forms.office.com/r/MRrtZ1bsbn

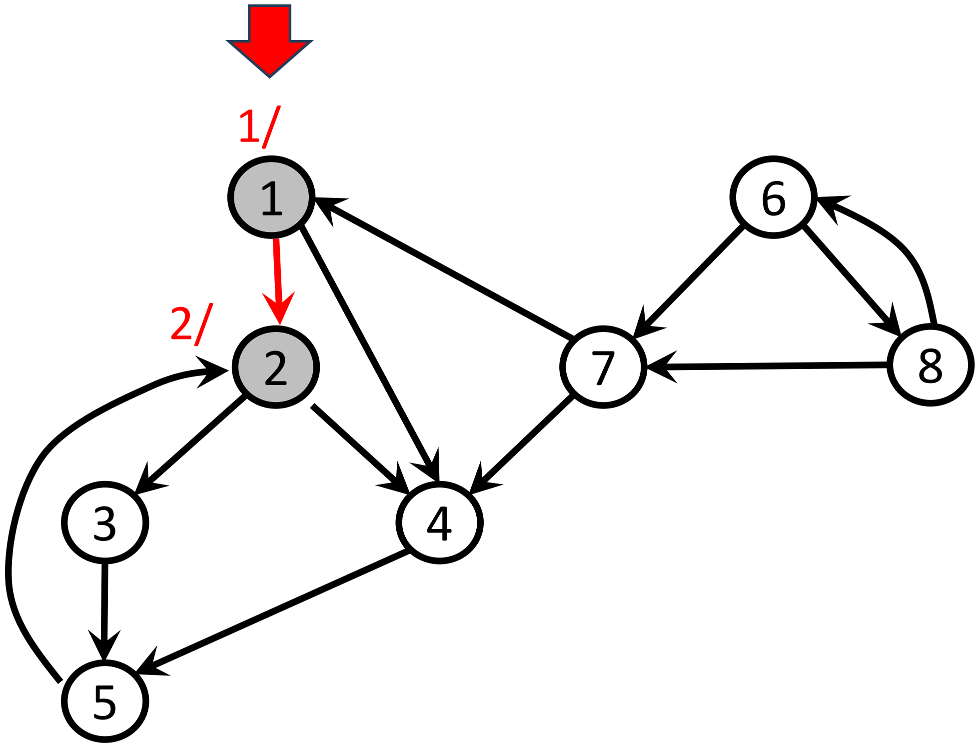
DFS: Live Poll 2

Label the numbered edges as ***Back***, ***Forward***, or ***Cross*** edges



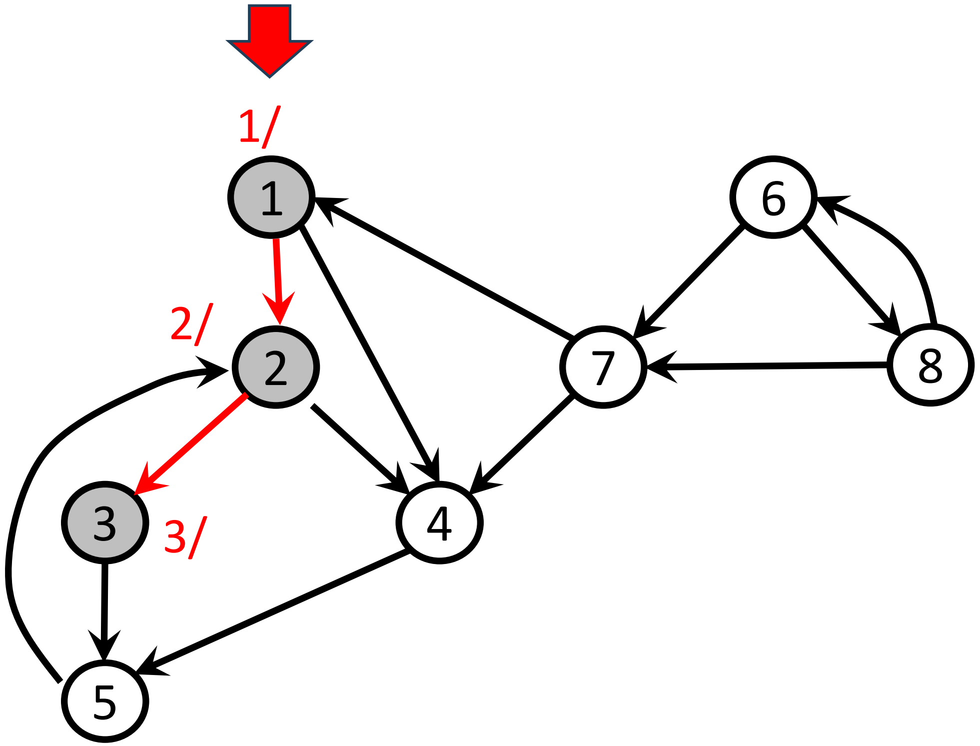
DFS: Live Poll 2

Label the numbered edges as ***Back***, ***Forward***, or ***Cross*** edges



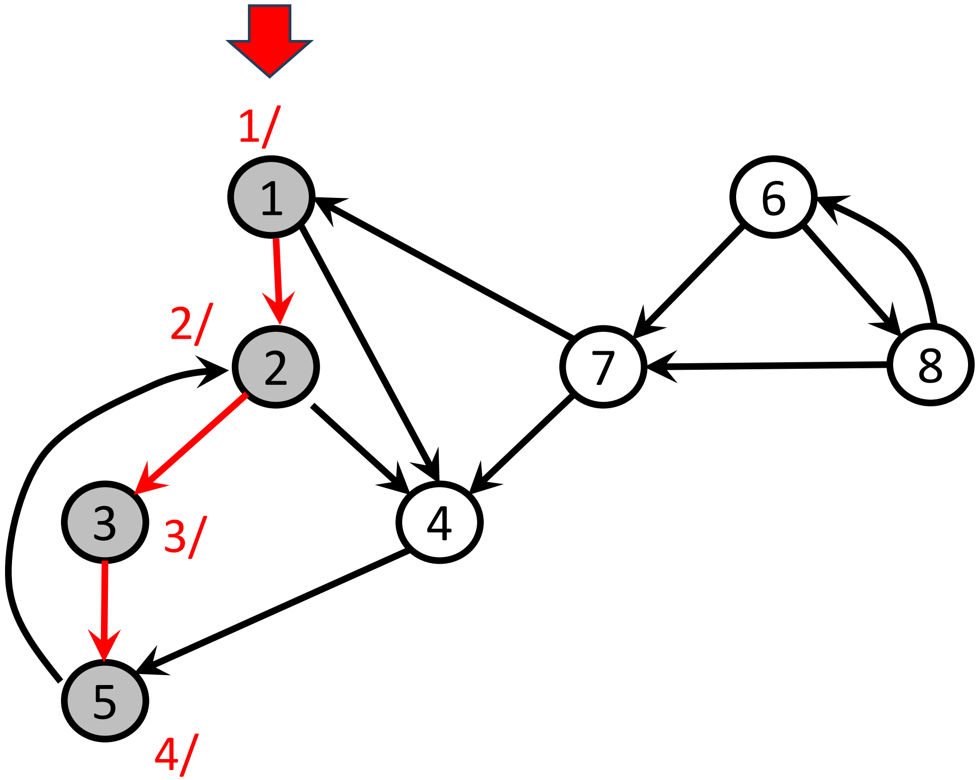
DFS: Live Poll 2

Label the numbered edges as ***Back***, ***Forward***, or ***Cross*** edges



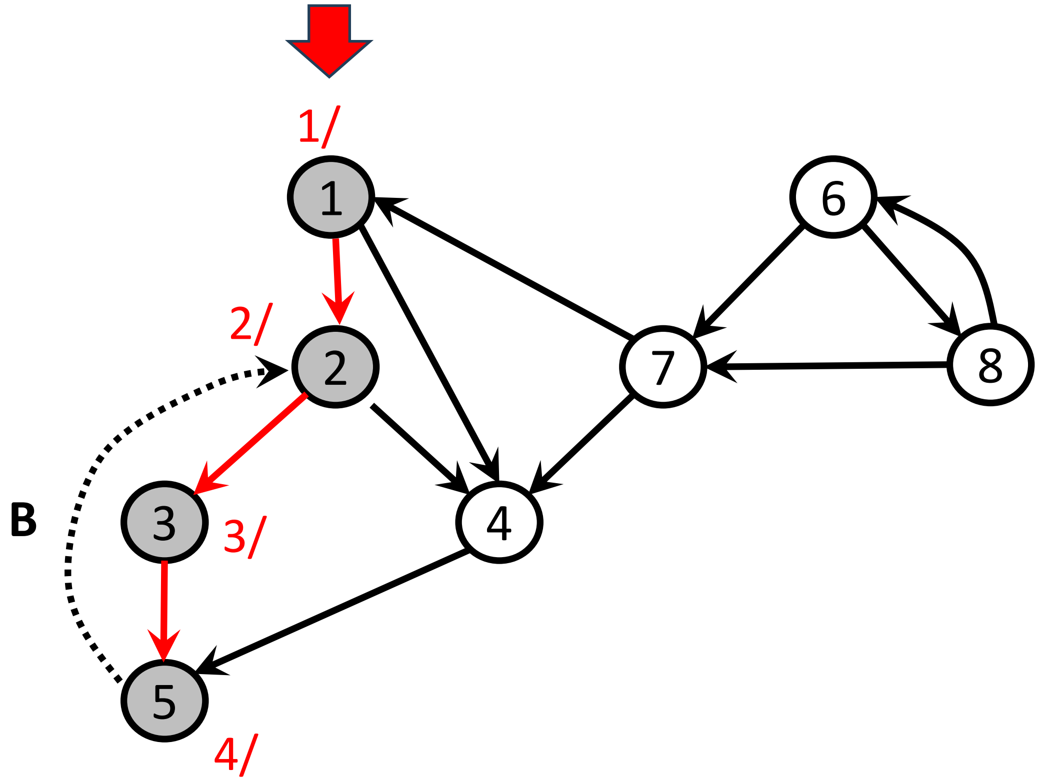
DFS: Live Poll 2

Label the numbered edges as ***Back***, ***Forward***, or ***Cross*** edges



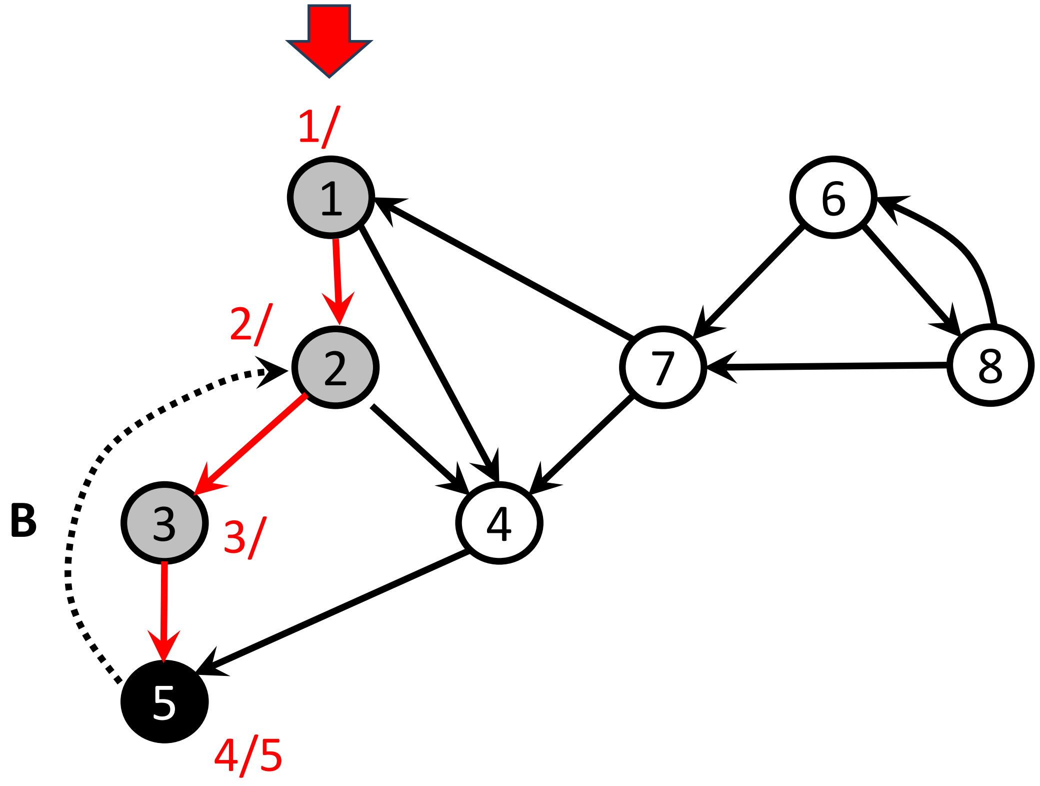
DFS: Live Poll 2

Label the numbered edges as ***Back***, ***Forward***, or ***Cross*** edges



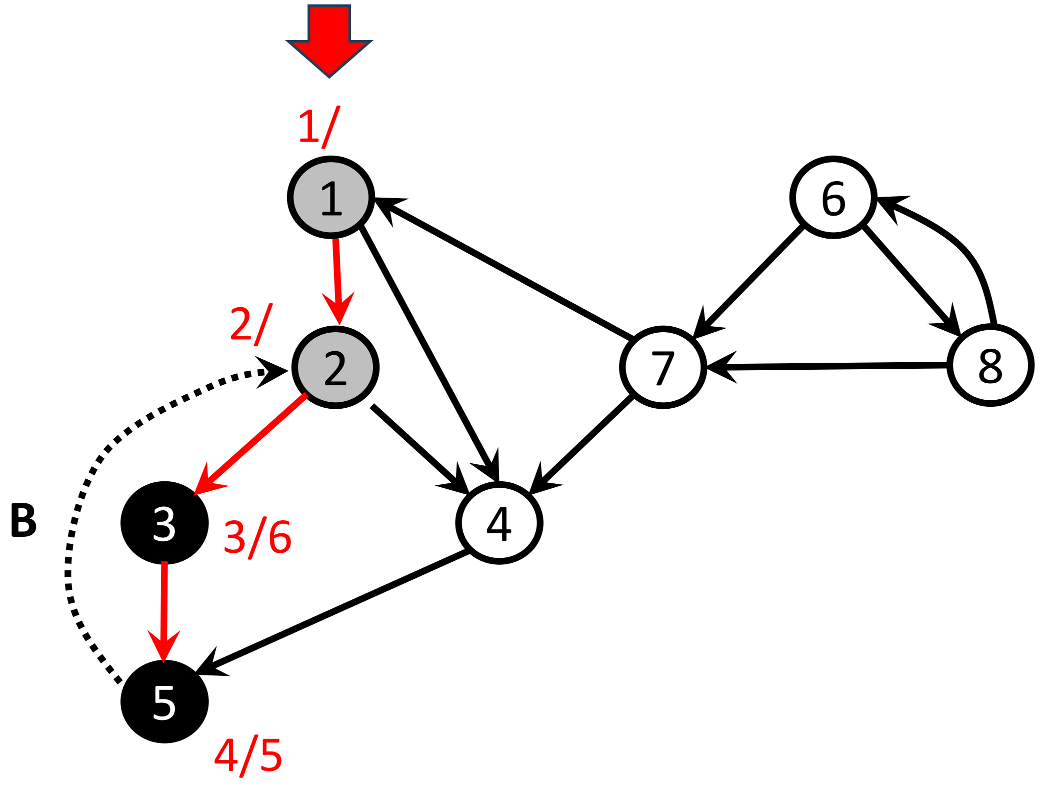
DFS: Live Poll 2

Label the numbered edges as ***Back***, ***Forward***, or ***Cross*** edges



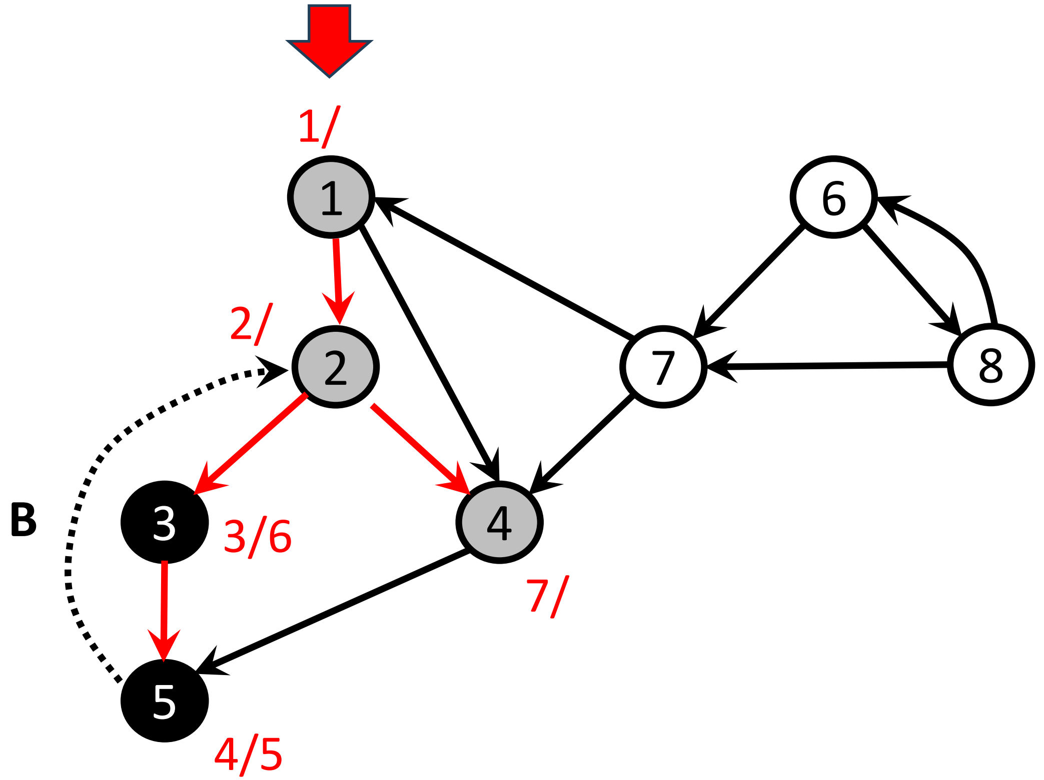
DFS: Live Poll 2

Label the numbered edges as ***Back***, ***Forward***, or ***Cross*** edges



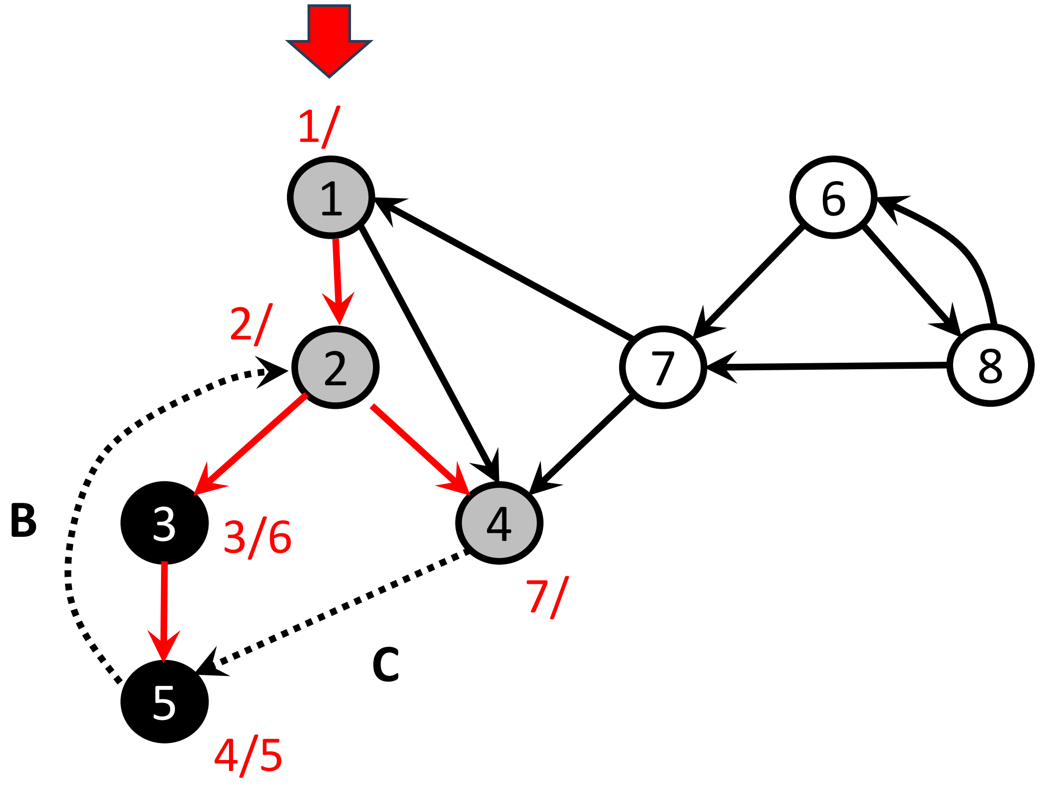
DFS: Live Poll 2

Label the numbered edges as ***Back***, ***Forward***, or ***Cross*** edges



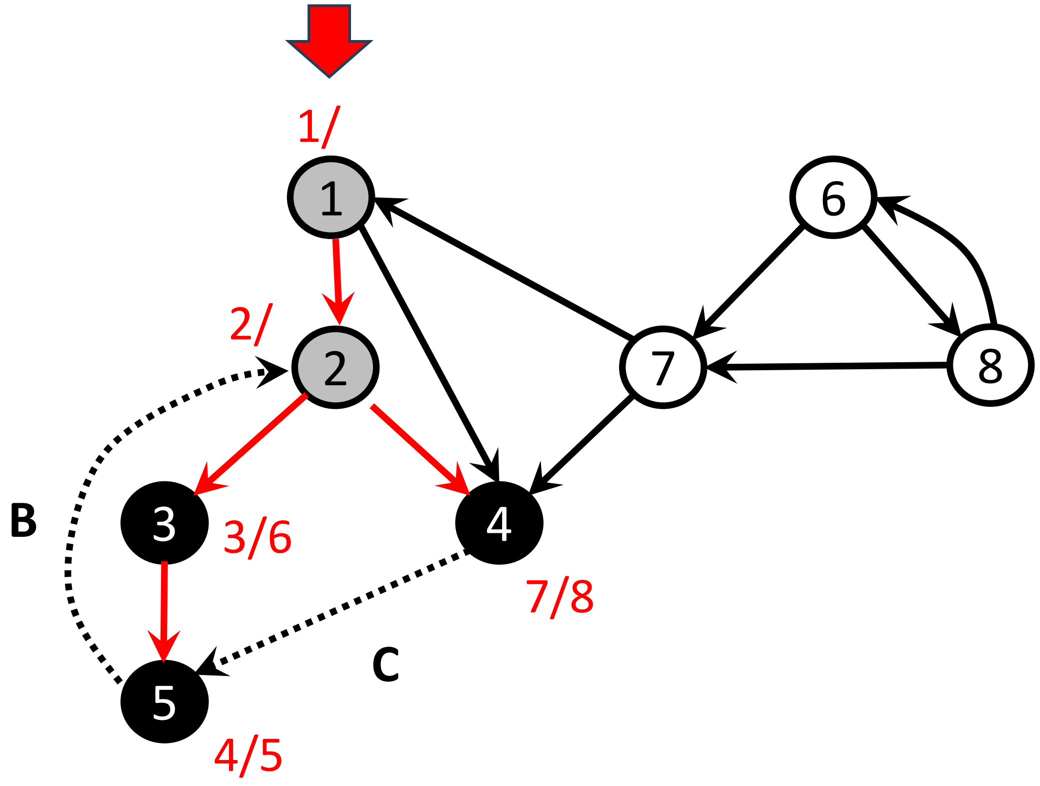
DFS: Live Poll 2

Label the numbered edges as ***Back***, ***Forward***, or ***Cross*** edges



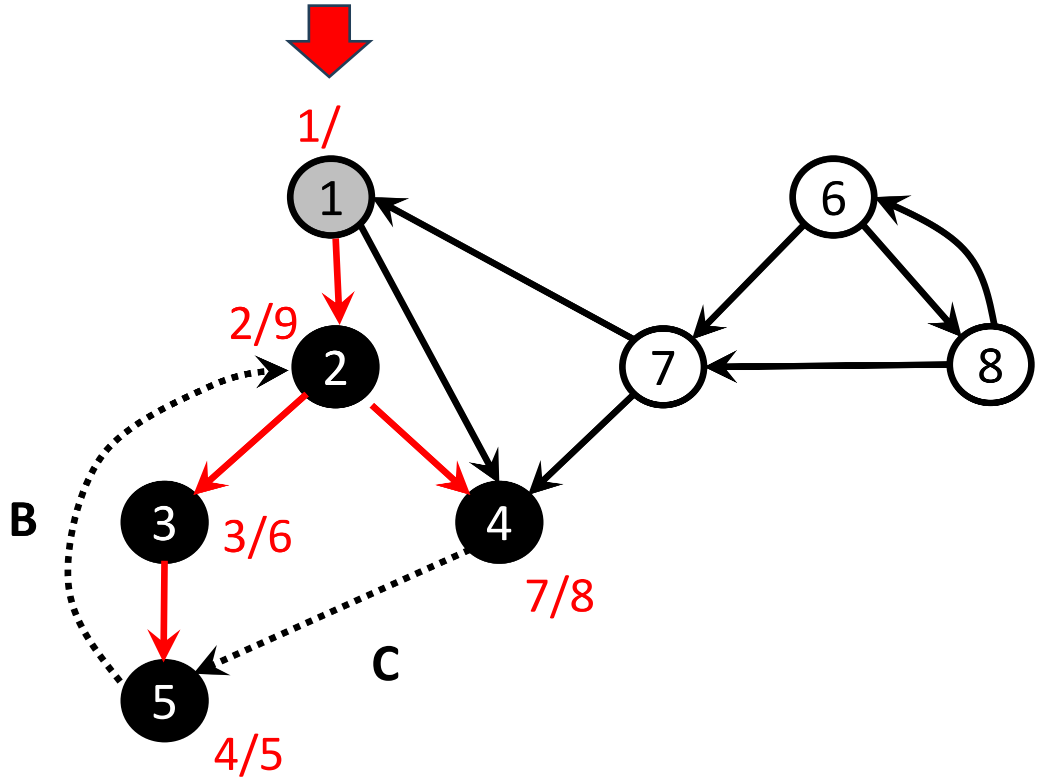
DFS: Live Poll 2

Label the numbered edges as ***Back***, ***Forward***, or ***Cross*** edges



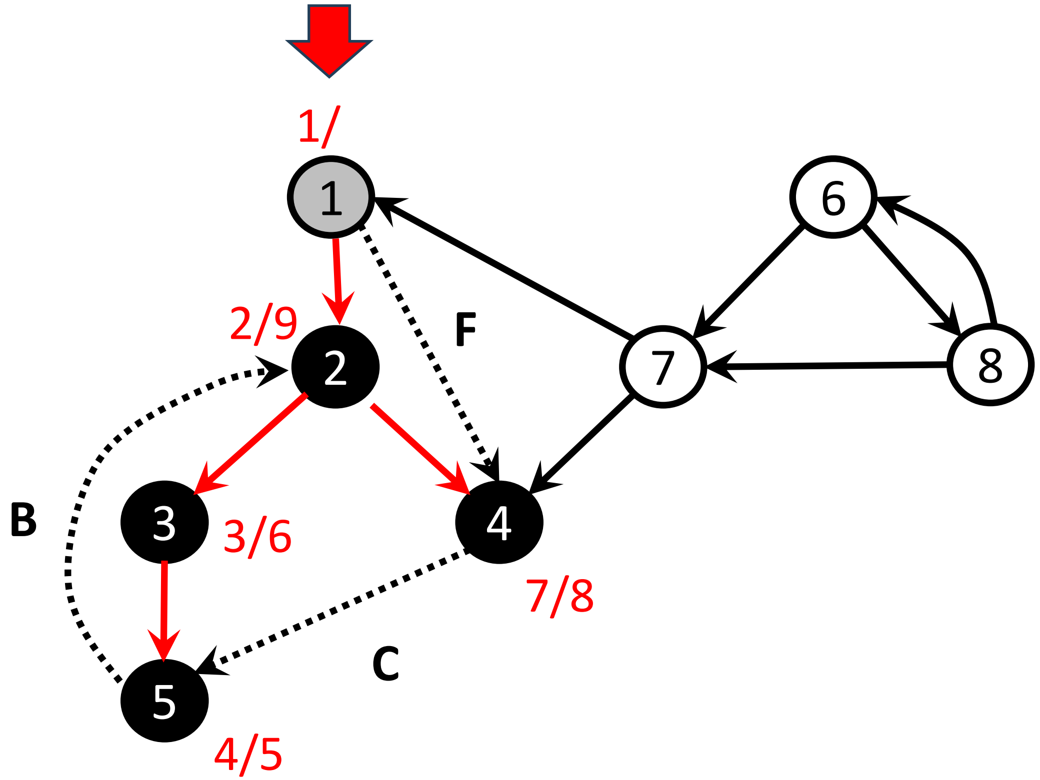
DFS: Live Poll 2

Label the numbered edges as ***Back***, ***Forward***, or ***Cross*** edges



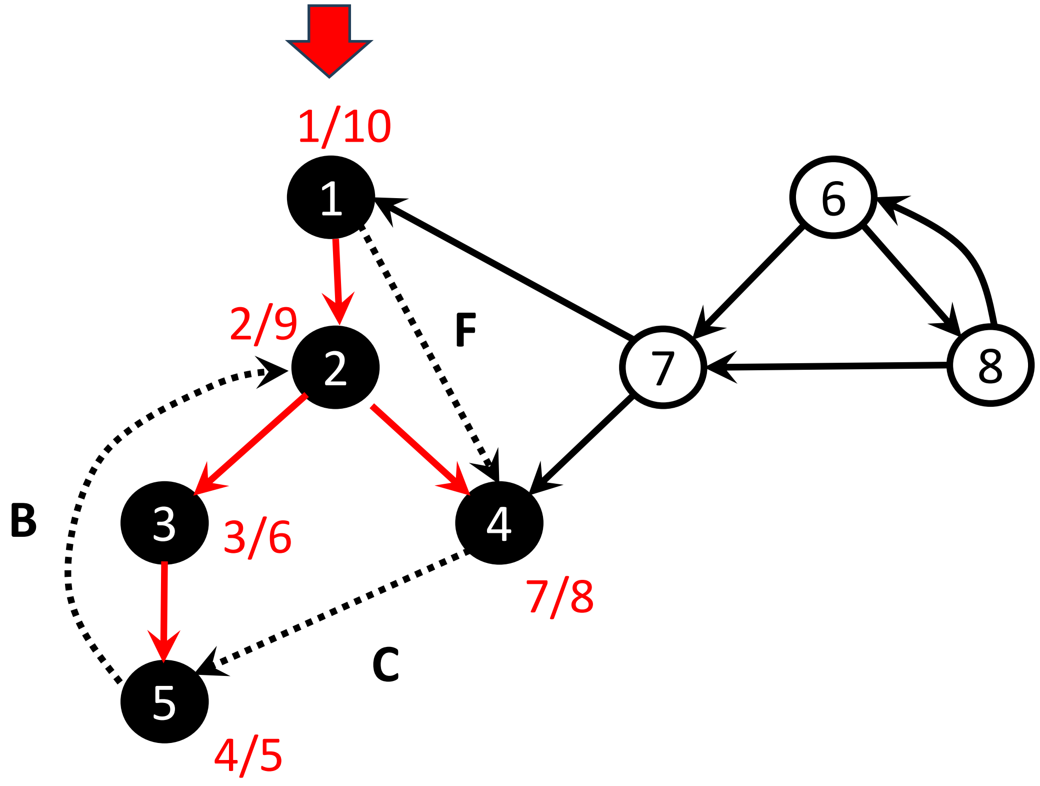
DFS: Live Poll 2

Label the numbered edges as ***Back***, ***Forward***, or ***Cross*** edges



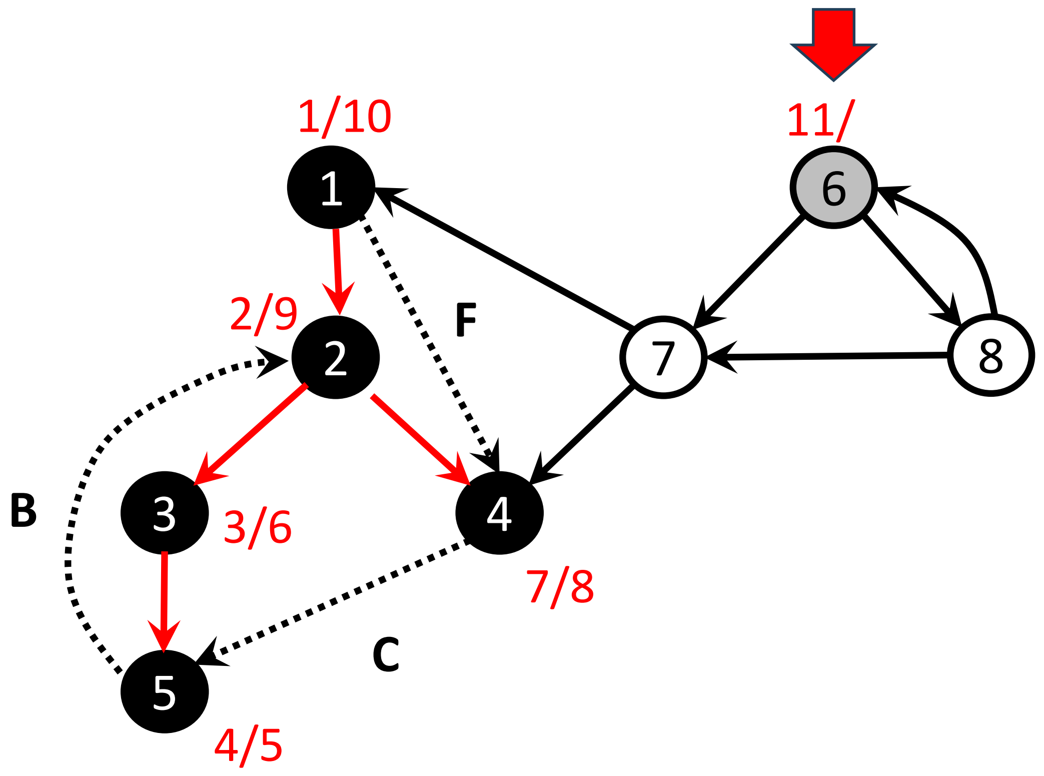
DFS: Live Poll 2

Label the numbered edges as ***Back***, ***Forward***, or ***Cross*** edges



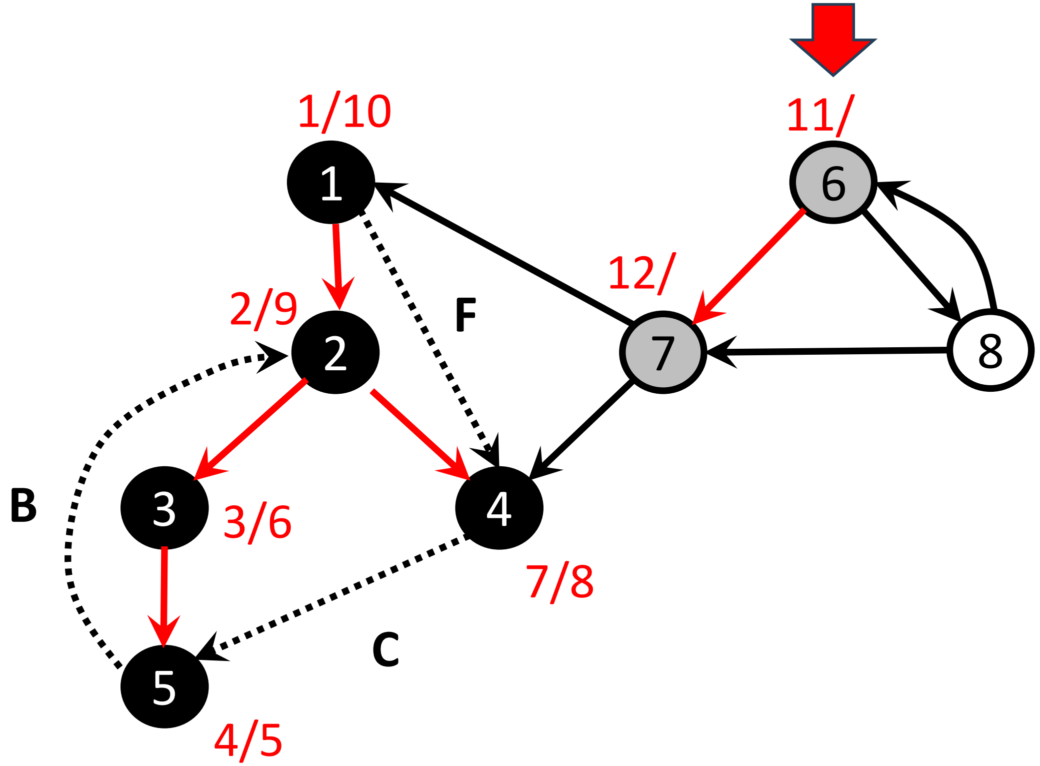
DFS: Live Poll 2

Label the numbered edges as ***Back***, ***Forward***, or ***Cross*** edges



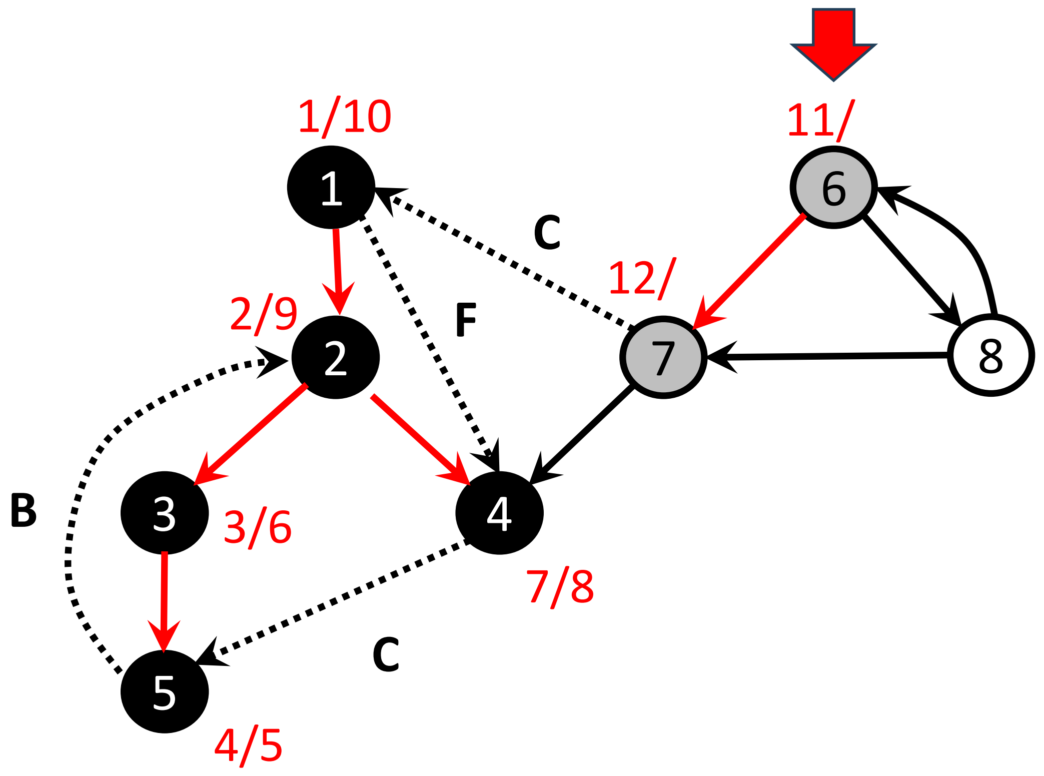
DFS: Live Poll 2

Label the numbered edges as ***Back***, ***Forward***, or ***Cross*** edges



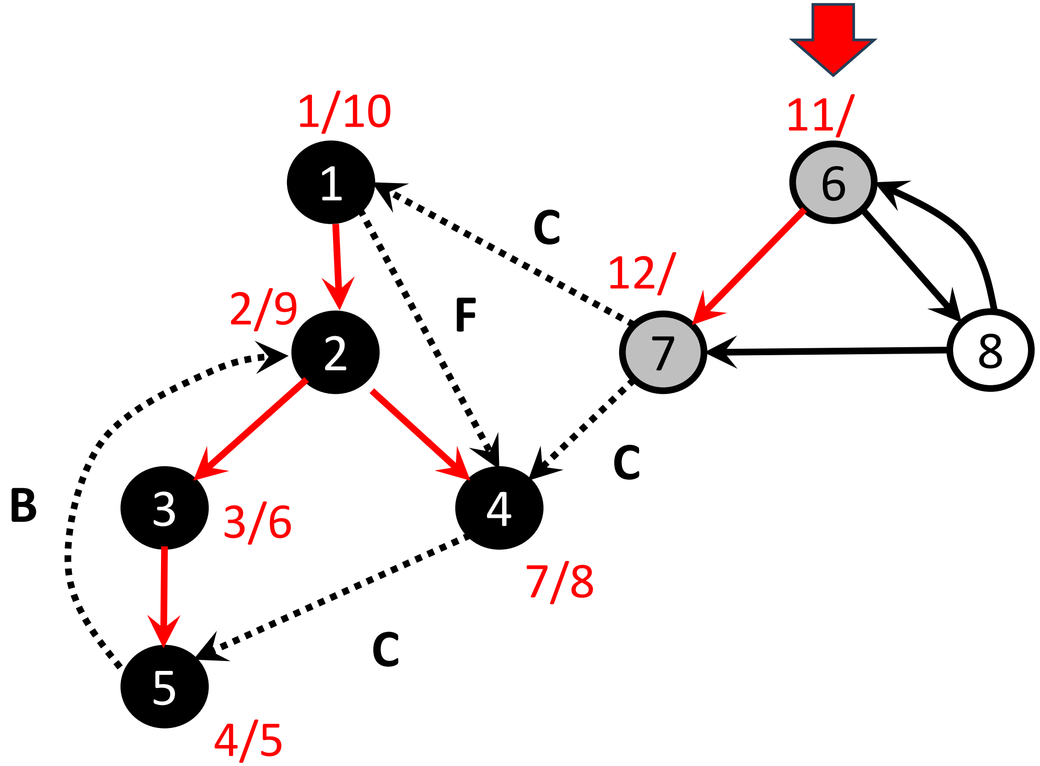
DFS: Live Poll 2

Label the numbered edges as ***Back***, ***Forward***, or ***Cross*** edges



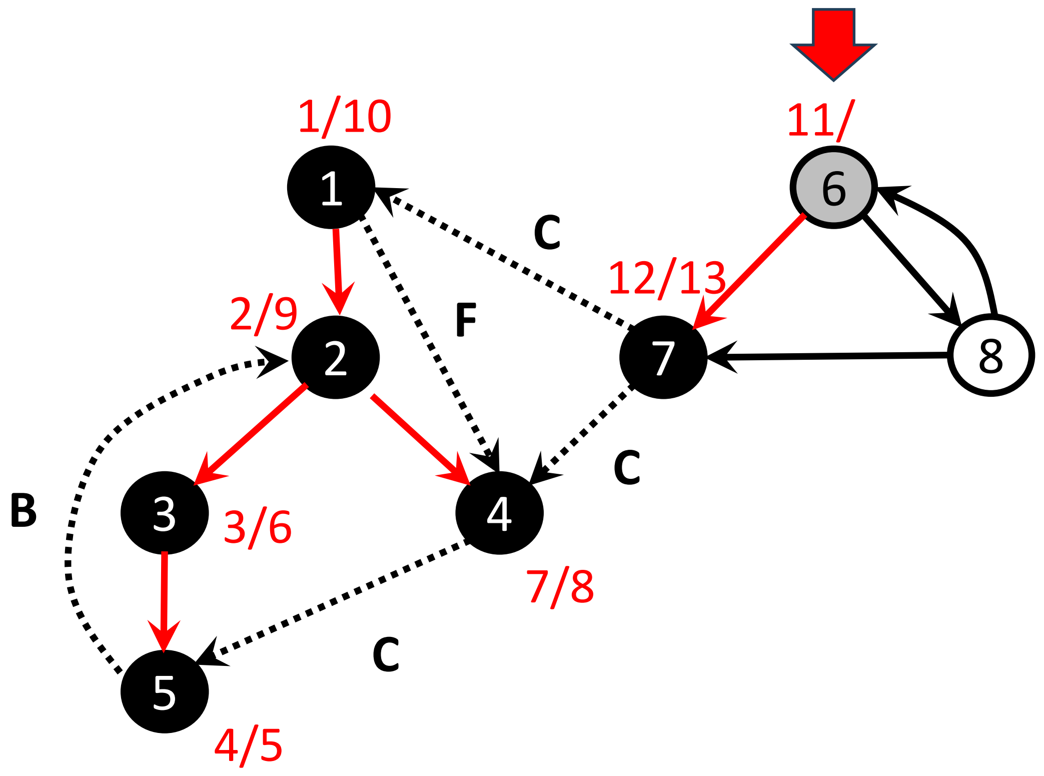
DFS: Live Poll 2

Label the numbered edges as ***Back***, ***Forward***, or ***Cross*** edges



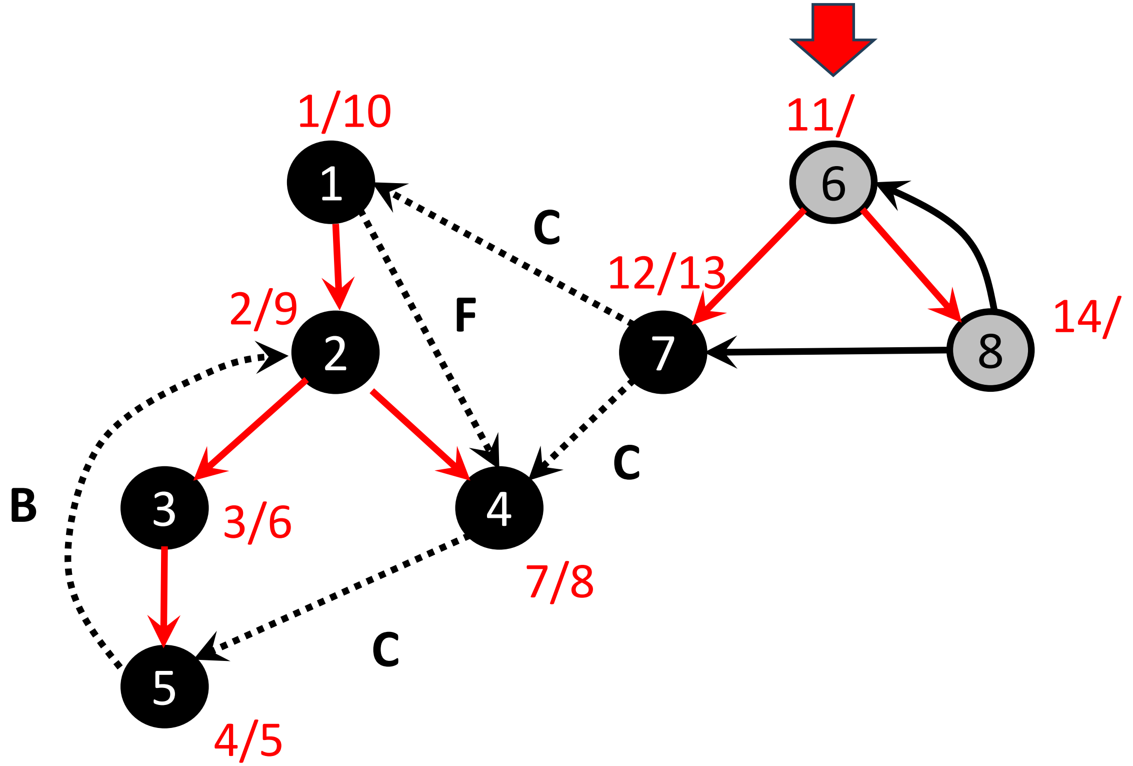
DFS: Live Poll 2

Label the numbered edges as ***Back***, ***Forward***, or ***Cross*** edges



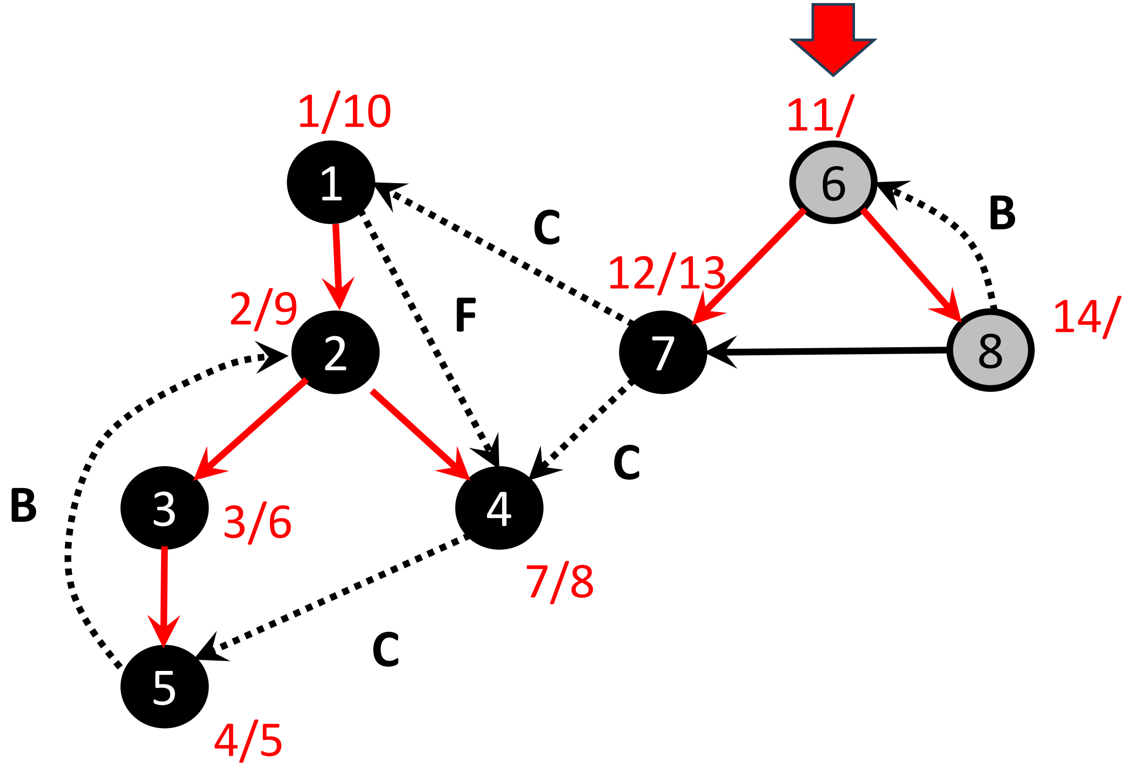
DFS: Live Poll 2

Label the numbered edges as ***Back***, ***Forward***, or ***Cross*** edges



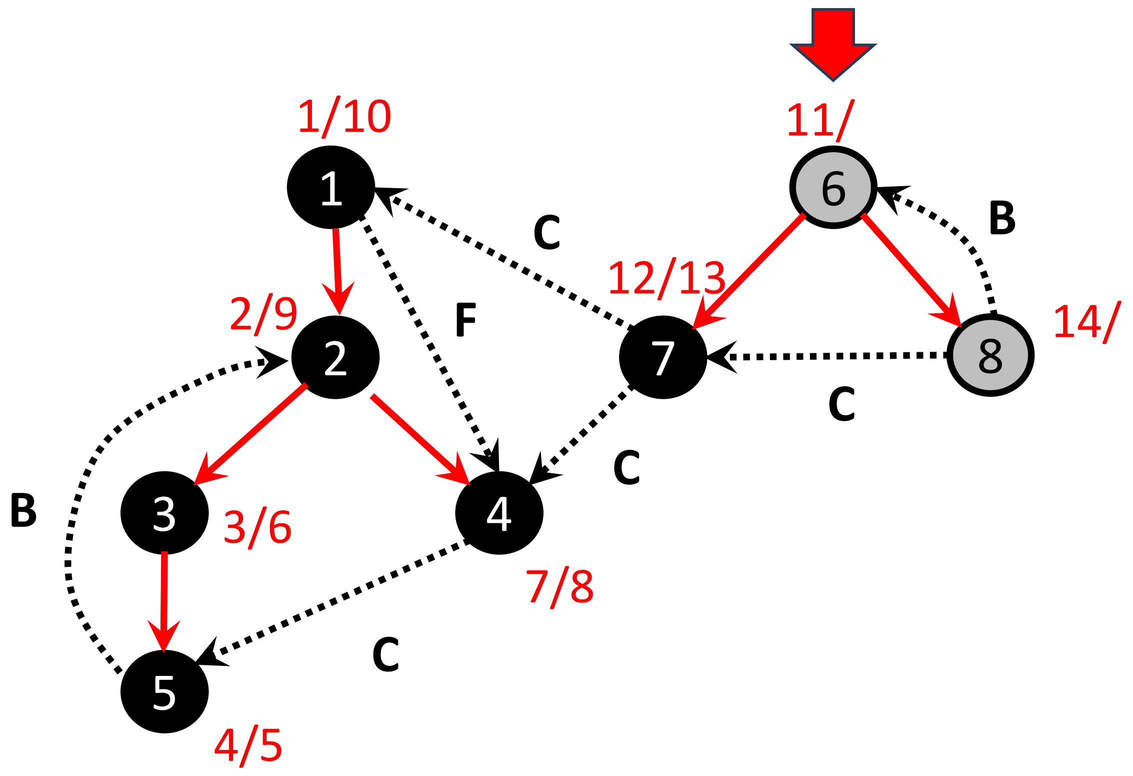
DFS: Live Poll 2

Label the numbered edges as ***Back***, ***Forward***, or ***Cross*** edges



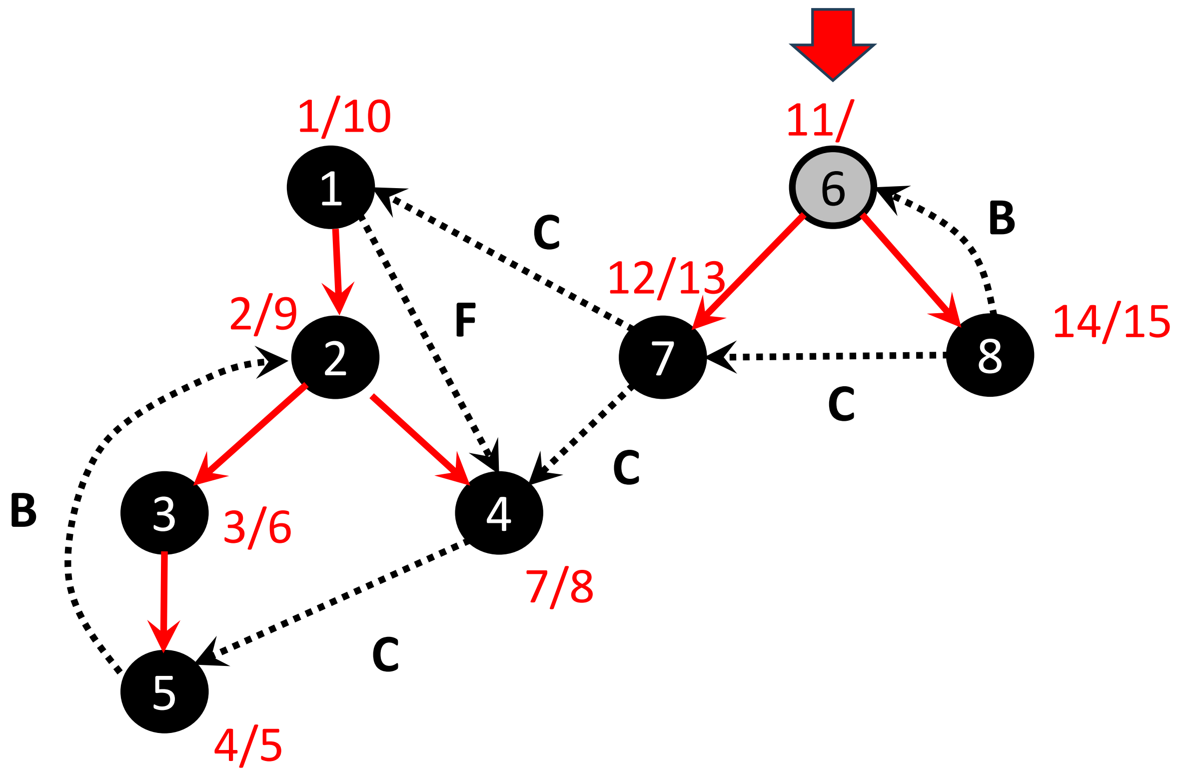
DFS: Live Poll 2

Label the numbered edges as ***Back***, ***Forward***, or ***Cross*** edges



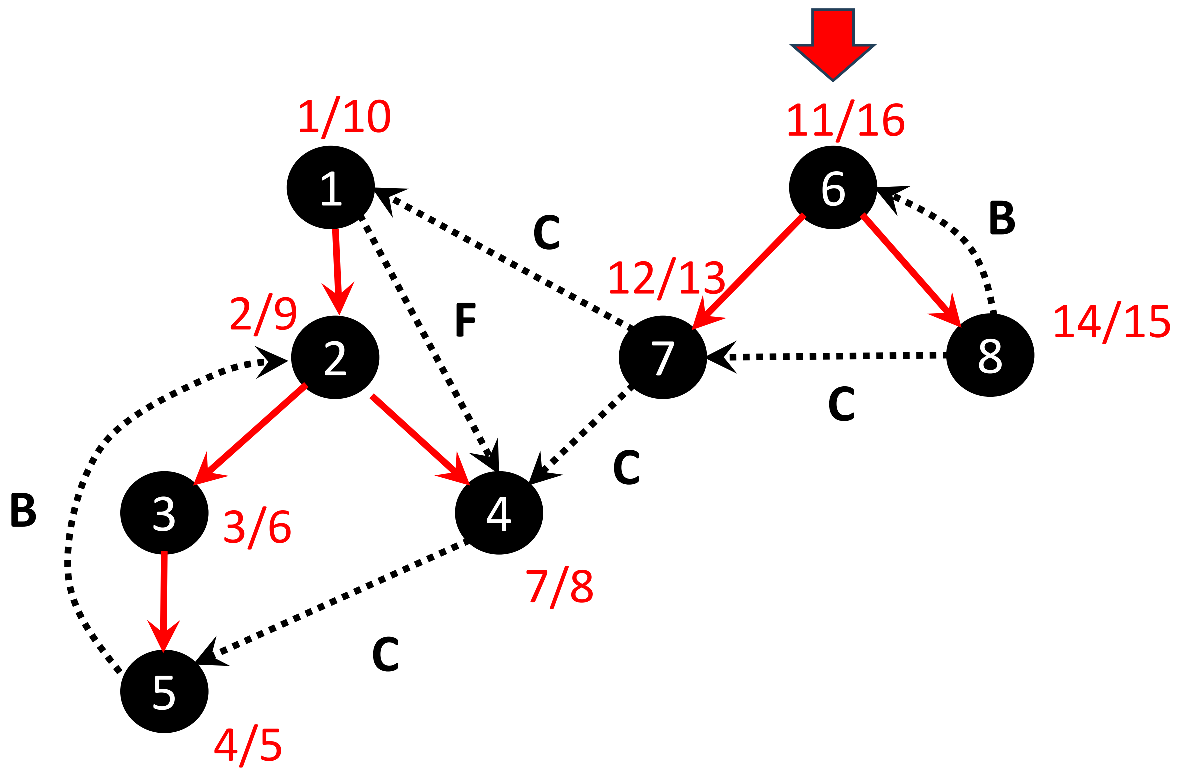
DFS: Live Poll 2

Label the numbered edges as ***Back***, ***Forward***, or ***Cross*** edges



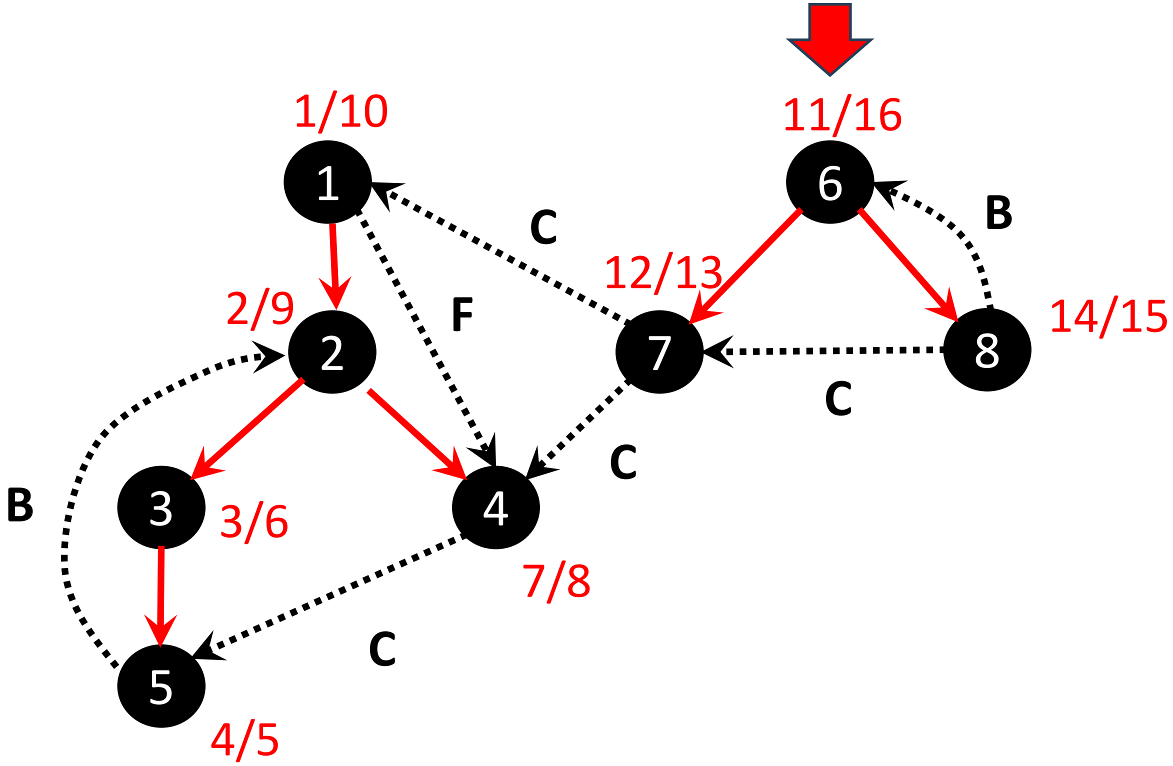
DFS: Live Poll 2

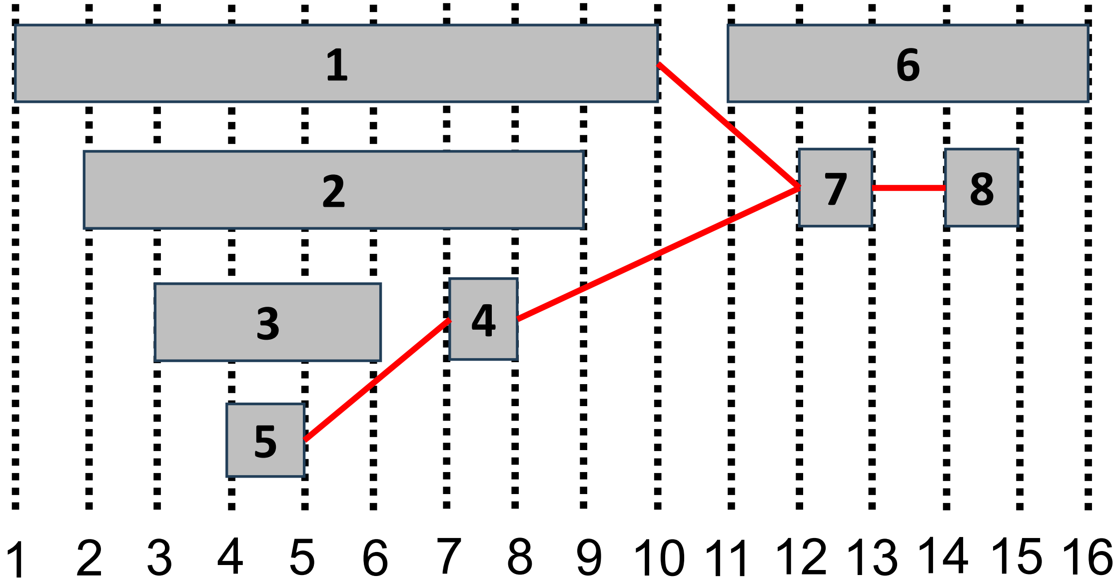
Label the numbered edges as ***Back***, ***Forward***, or ***Cross*** edges

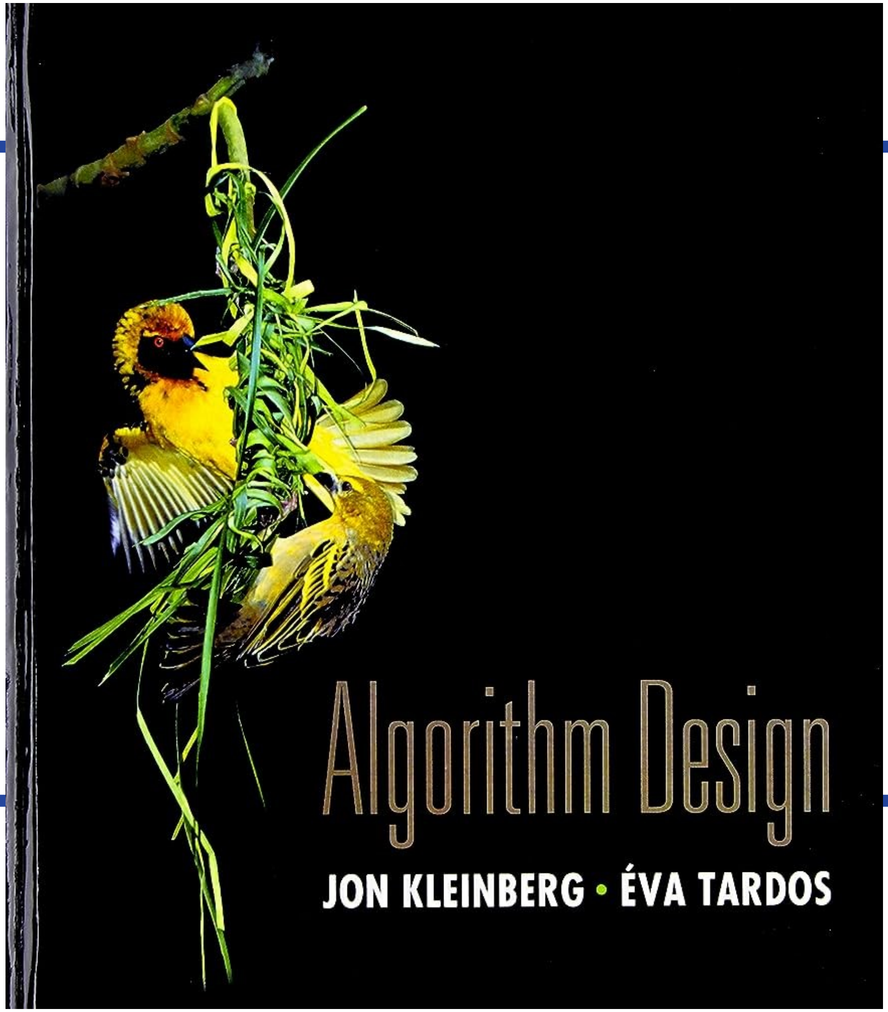


DFS: Live Poll 2

Label the numbered edges as ***Back***, ***Forward***, or ***Cross*** edges







Chapter 5:
Divide and Conquer

Divide-and-conquer

- Many algorithms are recursive in structure
  - Call themselves recursively one or more times to solve smaller sub-problems efficiently

- Divide-and-conquer paradigm – ***3 steps***
  1. **Divide** the problem into a number of sub-problems that are smaller instances of the original problem

  2. **Conquer** the sub-problems by solving them recursively. If the subproblem sizes are small enough, solve the subproblems in a straight-forward manner (in constant number of steps)

  3. **Combine** the solutions to the sub-problems into the solution to the original problem

Section 5.1:
Merge Sort

Divide-and-conquer

Sorting: Given an array of $\boldsymbol{n}$ elements, sort the array in ascending order

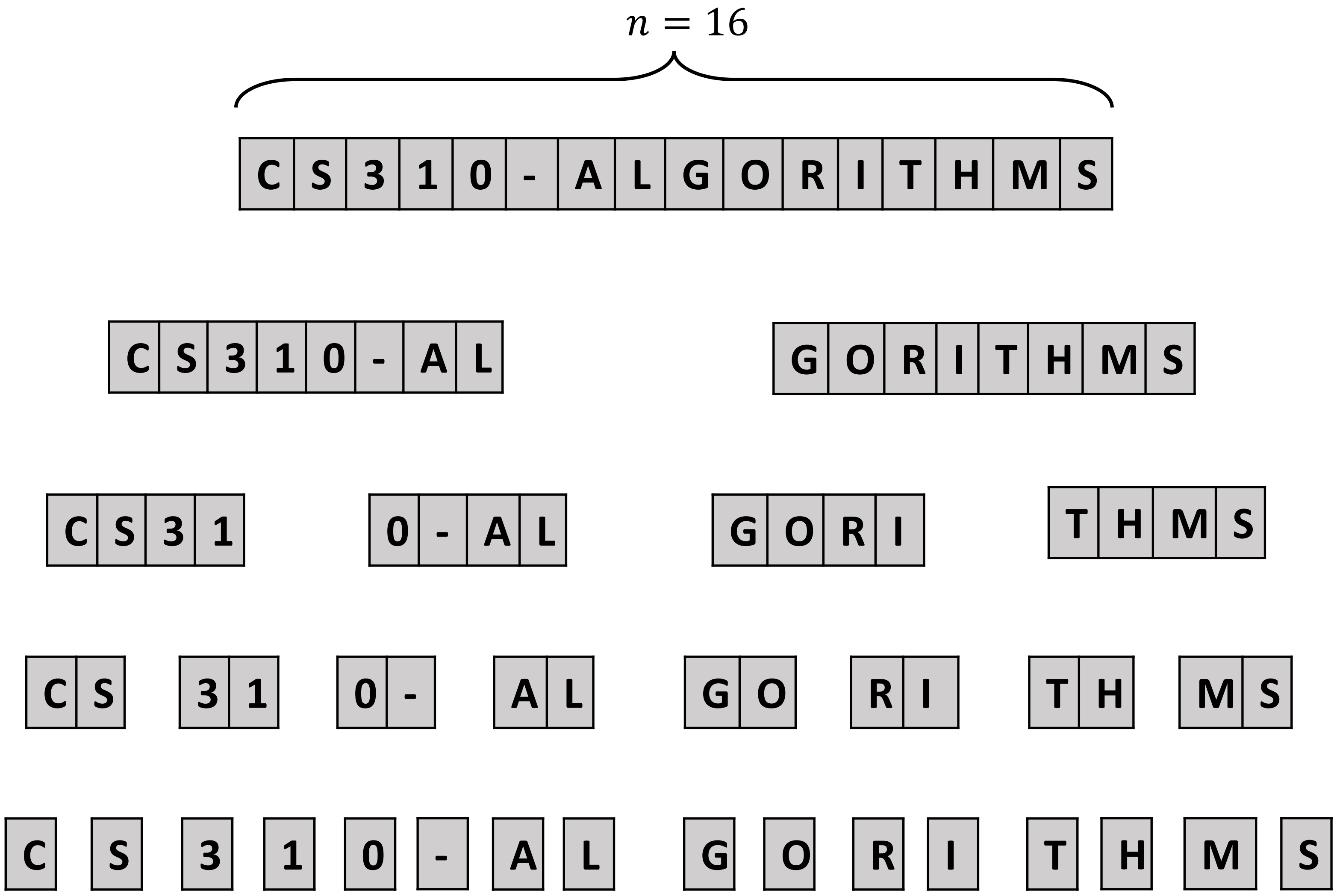
Merge Sort

1. **Divide:** Divide the $\boldsymbol{n}$ elements array into two subarrays of $\boldsymbol{n/2}$ elements each
2. **Conquer:** Sort the two subarrays recursively
3. **Combine:** Merge the two sorted subarrays to produce the sorted answer

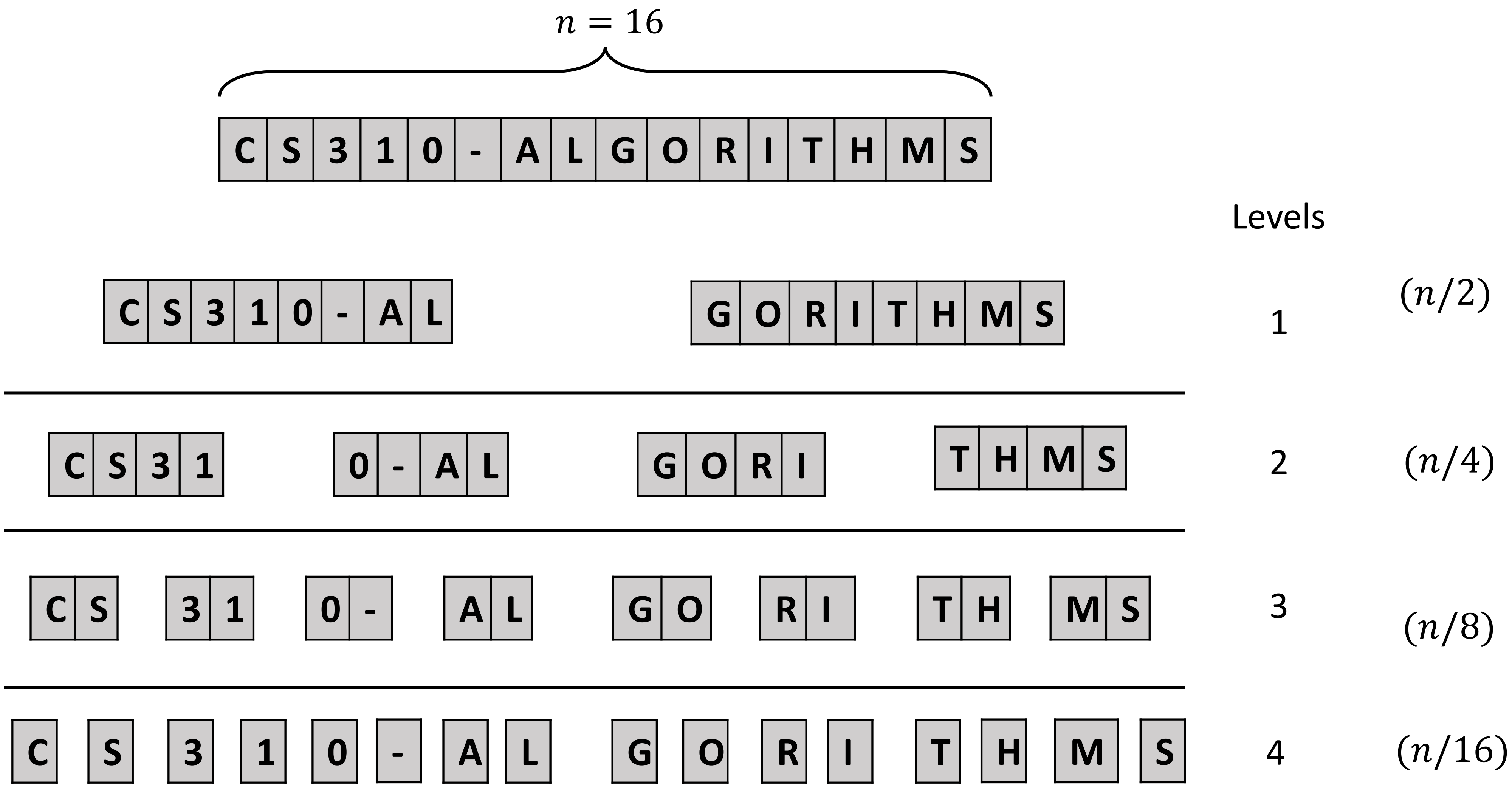
Mergesort

$MERGESORT(A)$

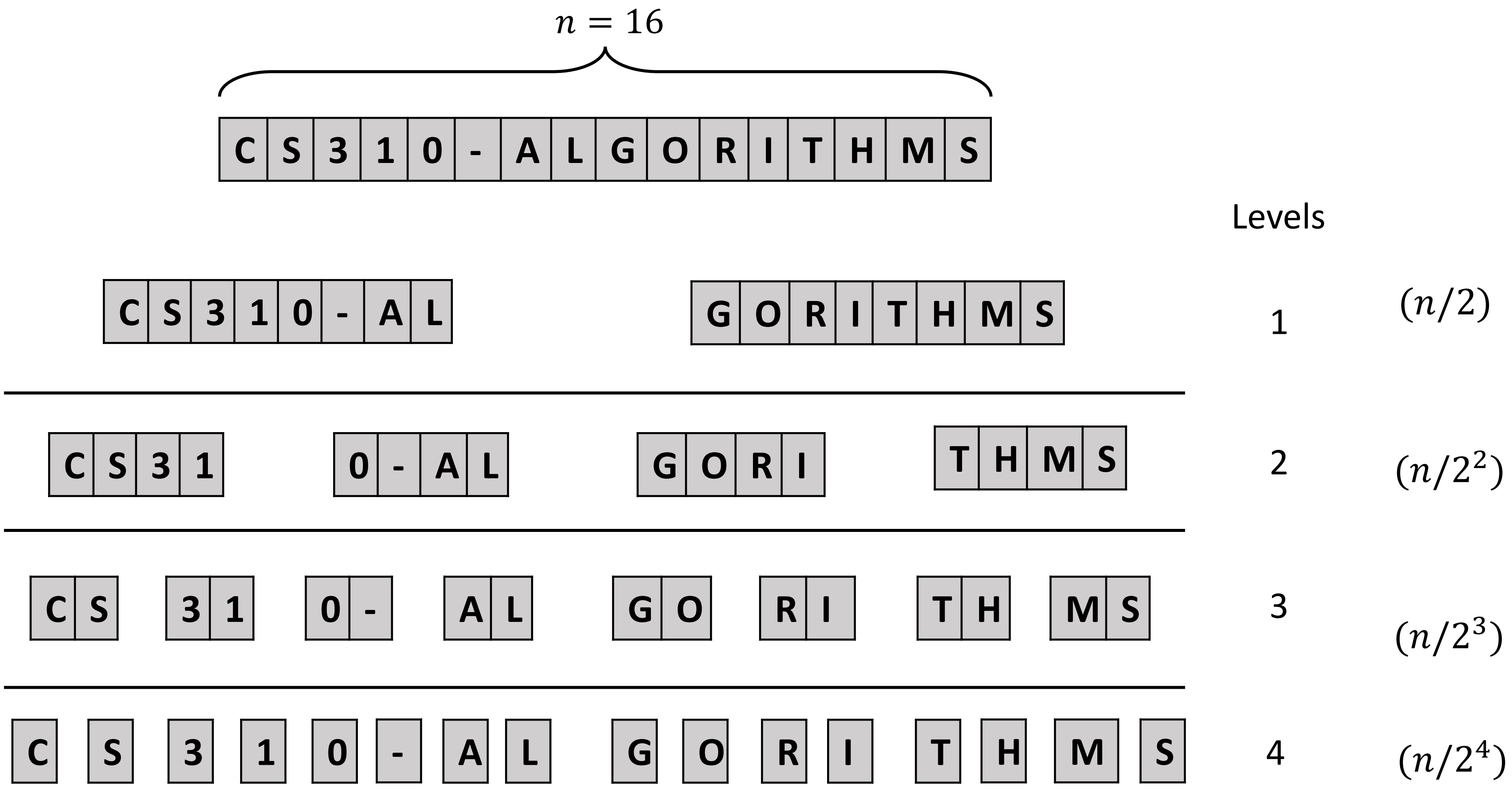
```
if (length(A) > 1)
    A_1 ← A[1 ⋯ ⌊n/2⌋ ]
    A_2 ← A[⌊n/2⌋ + 1 ⋯ n]
    A_1 ← MERGESORT(A_1)
    A_2 ← MERGESORT(A_2)
    A ← MERGE(A_1, A_2)
return A
```



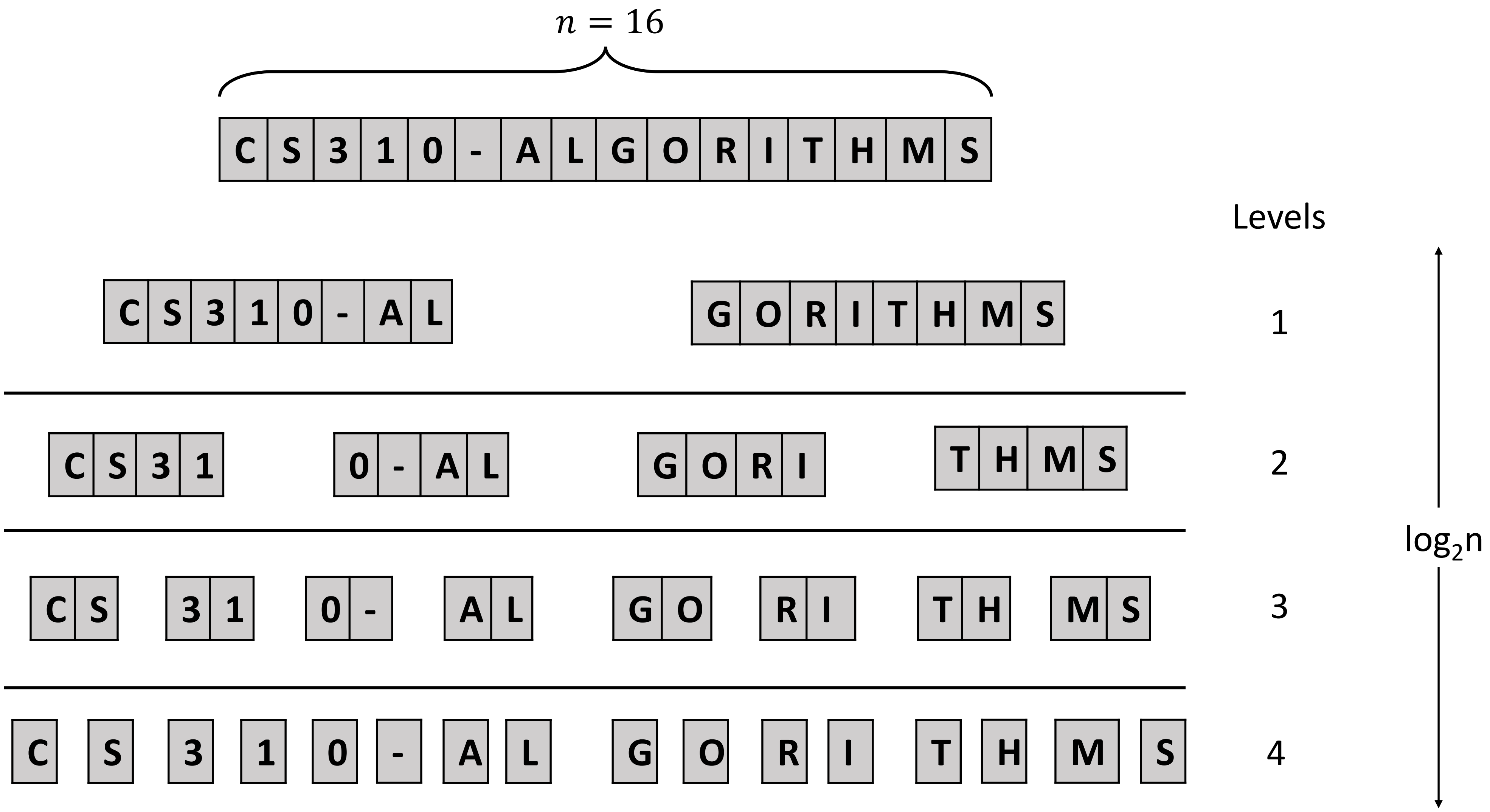
Mergesort


$n = 16$
C S 3 1 0 - A L G O R I T H M S
Levels
C S 3 1 0 - A L
G O R I T H M S
1
$(n/2)$
C S 3 1
0 - A L
G O R I
T H M S
2
$(n/4)$
C S
3 1
0 -
A L
G O
R I
T H
M S
3
$(n/8)$
C
S
3
1
0
-
A
L
G
O
R
I
T
H
M
S
4
$(n/16)$

Mergesort


$n = 16$
C S 3 1 0 - A L G O R I T H M S
Levels
C S 3 1 0 - A L
G O R I T H M S
1
$(n/2)$
C S 3 1
0 - A L
G O R I
T H M S
2
$(n/2^2)$
C S
3 1
0 -
A L
G O
R I
T H
M S
3
$(n/2^3)$
C
S
3
1
0
-
A
L
G
O
R
I
T
H
M
S
4
$(n/2^4)$

Mergesort

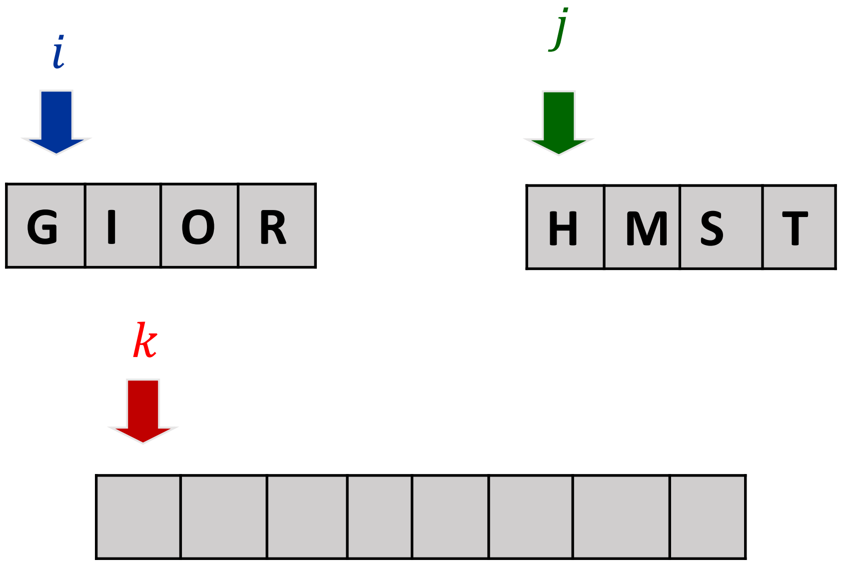


$n = 16$
C S 3 1 0 - A L G O R I T H M S
Levels
C S 3 1 0 - A L
G O R I T H M S
1
C S 3 1
0 - A L
G O R I
T H M S
2
$\log_2 n$
C S
3 1
0 -
A L
G O
R I
T H
M S
3
C S 3 1 0 - A L G O R I T H M S
4

Mergesort

$MERGE(A_1, A_2)$

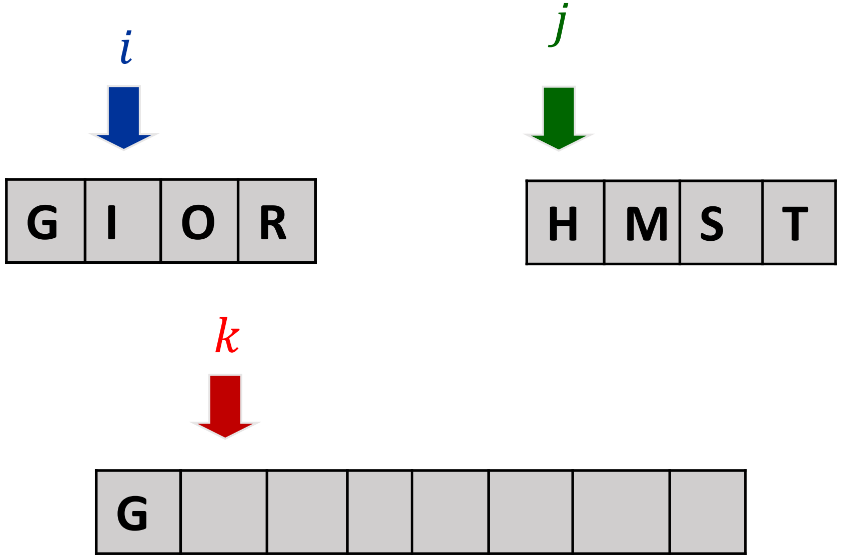
```
m ← length(A_1) + length(A_2)
i ← 1; j ← 1
for k = 1 to m
    if A_1[i] ≤ A_2[j]
        A[k] ← A_1[i]
        i ← i + 1
    else A[k] ← A_2[j]
        j ← j + 1
return A
```



Mergesort

$MERGE(A_1, A_2)$

$m \leftarrow length(A_1) + length(A_2)$
$i \leftarrow 1; j \leftarrow 1$
for $k = 1$ **to** m
    if $A_1[i] \leq A_2[j]$
        $A[k] \leftarrow A_1[i]$
        $i \leftarrow i + 1$
    else $A[k] \leftarrow A_2[j]$
        $j \leftarrow j + 1$
return A



Mergesort

$MERGE(A_1, A_2)$

$m \leftarrow length(A_1) + length(A_2)$

$i \leftarrow 1; j \leftarrow 1$

for $k = 1$ **to** m

  if $A_1[i] \leq A_2[j]$

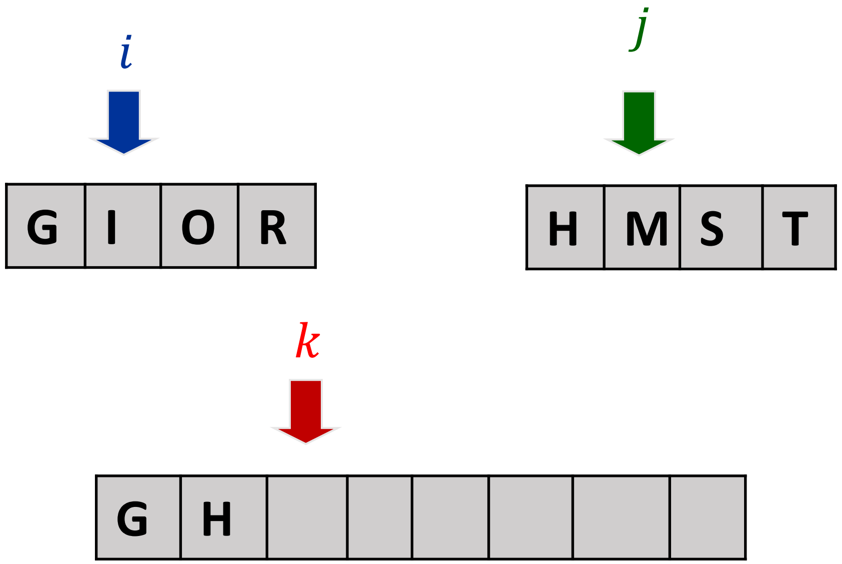
    $A[k] \leftarrow A_1[i]$

    $i \leftarrow i + 1$

  else $A[k] \leftarrow A_2[j]$

    $j \leftarrow j + 1$

return A



Mergesort

$MERGE(A_1, A_2)$

$m \leftarrow length(A_1) + length(A_2)$

$i \leftarrow 1; j \leftarrow 1$

for $k = 1$ **to** m

    if $A_1[i] \leq A_2[j]$

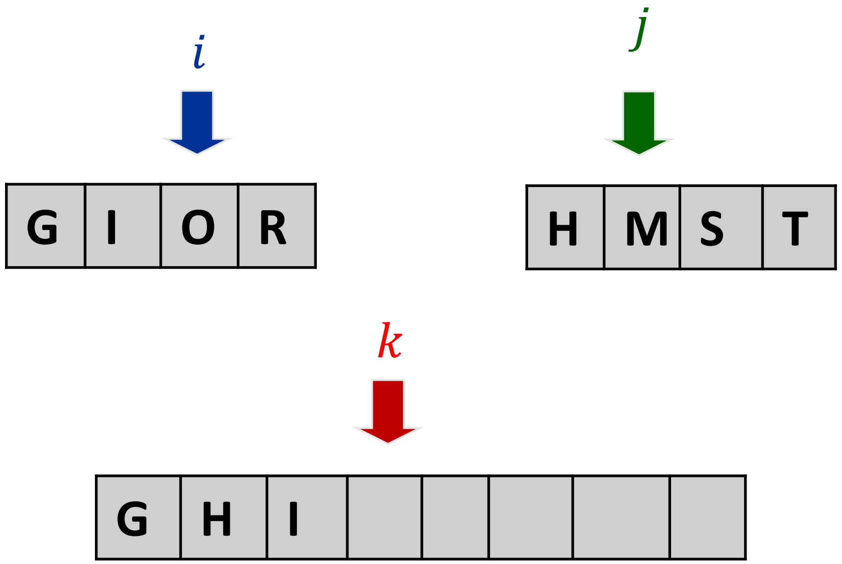
        $A[k] \leftarrow A_1[i]$

        $i \leftarrow i + 1$

    else $A[k] \leftarrow A_2[j]$

        $j \leftarrow j + 1$

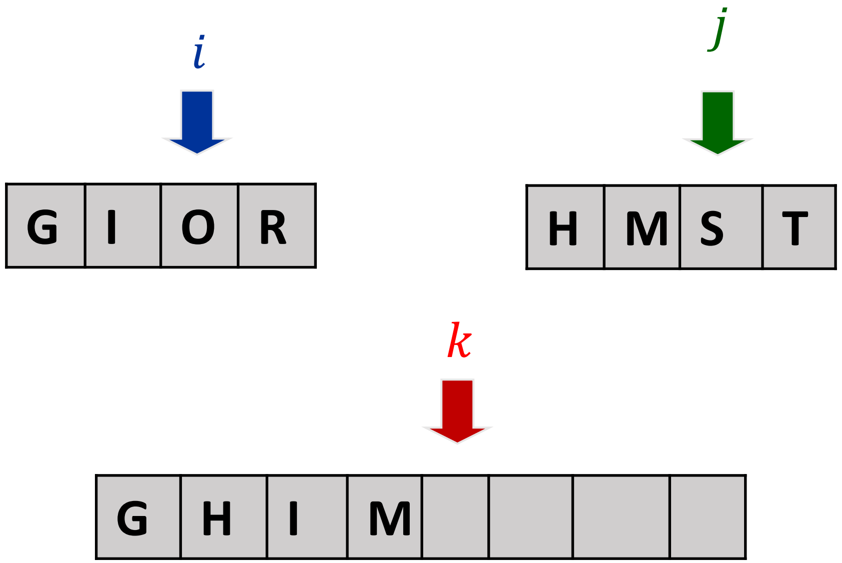
return A



Mergesort

$MERGE(A_1, A_2)$

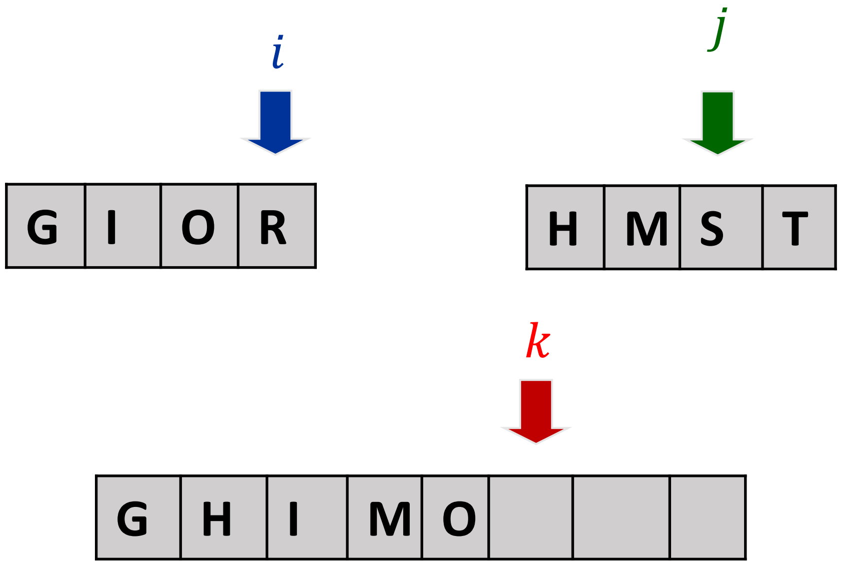
$m \leftarrow length(A_1) + length(A_2)$
$i \leftarrow 1; j \leftarrow 1$
for $k = 1$ **to** m
    if $A_1[i] \leq A_2[j]$
        $A[k] \leftarrow A_1[i]$
        $i \leftarrow i + 1$
    else $A[k] \leftarrow A_2[j]$
        $j \leftarrow j + 1$
return A



Mergesort

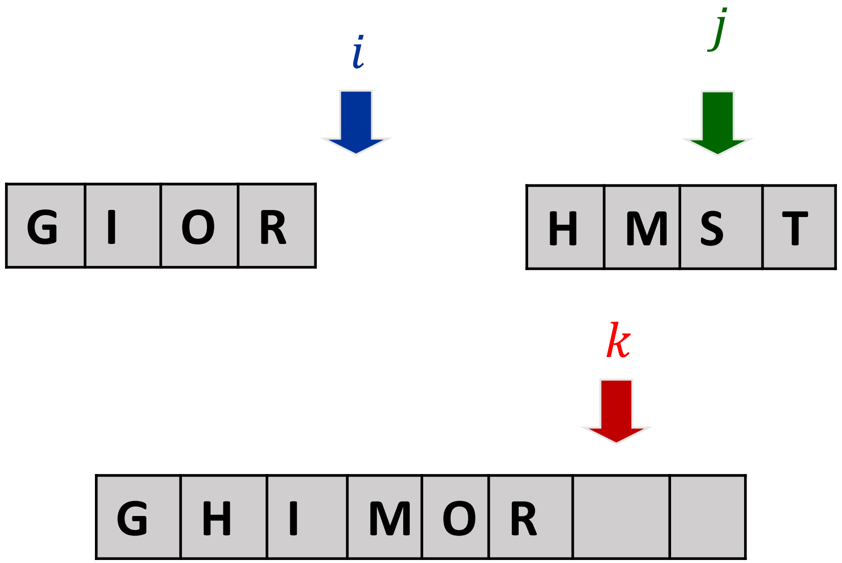
$MERGE(A_1, A_2)$

```
m ← length(A_1) + length(A_2)
i ← 1; j ← 1
for k = 1 to m
        if A_1[i] ≤ A_2[j]
                A[k] ← A_1[i]
                i ← i + 1
        else  A[k] ← A_2[j]
                j ← j + 1
return A
```



Mergesort

```
MERGE(A_1, A_2)
    m ← length(A_1) + length(A_2)
    i ← 1; j ← 1
    for k = 1 to m
        if A_1[i] ≤ A_2[j]
            A[k] ← A_1[i]
            i ← i + 1
        else  A[k] ← A_2[j]
            j ← j + 1
    return A
```



Mergesort

$MERGE(A_1, A_2)$

$m \leftarrow length(A_1) + length(A_2)$

$i \leftarrow 1; j \leftarrow 1$

for $k = 1$ **to** m

  if $A_1[i] \leq A_2[j]$

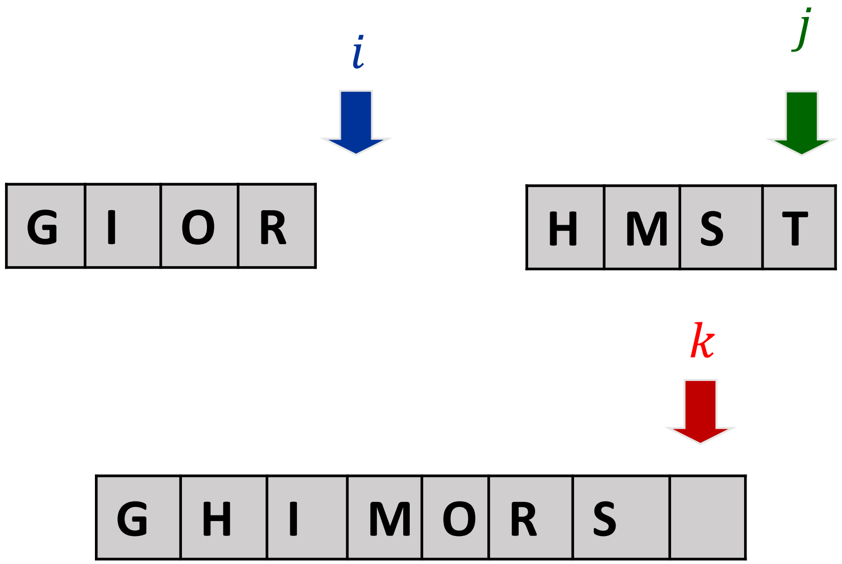
    $A[k] \leftarrow A_1[i]$

    $i \leftarrow i + 1$

  else $A[k] \leftarrow A_2[j]$

    $j \leftarrow j + 1$

return A



Mergesort

$MERGE(A_1, A_2)$

$m \leftarrow length(A_1) + length(A_2)$

$i \leftarrow 1; j \leftarrow 1$

for $k = 1$ **to** m

  if $A_1[i] \leq A_2[j]$

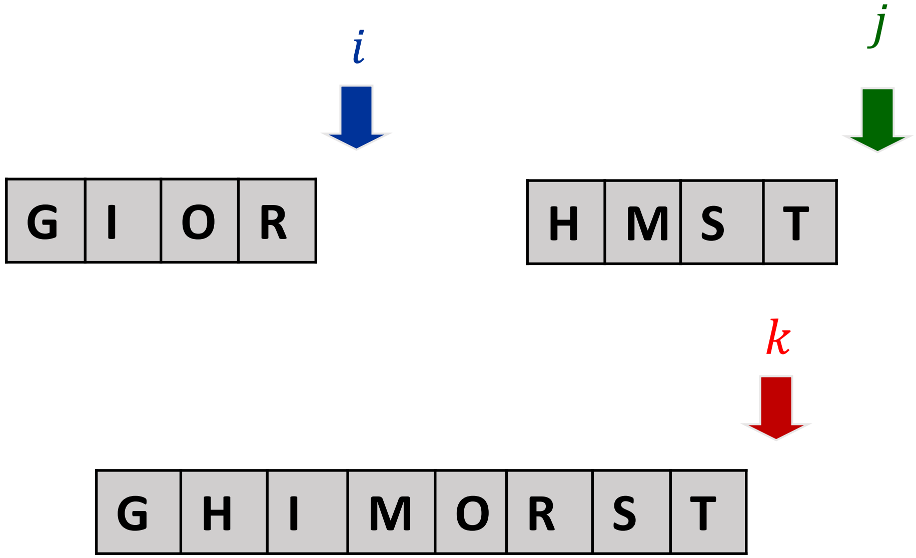
    $A[k] \leftarrow A_1[i]$

    $i \leftarrow i + 1$

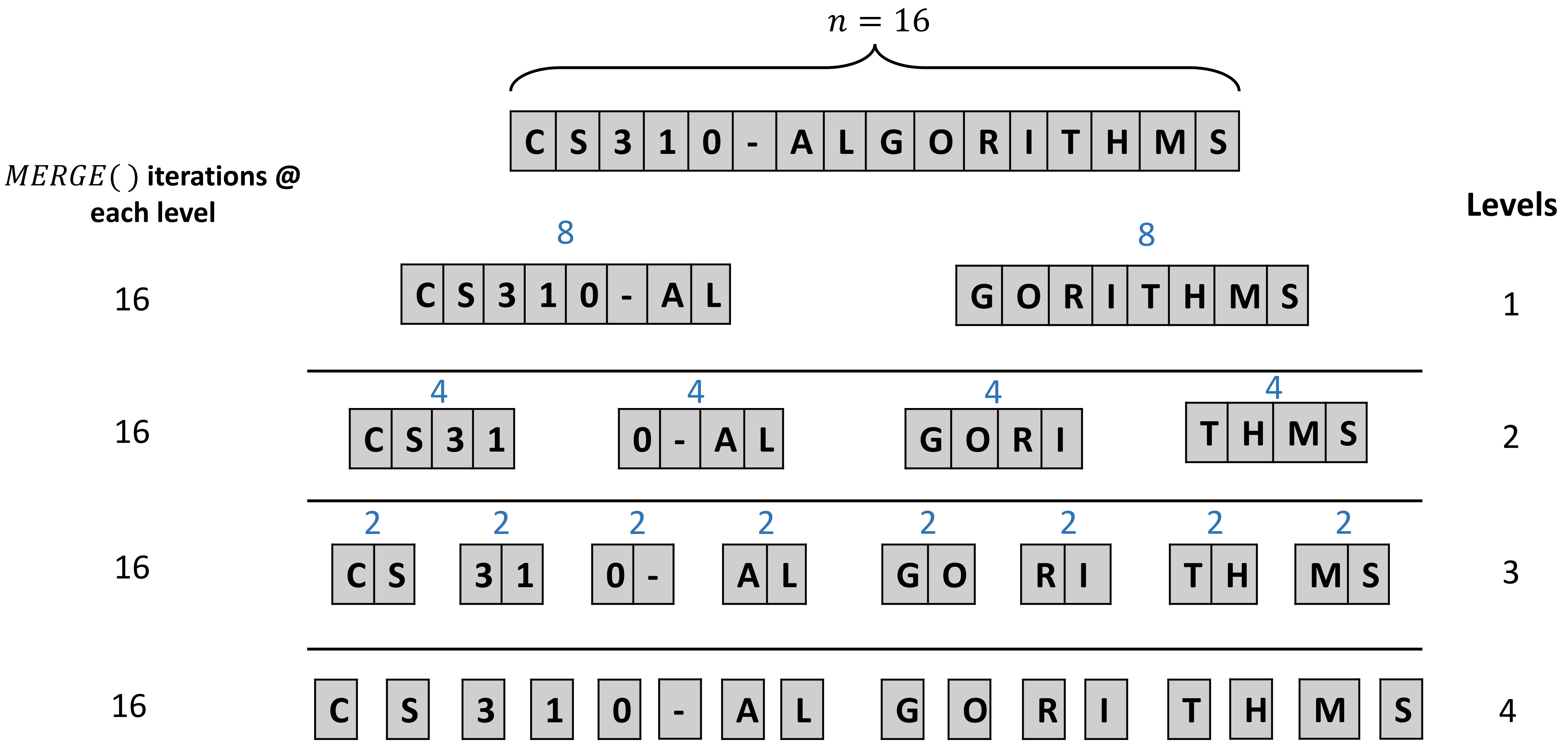
  else $A[k] \leftarrow A_2[j]$

    $j \leftarrow j + 1$

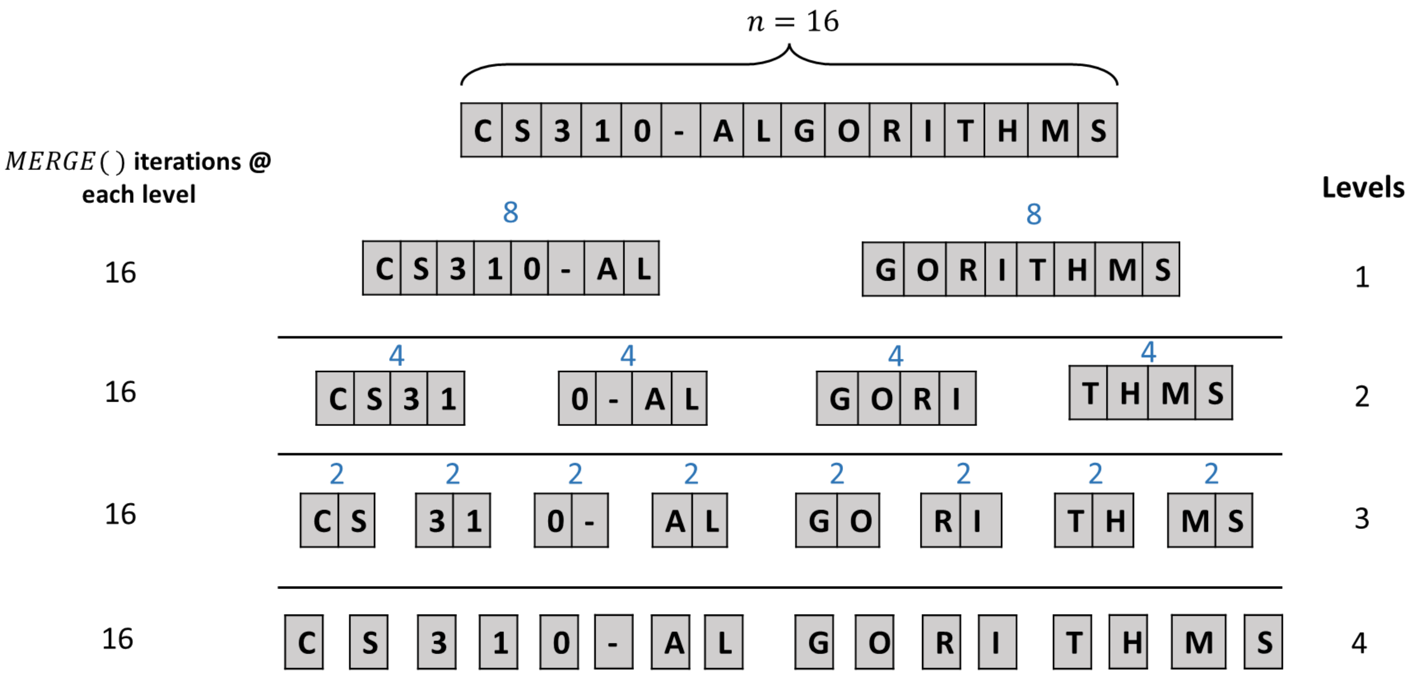
return A



Mergesort (Recursion Tree Method)



Mergesort (Recursion Tree Method)



Time complexity will be:

$MERGE(\,)$ **iterations @ each level**	**X**	**Levels**
n	**X**	$\log_2 n$

$n \log_2 n$

Mergesort: Proof of correctness [induction]

Proposition. Mergesort sorts any list of n elements.

Pf. [by strong induction on n]

- Base case: $n = 1$.
- Inductive hypothesis: assume true for $1, 2, \ldots, n-1$.
- By inductive hypothesis, mergesort sorts both left and right halves.
- ***Merging operation*** combines two sorted lists into a sorted whole.

Background on **"Proof by Induction"**

Base Case:

- The base case serves as the foundation for the induction.
- You start by proving that the statement is true for a specific value, usually the smallest value **n=0** or **n=1**.

Inductive Step:

- The inductive step is where you assume that the statement holds for some arbitrary value **k** where **k<n** (this assumption is called the "***inductive hypothesis***").

Slide credit: Kevin Wayne.

Mergsort: Recurrence Relation

A	L	G	O	R	I	T	H	M	S

T(n)

A	L	G	O	R

I	T	H	M	S

divide O(1)

T(n/2) **T(n/2)**

A	G	L	O	R

H	I	M	S	T

sort 2T(n/2)

A	G	H	I	L	M	O	R	S	T

merge O(n)

Slide credit: Kevin Wayne.

Mergsort: Recurrence Relation

$MERGESORT(A)$

if $(length(A) > 1)$

$\quad A_1 \leftarrow A[1 \cdots \lfloor n/2 \rfloor]$

$\quad A_2 \leftarrow A[\lfloor n/2 \rfloor + 1 \cdots n]$

$\quad A_1 \leftarrow MERGESORT(A_1)$

$\quad A_2 \leftarrow MERGESORT(A_2)$

$\quad \mathrm{A} \leftarrow MERGE(A_1, A_2)$

return A

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

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

Mergsort: Recurrence Relation

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

Any algorithm satisfying this recurrence equation is bounded by $O(n \log_2 n)$, when $n > 1$

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

Proof by induction

- Proposition. If $T(n)$ satisfies the following recurrence, then $T(n) = n \log_2 n$.

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 2\,T(n/2) + n & \text{if } n > 1 \end{cases}$$

assuming n is a power of 2

- Pf. [by induction on n]
  - **Base case:** when $n = 1$, $T(1) = 0 = n \log_2 n$.
  - **Inductive hypothesis:** assume $T(n) = n \log_2 n$.
  - **Goal:** show that $T(2n) = 2n \log_2 (2n)$.

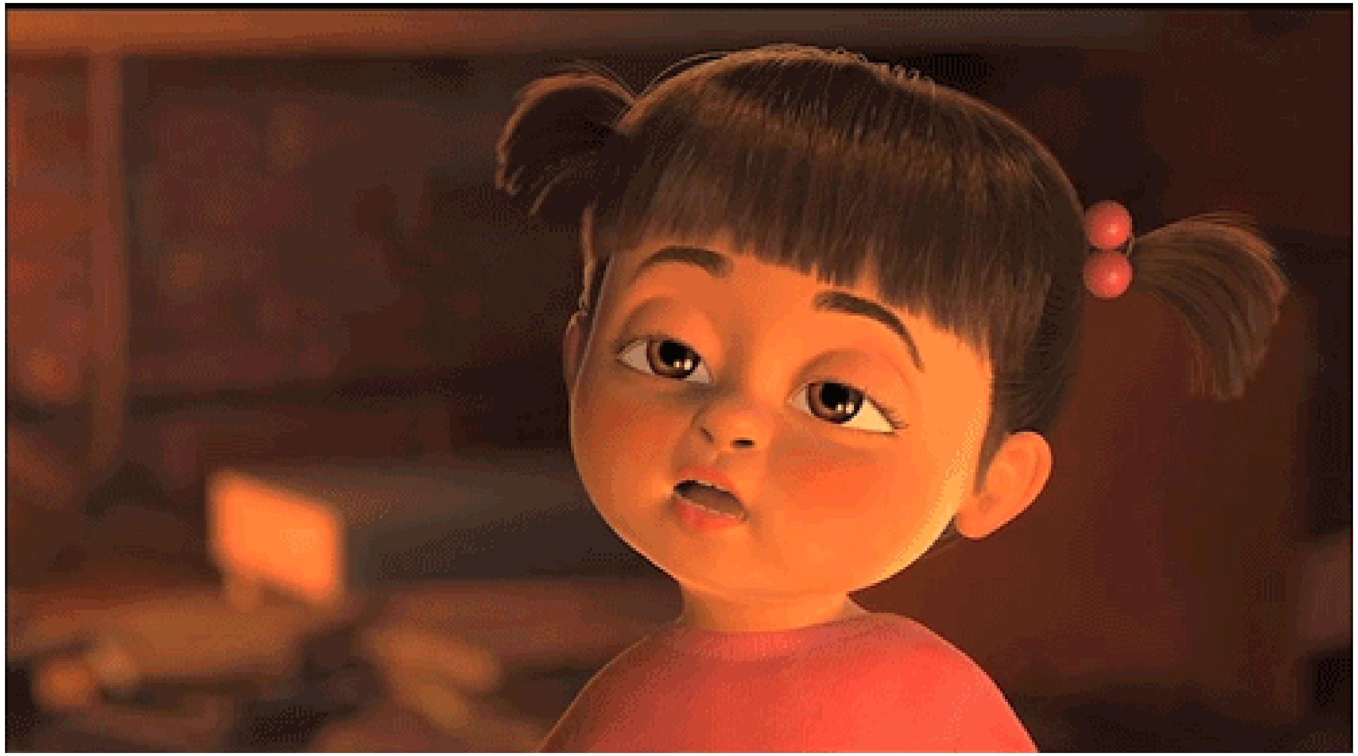
recurrence

inductive hypothesis

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

Slide credit: Kevin Wayne.

Thanks a lot



If you are taking a Nap, **wake up**.......Lecture Over