Computer Programming II

Ming-Feng Tsai (Victor Tsai)

Dept. of Computer Science
National Chengchi University

Graph Basics

Description of Graphs

- Graphs are some of the most flexible data structures in computing
- Most other data structures can be represented as graphs
- Graphs can be used to model problems defined in terms of relationships or connections between objects
    - Objects may be tangible entities such as nodes in a network or islands in a river

Description of Graphs

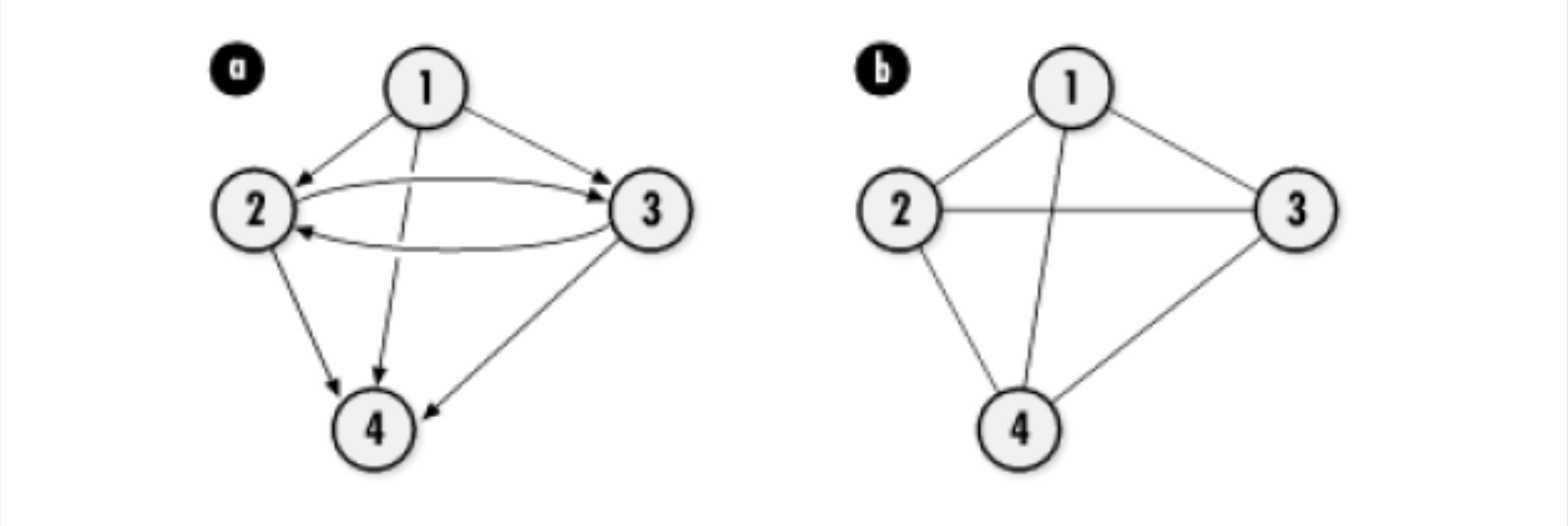
- Related topics about graphs
  - Graphs
    - How to model problems defined in terms of relations or connections between objects
  - Search methods
    - Techniques for visiting the vertices of a graph in a specific order: BFS and DFS

Description of Graphs

- Some applications of graphs
  - Graph algorithms
  - Counting network hops
  - Topological sorting
  - Graph coloring
  - Hamiltonian-cycle problems
  - Clique problems
  - PageRank

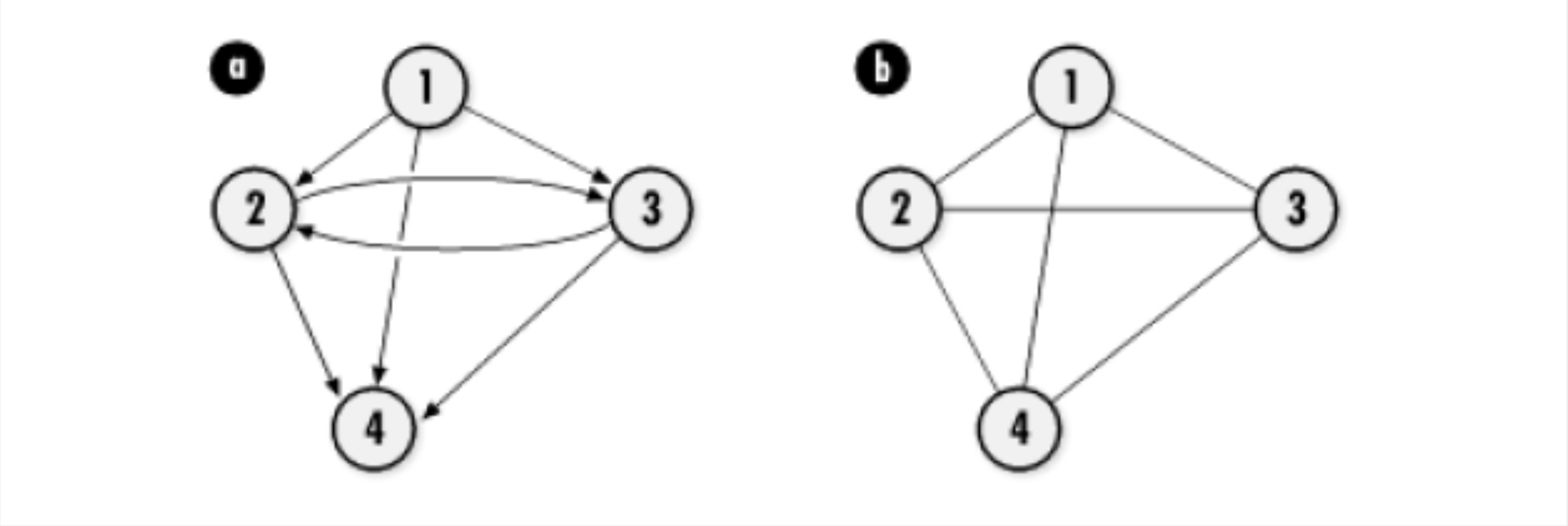
Description of Graphs

- Graphs are composed of two types of elements
  - vertices: objects
  - edges: relationships between the objects
- Graphs may be either directed or undirected



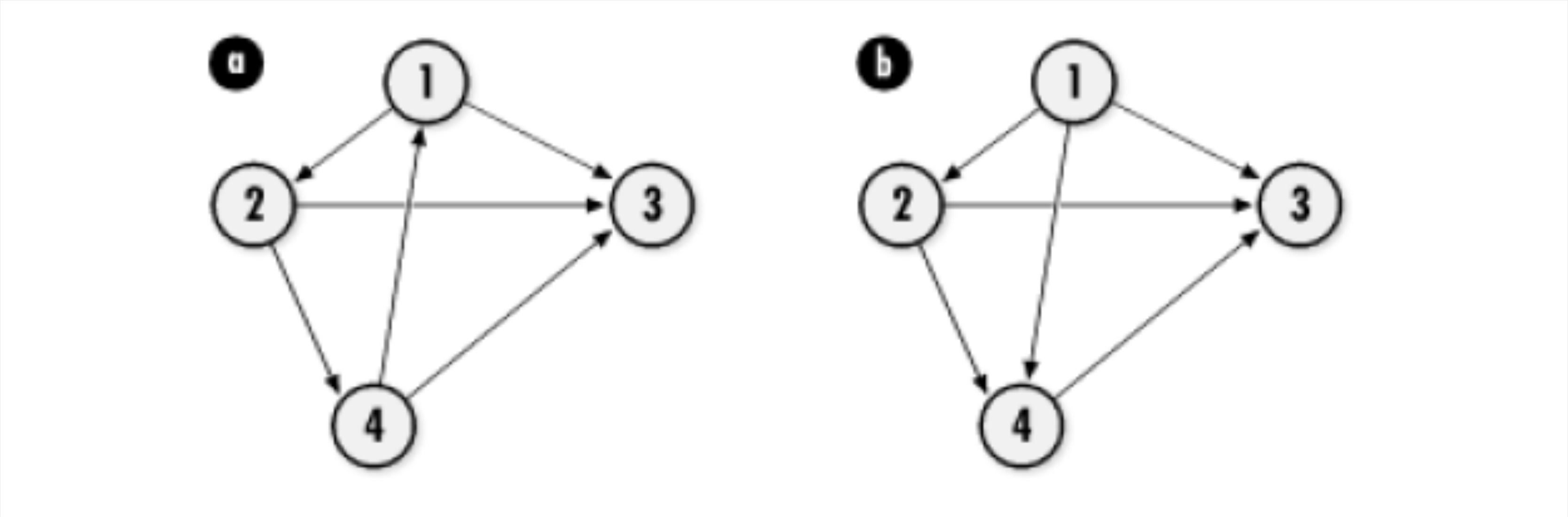
Description of Graphs

- Formally, a graph is a pair G = (V,E)
  - (a). Directed Graph: V = {1, 2, 3, 4} and E = {(1,2), (1,3), (1,4), (2,3), (2,4), (3,2), (3,4)}
  - (b). Undirected Graph: V = {1, 2, 3, 4} and E = {(1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3,4)}



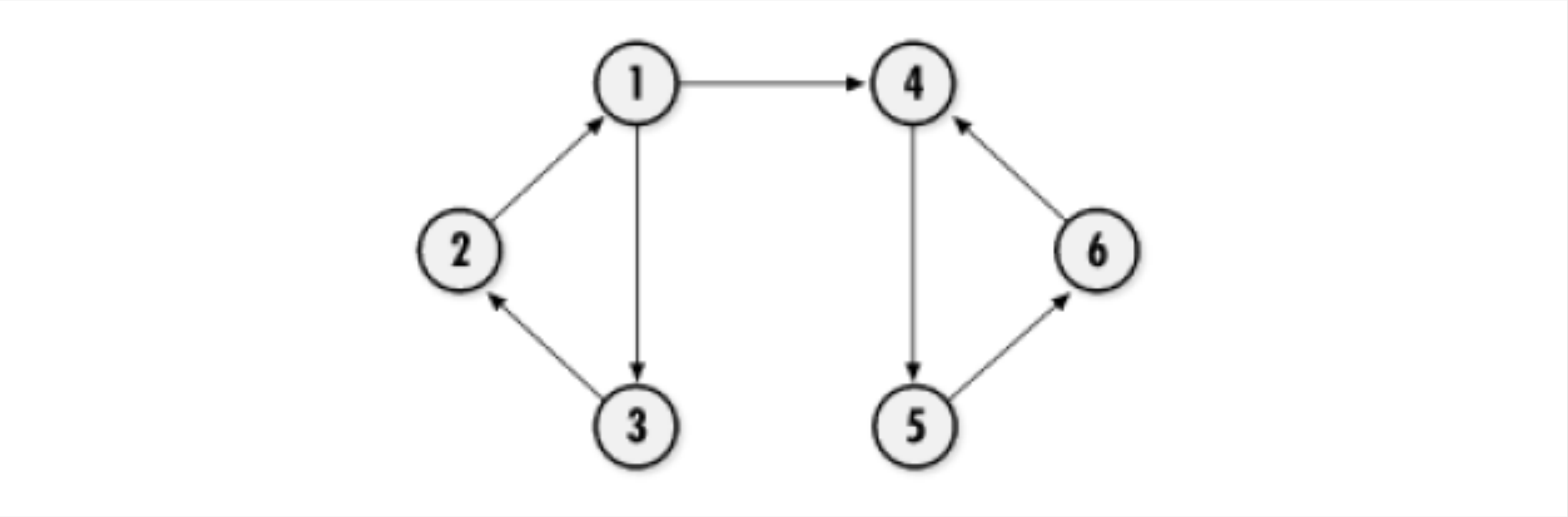
Description of Graphs

- A cycle is a path that includes the same vertex two or more times
  - (a). a directed graph containing the cycle {1, 2, 4, 1}
  - (b). a directed acyclic graph, or dag



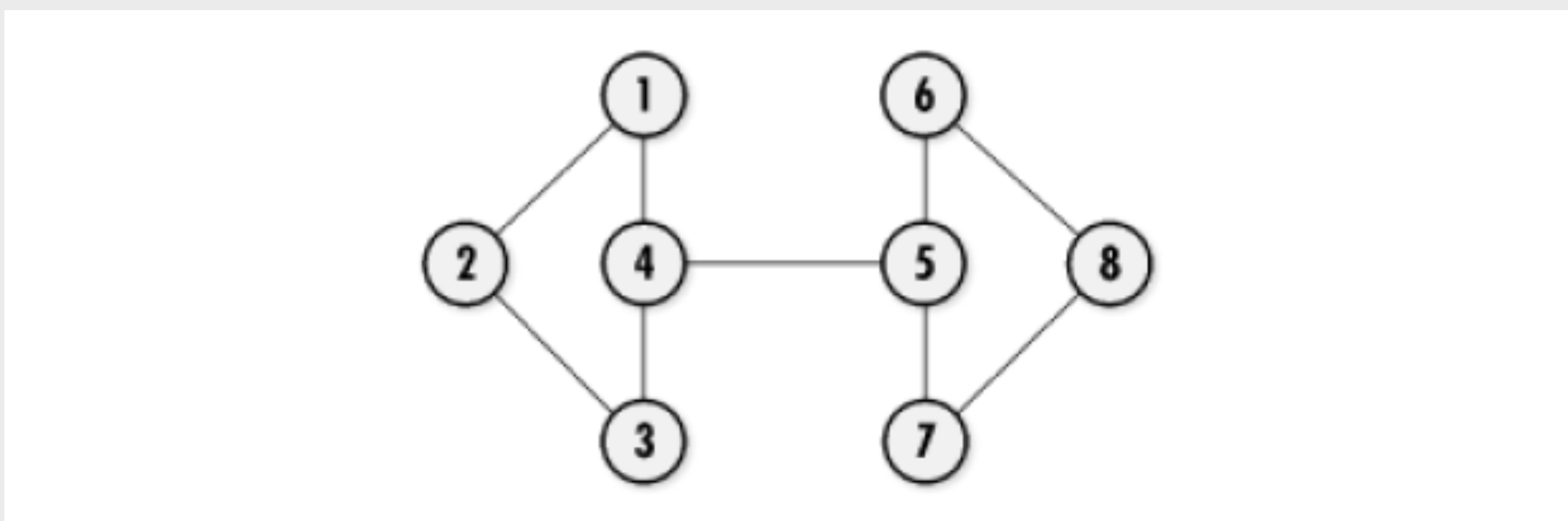
Description of Graphs

- An undirected graph is connected if every vertex is reachable from each other by following some path. If this is true in a directed graph, we say the graph is strongly connected
- A directed graph with two strongly connected components: {1, 2, 3} and {4, 5, 6}



Description of Graphs

- An undirected graph with articulation points 4 and 5, and the bridge (4, 5)

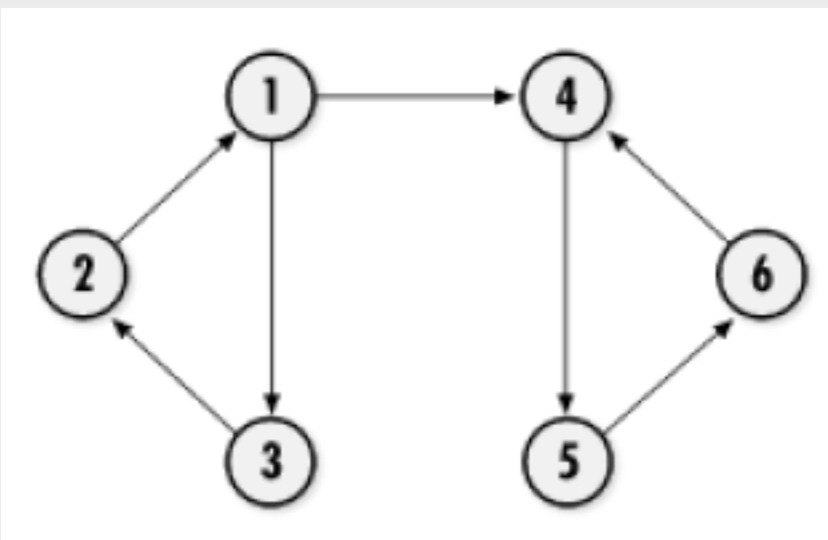


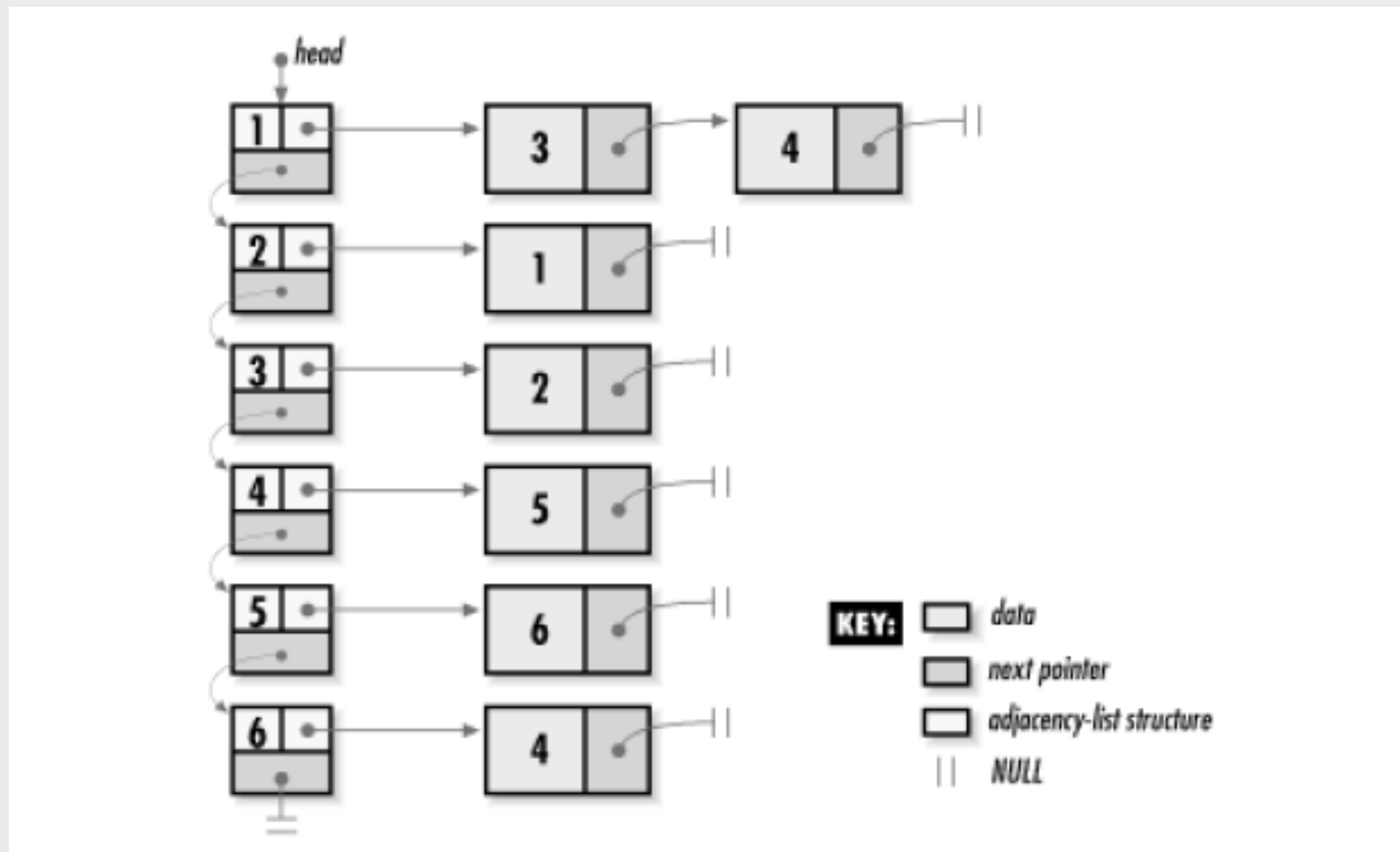
Description of Graphs

- The most common way to represent a graph in a computer is using an adjacency-list representation
- This consists of a linked list of adjacency-list structures.
- Each structure in the list contains two members: a vertex and a list of vertices adjacent to the vertex
- Adjacency-list representations require O (VE) space.

Description of Graphs

- Adjacency-list representation

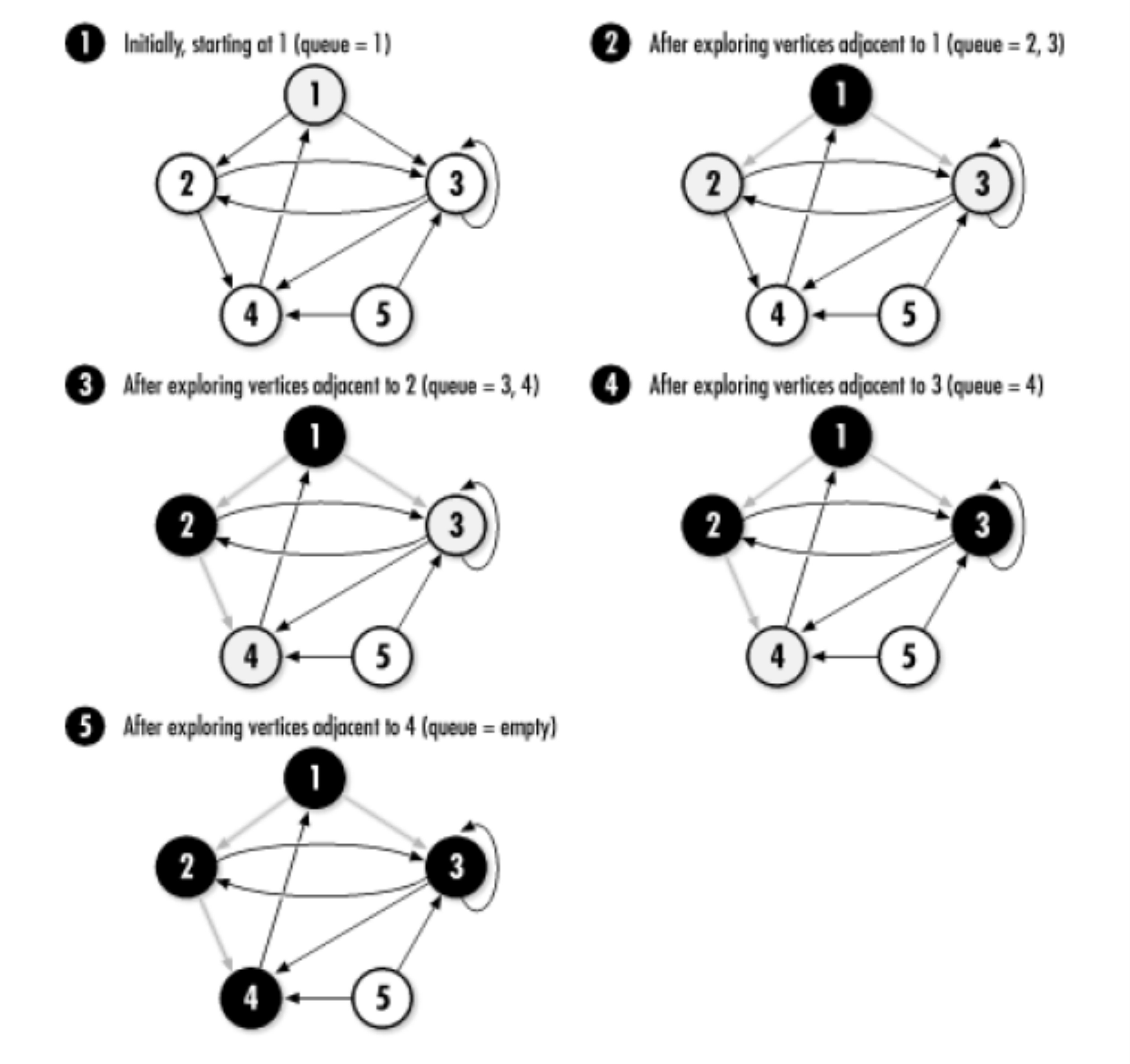




Description of Graphs

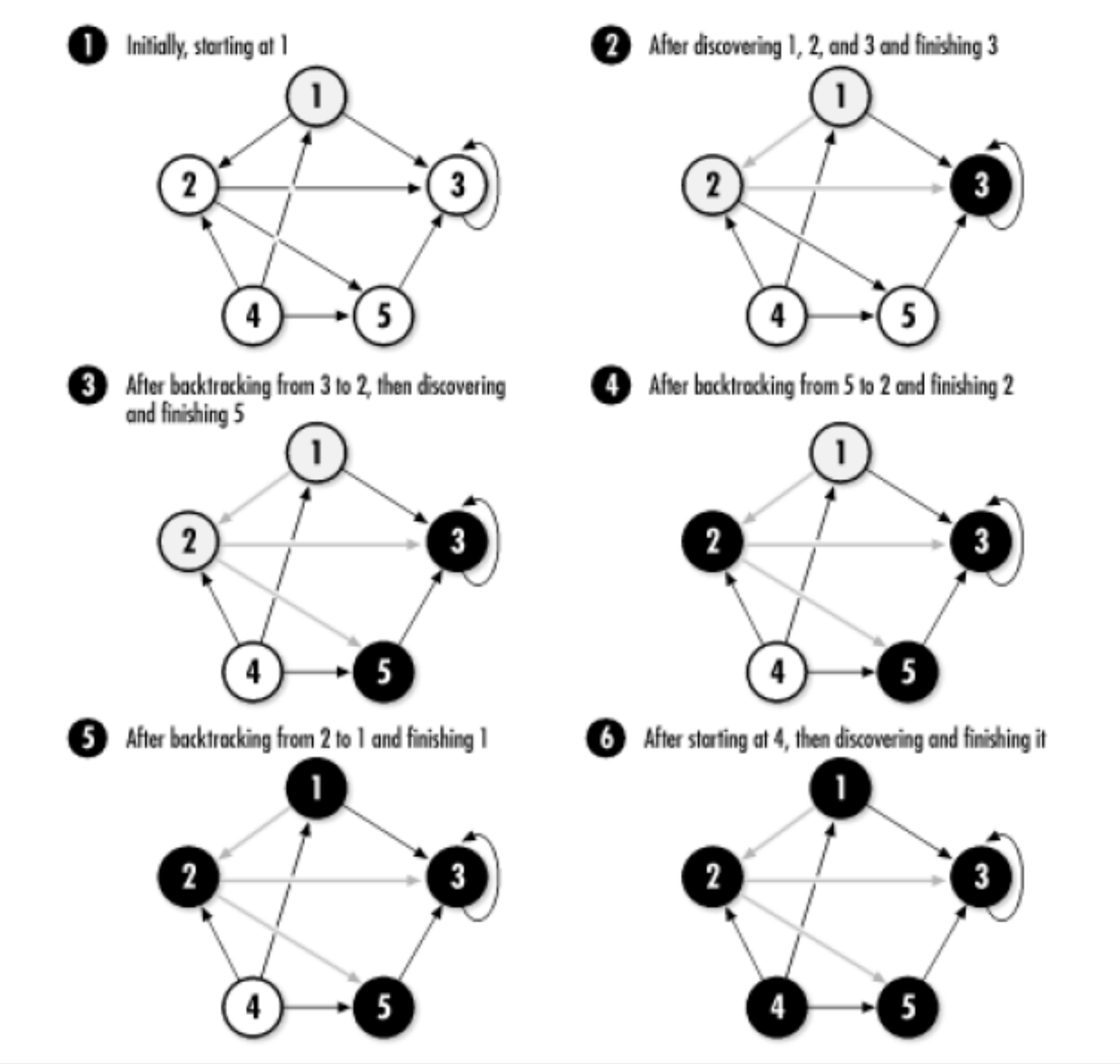
- Search Methods
  - BFS: breadth-first search
    - finding minimum spanning trees and shortest paths
  - DFS: depth-first search
    - cycle detection and topological sorting

BFS



1 Initially, starting at 1 (queue = 1)
2 After exploring vertices adjacent to 1 (queue = 2, 3)
3 After exploring vertices adjacent to 2 (queue = 3, 4)
4 After exploring vertices adjacent to 3 (queue = 4)
5 After exploring vertices adjacent to 4 (queue = empty)

DFS



1 Initially, starting at 1
2 After discovering 1, 2, and 3 and finishing 3
3 After backtracking from 3 to 2, then discovering and finishing 5
4 After backtracking from 5 to 2 and finishing 2
5 After backtracking from 2 to 1 and finishing 1
6 After starting at 4, then discovering and finishing it

Sorting and Searching

Sorting and Searching

- Sorting
  - Arrange a set of elements in a prescribed order
  - Two classes
    - comparison sorts
      - Rely on comparing elements to place them in the correct order
      - *Best: $O(n \log n)$*
    - linear-time sorts
      - Rely on certain characteristics in the data
      - Best: $O(n)$

Sorting and Searching

- Sorting Storage
  - In-place sorts
    - use the same storage that contains the data to store output as the sort proceeds
  - Extra storage
    - use extra storage for the output data

Sorting and Searching

- Searching
  - the ubiquitous task of locating an element in a set of data
  - Linear search
    - the simplest approach
    - scan the set from one end to the other
    - with data structures that do not support random access, such as linked list
  - Binary search
    - data structures like hash tables and binary search trees

Sorting and Searching

- Common sorting algorithms
  - Insertion sort
    - suitable for incremental sorting on small sets of data
  - Quicksort
    - in-place sorting
    - the best for sorting in the general case
    - suitable for medium to large sets of data
  - Merge sort
    - the same performance as quicksort, but with twice its storage requirements
    - suitable for large sets of data because it inherently facilitates working with divisions of the original unsorted set

Sorting and Searching

- Common searching algorithm
  - Binary search
    - An effective way to search sorted data in which we do not expect frequent insertions or deletions
    - Binary search is the best when the data does not change
- Some applications related to sorting and searching
  - Order statistics
  - Binary search
  - Directory listings
  - Spell checkers

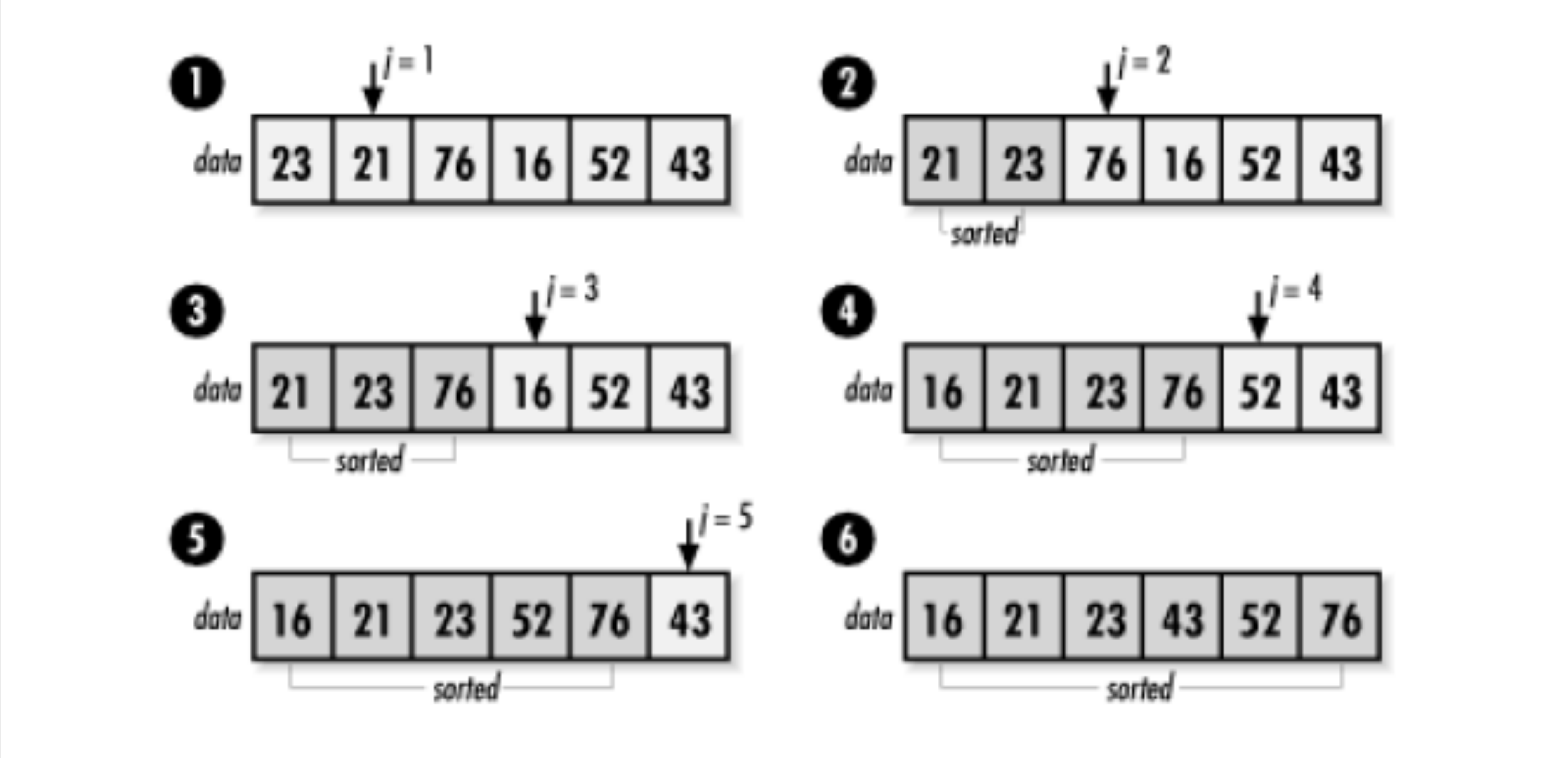
Insert Sort

- Description
  - one of the simplest sorting algorithm
  - Approach
    - takes one element at a time from an unsorted set
    - insets it into a sorted one by scanning the set of sorted elements to determine where the new element belongs

Insert Sort

- Description
  - inefficient for large sets of data
  - because determining where each element belongs in the sorted set potentially requires comparing it with every other element in the sorted set so far
  - insertion sort efficient for incremental sorting

Insert Sort



1
j = 1
data
23 21 76 16 52 43
2
j = 2
data
21 23 76 16 52 43
sorted
3
j = 3
data
21 23 76 16 52 43
sorted
4
j = 4
data
16 21 23 76 52 43
sorted
5
j = 5
data
16 21 23 52 76 43
sorted
6
data
16 21 23 43 52 76
sorted

Insert Sort

- Number of Comparisons

5 7 0 3 4 2 6 1 (0)

5 7 0 3 4 2 6 1 (0)

0 5 7 3 4 2 6 1 (2)

0 **3** 5 7 4 2 6 1 (2)

0 3 **4** 5 7 2 6 1 (2)

0 **2** 3 4 5 7 6 1 (4)

0 2 3 4 5 **6** 7 1 (1)

0 **1** 2 3 4 5 6 7 (6)

Insert Sort

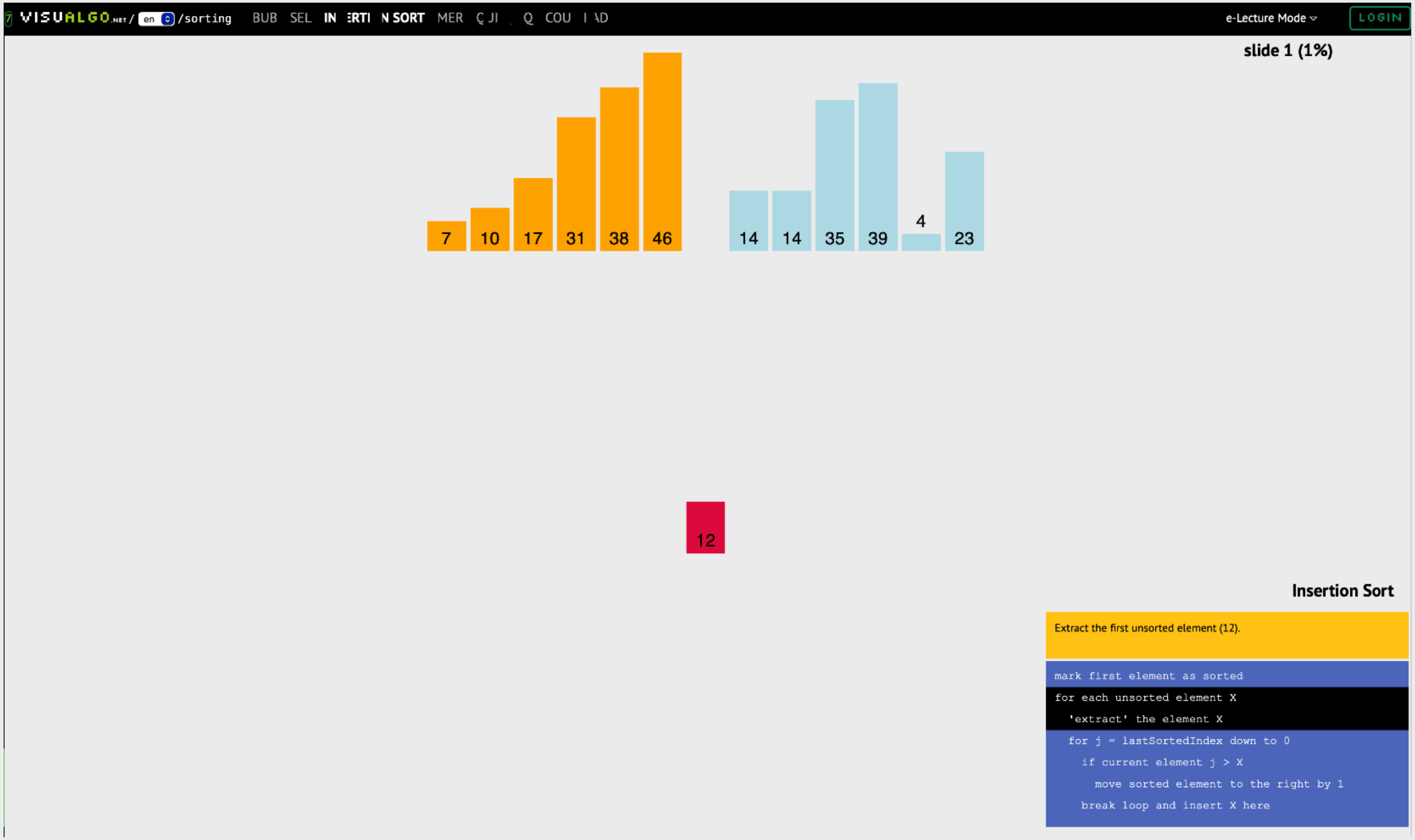
- Number of Comparisons

5	7	0	3	4	2	6	1	(0)
5	7	<u>0</u>	3	4	2	6	1	(0)
0	5	7	<u>3</u>	4	2	6	1	(2)
0	**3**	5	7	<u>4</u>	2	6	1	(2)
0	3	**4**	5	7	<u>2</u>	6	1	(2)
0	**2**	3	4	5	7	<u>6</u>	1	(4)
0	2	3	4	5	**6**	7	<u>1</u>	(1)
0	**1**	2	3	4	5	6	7	(6)

Altogether this amounts to 17 steps

Insert Sort

- Video Demo: Insertion Sort



Insert Sort

- Video Demo: Sorting Algorithms

Insert Sort

- Example: sort/issort.c

```c
int issort(void *data, int size, int esize,
        int (*compare)(const void *key1, const void *key2)) {

    char               *a = data;
    void               *key;
    int                i, j;

    /*****************************************************************************
     *  Allocate storage for the key element.
     *****************************************************************************
    if ((key = (char *)malloc(esize)) == NULL)
        return -1;

    /*****************************************************************************
     *  Repeatedly insert a key element among the sorted elements.
     *****************************************************************************
    for (j = 1; j < size; j++) {
        memcpy(key, &a[j * esize], esize);
        i = j - 1;

        /*************************************************************************
         *  Determine the position at which to insert the key element.
         *************************************************************************
        while (i >= 0 && compare(&a[i * esize], key) > 0) {
            memcpy(&a[(i + 1) * esize], &a[i * esize], esize);
            i--;
        }
        memcpy(&a[(i + 1) * esize], key, esize);
    }

    /*****************************************************************************
     *  Free the storage allocated for sorting.
     *****************************************************************************
    free(key);

    return 0;
}
```

Insert Sort

- Example: sort/issort.c

```c
int issort(void *data, int size, int esize,
        int (*compare)(const void *key1, const void *key2)) {

    char                *a = data;
    void                *key;
    int                 i, j;

    /*****************************************************************
     *  Allocate storage for the key element.
     *****************************************************************/
    if ((key = (char *)malloc(esize)) == NULL)
        return -1;

    /*****************************************************************
     *  Repeatedly insert a key element among the sorted elements.
     *****************************************************************/
    for (j = 1; j < size; j++) {
        memcpy(key, &a[j * esize], esize);
        i = j - 1;

        /*************************************************************
         *  Determine the position at which to insert the key element.
         *************************************************************/
        while (i >= 0 && compare(&a[i * esize], key) > 0) {
            memcpy(&a[(i + 1) * esize], &a[i * esize], esize);
            i--;
        }
        memcpy(&a[(i + 1) * esize], key, esize);
    }

    /*****************************************************************
     *  Free the storage allocated for sorting.
     *****************************************************************/
    free(key);

    return 0;
}
```

Insert Sort

- Analysis of time complexity
  - the runtime complexity of insertion sort focuses on its nested loops
  - the running time of the nested loop is a summation from *1* to *n-1*
    - $T(n) = (n\ (n+1)\ /\ 2) - n = O(n^2)$

Insert Sort

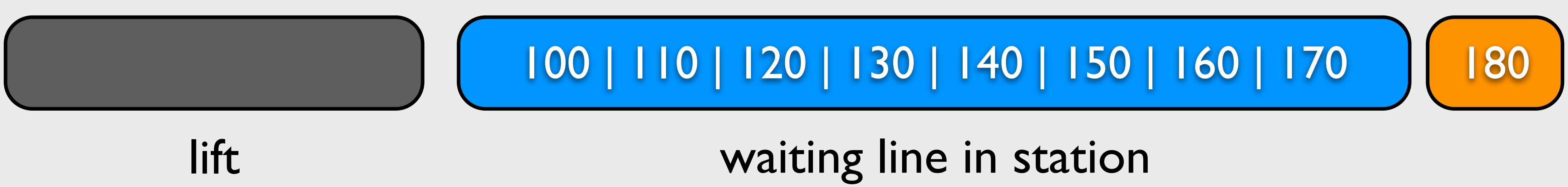
- Summary
  - Simple implementation
  - Efficient for (quite) small data sets
  - More efficient in practice than most other simple quadratic algorithms
  - Stable (i.e., does not change the relative order of elements with equal keys)
  - In-place (i.e., O(1) space)
  - Online (i.e., can sort a list as it receives it)

Assignment 9

- You have to implement your stack and queue
- Use the stack and queue to solve the Riverfront Lift Problem (see the webpage)

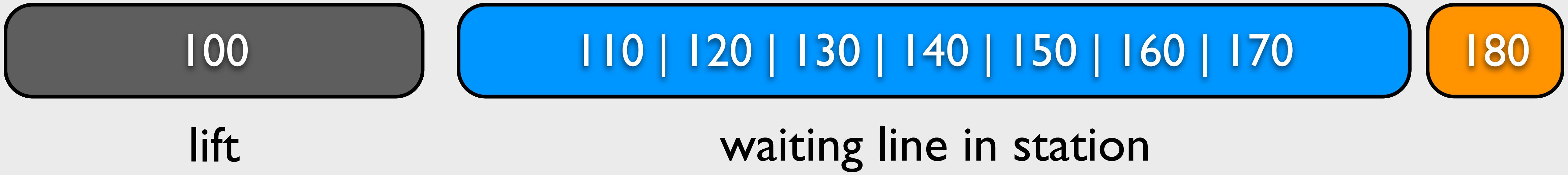
An Example for Explanation

- Use queue to simulate the waiting line in the station
- Use stack to simulate the riverfront lift
- Example: {100, 110, 120, 130, 140, 150, 160, 170, 180}



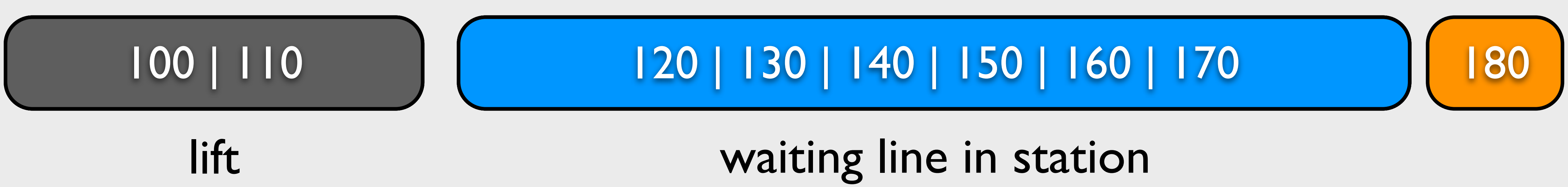
An Example for Explanation

- Use queue to simulate the waiting line in the station
- Use stack to simulate the riverfront lift
- Example: {100, 110, 120, 130, 140, 150, 160, 170, 180}



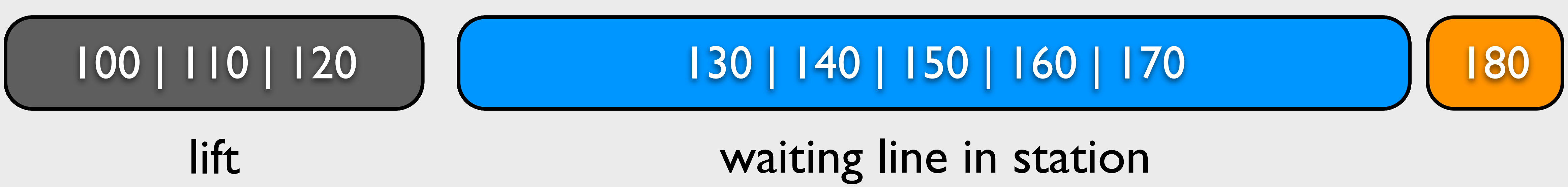
An Example for Explanation

- Use queue to simulate the waiting line in the station
- Use stack to simulate the riverfront lift
- Example: {100, 110, 120, 130, 140, 150, 160, 170, 180}



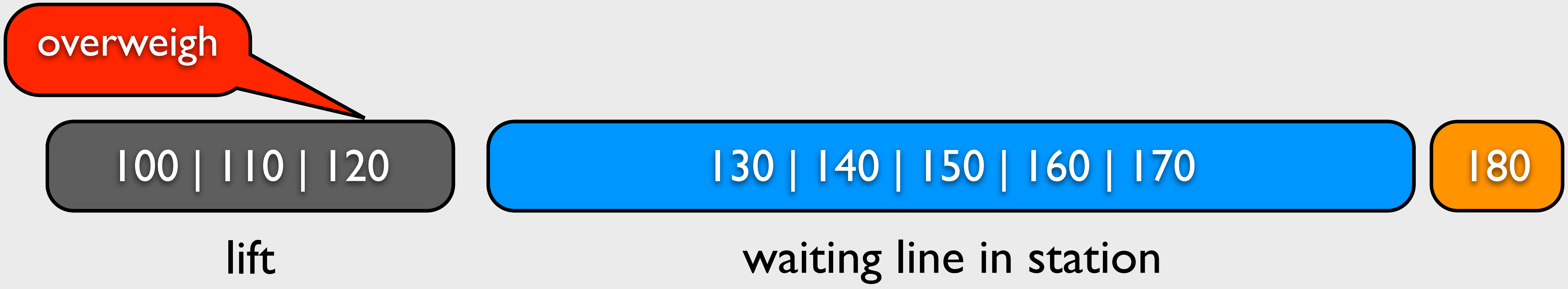
An Example for Explanation

- Use queue to simulate the waiting line in the station
- Use stack to simulate the riverfront lift
- Example: {100, 110, 120, 130, 140, 150, 160, 170, 180}



An Example for Explanation

- Use queue to simulate the waiting line in the station
- Use stack to simulate the riverfront lift
- Example: {100, 110, 120, 130, 140, 150, 160, 170, 180}

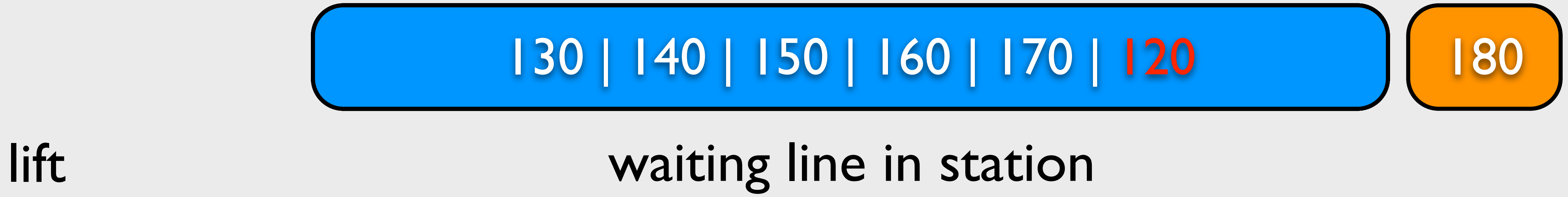


An Example for Explanation

- Use queue to simulate the waiting line in the station
- Use stack to simulate the riverfront lift
- Example: {100, 110, 120, 130, 140, 150, 160, 170, 180}

An Example for Explanation

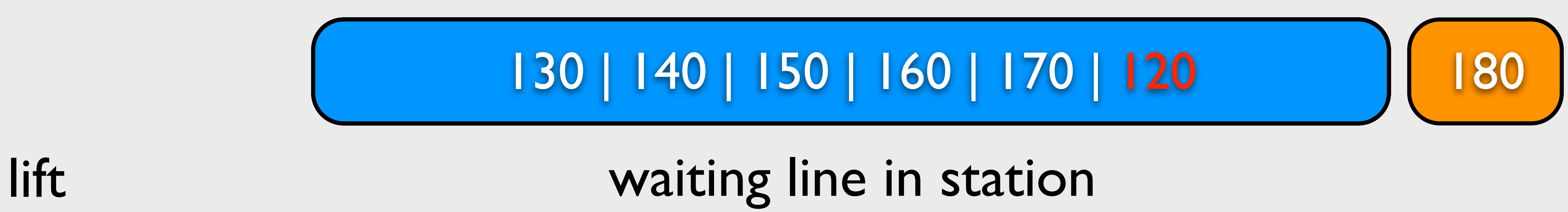
- Use queue to simulate the waiting line in the station
- Use stack to simulate the riverfront lift
- Example: {100, 110, 120, 130, 140, 150, 160, 170, 180}



An Example for Explanation

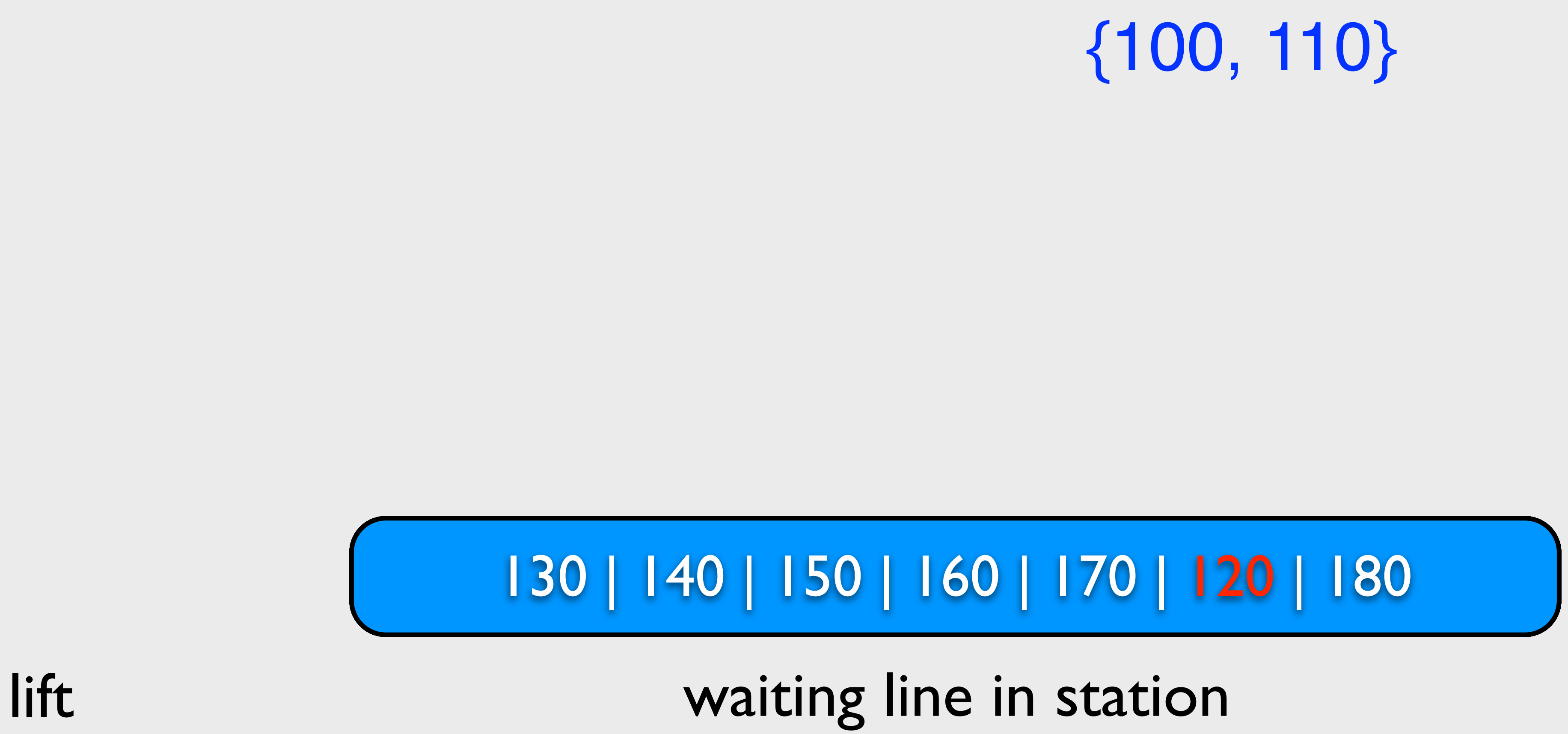
- Use queue to simulate the waiting line in the station
- Use stack to simulate the riverfront lift
- Example: {100, 110, 120, 130, 140, 150, 160, 170, 180}

{100, 110}



An Example for Explanation

- Use queue to simulate the waiting line in the station
- Use stack to simulate the riverfront lift
- Example: {100, 110, 120, 130, 140, 150, 160, 170, 180}



An Example for Explanation

- Use queue to simulate the waiting line in the station
- Use stack to simulate the riverfront lift
- Example: {100, 110, 120, 130, 140, 150, 160, 170, 180}

{100, 110}

An Example for Explanation

- Use queue to simulate the waiting line in the station
- Use stack to simulate the riverfront lift
- Example: {100, 110, 120, 130, 140, 150, 160, 170, 180}

{100, 110}

130	140 | 150 | 160 | 170 | 120 | 180
lift	waiting line in station

An Example for Explanation

- Use queue to simulate the waiting line in the station
- Use stack to simulate the riverfront lift
- Example: {100, 110, 120, 130, 140, 150, 160, 170, 180}

{100, 110}

130 | 140

lift

150 | 160 | 170 | 120 | 180

waiting line in station

An Example for Explanation

- Use queue to simulate the waiting line in the station
- Use stack to simulate the riverfront lift
- Example: {100, 110, 120, 130, 140, 150, 160, 170, 180}

{100, 110}

130 | 140 | 150
lift

160 | 170 | 120 | 180
waiting line in station

An Example for Explanation

- Use queue to simulate the waiting line in the station
- Use stack to simulate the riverfront lift
- Example: {100, 110, 120, 130, 140, 150, 160, 170, 180}

{100, 110}

An Example for Explanation

- Use queue to simulate the waiting line in the station
- Use stack to simulate the riverfront lift
- Example: {100, 110, 120, 130, 140, 150, 160, 170, 180}

{100, 110}

130 | 140

lift

160 | 170 | 120 | 180 | 150

waiting line in station

An Example for Explanation

- Use queue to simulate the waiting line in the station
- Use stack to simulate the riverfront lift
- Example: {100, 110, 120, 130, 140, 150, 160, 170, 180}

{100, 110}

160 | 170 | 120 | 180 | 150

lift

waiting line in station

An Example for Explanation

- Use queue to simulate the waiting line in the station
- Use stack to simulate the riverfront lift
- Example: {100, 110, 120, 130, 140, 150, 160, 170, 180}

{100, 110}

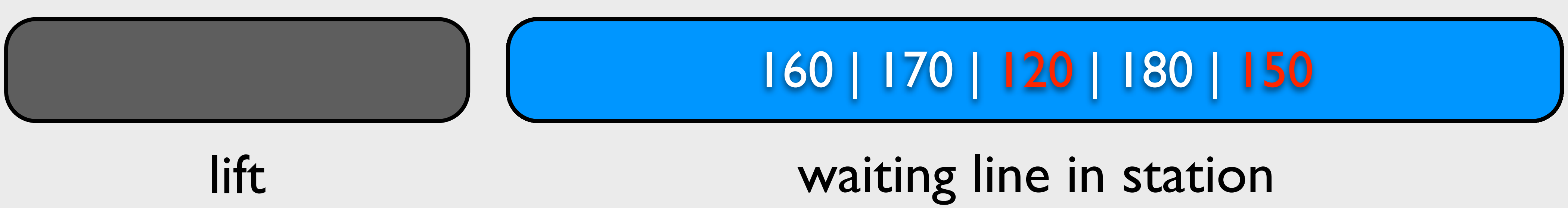
{130, 140}

An Example for Explanation

- Use queue to simulate the waiting line in the station
- Use stack to simulate the riverfront lift
- Example: {100, 110, 120, 130, 140, 150, 160, 170, 180}

{100, 110}

{130, 140}



An Example for Explanation

- Use queue to simulate the waiting line in the station
- Use stack to simulate the riverfront lift
- Example: {100, 110, 120, 130, 140, 150, 160, 170, 180}

{100, 110}

{130, 140}

lift	waiting line in station
160	170 | 120 | 180 | 150

An Example for Explanation

- Use queue to simulate the waiting line in the station
- Use stack to simulate the riverfront lift
- Example: {100, 110, 120, 130, 140, 150, 160, 170, 180}

{100, 110}

{130, 140}

160 | 170

lift

120 | 180 | 150

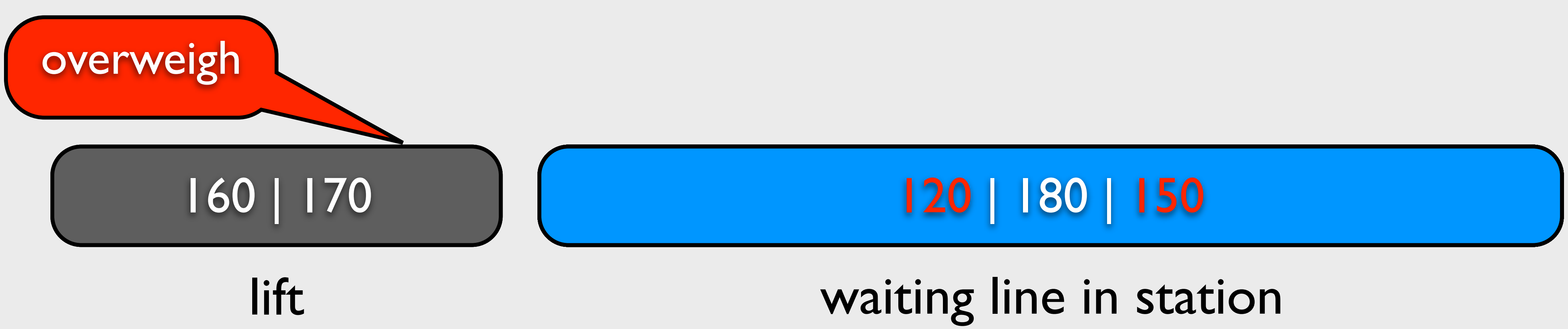
waiting line in station

An Example for Explanation

- Use queue to simulate the waiting line in the station
- Use stack to simulate the riverfront lift
- Example: {100, 110, 120, 130, 140, 150, 160, 170, 180}

{100, 110}

{130, 140}



An Example for Explanation

- Use queue to simulate the waiting line in the station
- Use stack to simulate the riverfront lift
- Example: {100, 110, 120, 130, 140, 150, 160, 170, 180}

{100, 110}

{130, 140}

160

lift

120 | 180 | 150 | 170

waiting line in station

An Example for Explanation

- Use queue to simulate the waiting line in the station
- Use stack to simulate the riverfront lift
- Example: {100, 110, 120, 130, 140, 150, 160, 170, 180}

{100, 110}

{130, 140}

120 | 180 | 150 | 170

lift

waiting line in station

An Example for Explanation

- Use queue to simulate the waiting line in the station
- Use stack to simulate the riverfront lift
- Example: {100, 110, 120, 130, 140, 150, 160, 170, 180}

{100, 110}

{130, 140}

{160}

120 | 180 | 150 | 170

lift

waiting line in station

An Example for Explanation

- Use queue to simulate the waiting line in the station
- Use stack to simulate the riverfront lift
- Example: {100, 110, 120, 130, 140, 150, 160, 170, 180}

{100, 110}

{130, 140}

{160}

An Example for Explanation

- Use queue to simulate the waiting line in the station
- Use stack to simulate the riverfront lift
- Example: {100, 110, 120, 130, 140, 150, 160, 170, 180}

{100, 110}

{130, 140}

{160}

120

lift

180 | 150 | 170

waiting line in station

An Example for Explanation

- Use queue to simulate the waiting line in the station
- Use stack to simulate the riverfront lift
- Example: {100, 110, 120, 130, 140, 150, 160, 170, 180}

{100, 110}

{130, 140}

{160}

An Example for Explanation

- Use queue to simulate the waiting line in the station
- Use stack to simulate the riverfront lift
- Example: {100, 110, 120, 130, 140, 150, 160, 170, 180}

{100, 110}

{130, 140}

{160}

lift

waiting line in station

An Example for Explanation

- Use queue to simulate the waiting line in the station
- Use stack to simulate the riverfront lift
- Example: {100, 110, 120, 130, 140, 150, 160, 170, 180}

{100, 110}

{130, 140}

{160}

{120, 180}

150 | 170

lift

waiting line in station