충북대학교 NOVAAPERIO 1951 CHUNGBUK NATIONAL UNIVERSITY

Lecture 8:
Sorting algorithm (Part. 2)

Algorithm

Jeong-Hun Kim

Remind

- Sorting problem
- Sorting algorithms
  - Basic sorting algorithms
    - Selection sort
    - Bubble sort
    - Insertion sort
  - Advanced sorting algorithms
    - Shell sort
    - Merge sort
    - Quick sort
    - Heap sort
    - Etc.

Table of Contents

- Part 1
  - How to Improve the Sorting Algorithm

- Part 2
  - Representative Advanced Sorting Algorithms

Part 1

HOW TO IMPROVE THE SORTING ALGORITHM

How to Improve Sorting Algorithm

- Factors that determine the complexity of the sorting algorithm
  - n: the number of elements
    - The number of comparisons
    - The number of swaps (in case of the in-place comparison sort)
  - In general, the time complexity of basic sorting algorithm: $O(n^2)$

- Ways to reduce the time complexity
  - Reducing the number of comparisons
  - Reducing the number of swaps
    - Trade-off between time and space complexities

How to Improve Sorting Algorithm

- Existing advanced sorting algorithms
  - Specific data structures
  - Additional memory space
  - E.g.,
    - Shell sort
    - Merge sort: divide-and-conquer (additional memory space)
    - Quick sort: pivot, divide-and-conquer (additional memory space)
    - Heap sort: tree structure
      - Specific data structure, additional memory space
    - Radix sort: bins (additional memory space)
    - Bucket sort: buckets (additional memory space)
    - Tim sort: runs (additional memory space)

Part 2

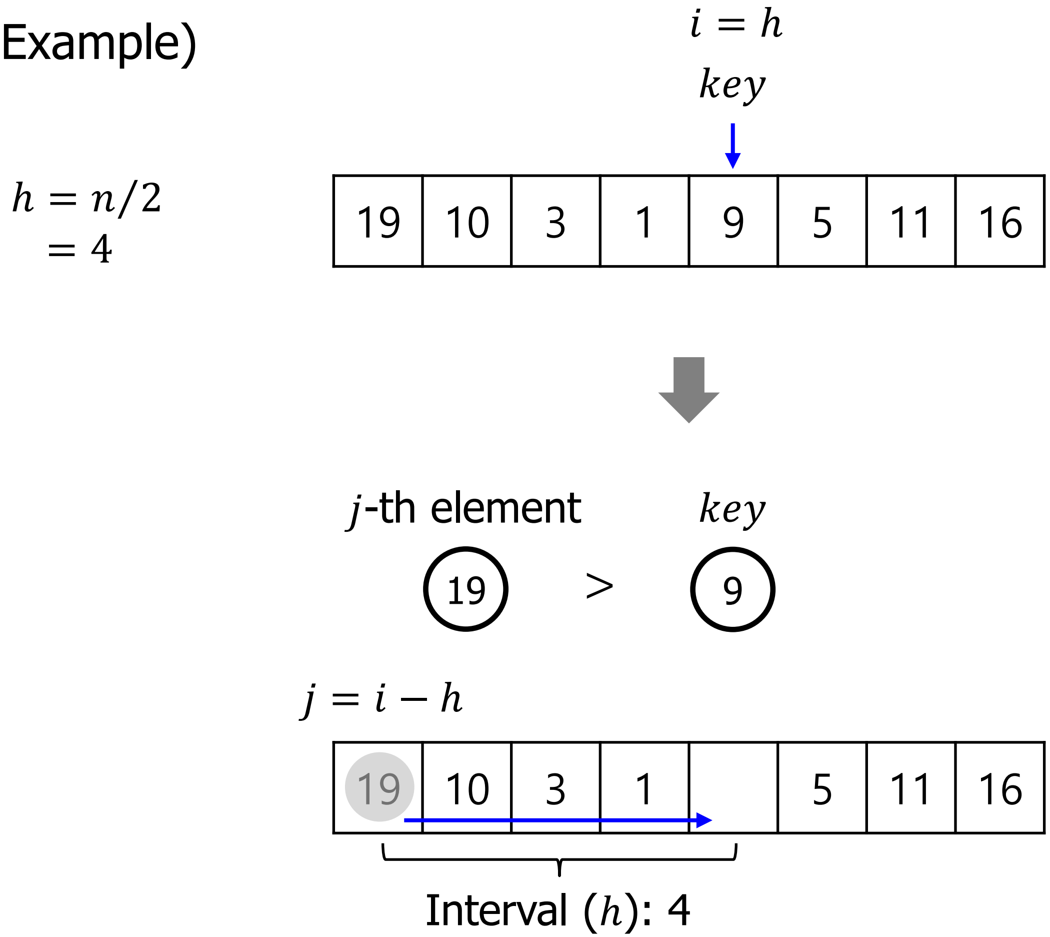
REPRESENTATIVE ADVANCED SORTING ALGORITHMS

Representative Advanced Sorting Algorithms

- Shell sort
  - A variation of insertion sort algorithm
    - Insertion sort is efficient for lists that are somewhat sorted
    - Swapping is allowed not only with adjacent neighbors, but also with spaced neighbors
  - In-place comparison sort
  - Time complexity: $O(n^{1.25})$
  - Methodology:
    1) Initialize the value of gap size h
    2) Divde the list into smaller sub-part
    3) Sort these sub-lists using insertion sort
    4) Repeat 2) – 3) steps until the list is sorted

Representative Advanced Sorting Algorithms

- Shell sort
  - Example)



$i = h$

key

$h = n/2$
$= 4$

19	10	3	1	9	5	11	16

j-th element $\quad key$

19 > 9

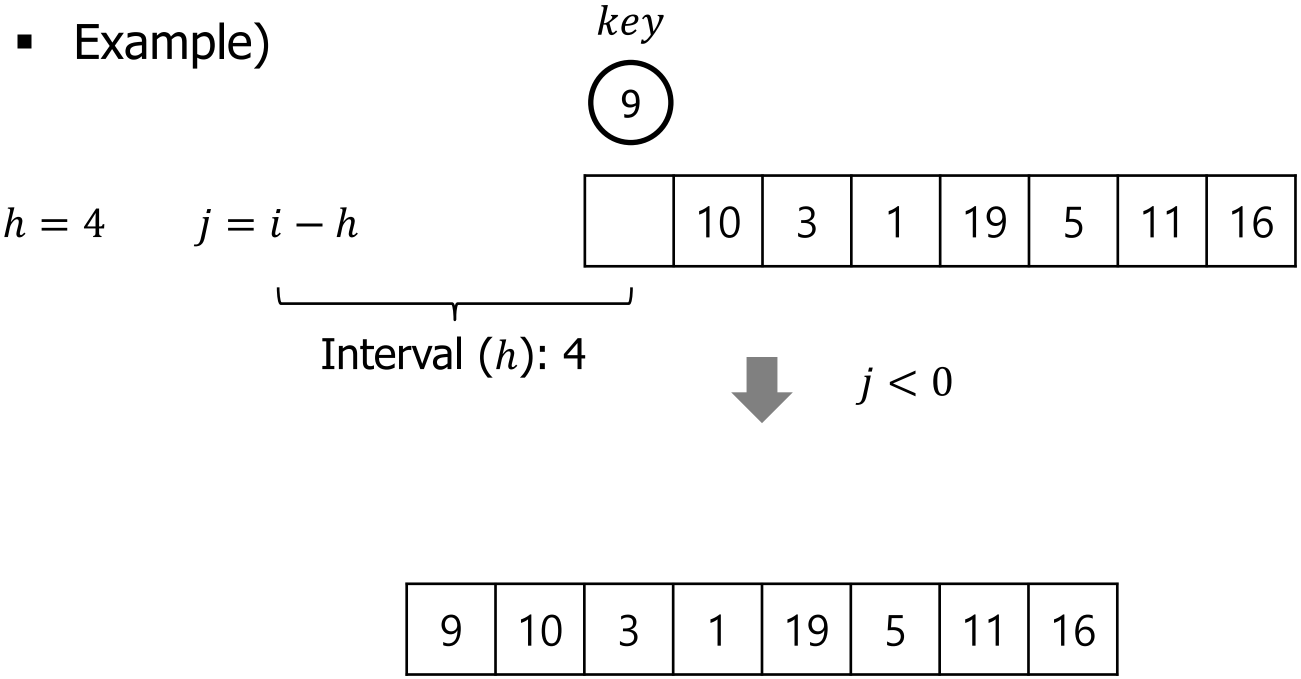
$j = i - h$

19	10	3	1		5	11	16

Interval (h): 4

Representative Advanced Sorting Algorithms

- Shell sort
  - Example)



key

9

$h = 4$ $j = i - h$

	10	3	1	19	5	11	16

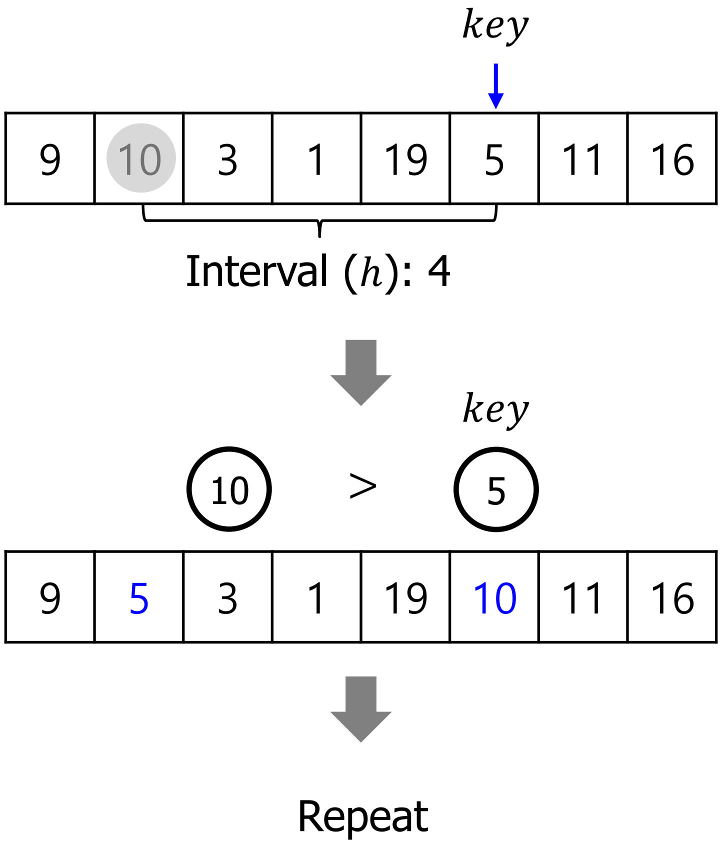
Interval (h): 4

$j < 0$

9	10	3	1	19	5	11	16

Representative Advanced Sorting Algorithms

- Shell sort
  - Example)



key

9	10	3	1	19	5	11	16

Interval (h): 4

key

10 > 5

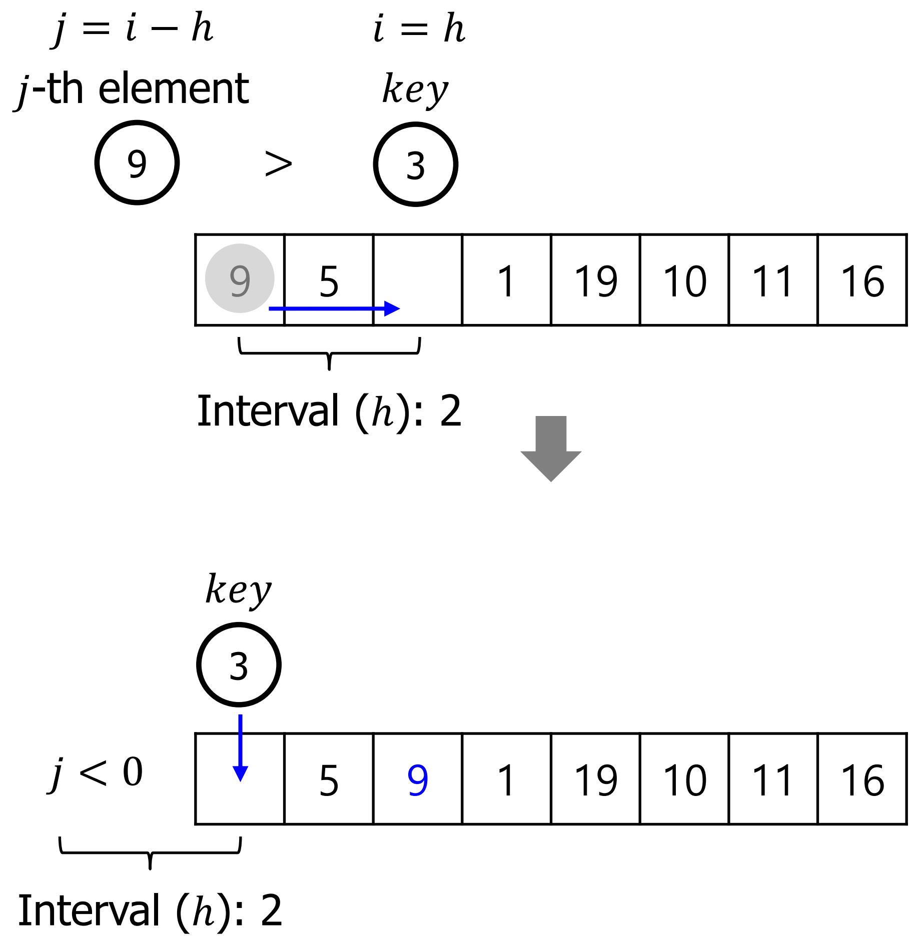
9	5	3	1	19	10	11	16

Repeat

Representative Advanced Sorting Algorithms

- Shell sort
  - Example)

$h = 2$



$j = i - h$

j-th element

9 > 3

$i = h$

key

9	5		1	19	10	11	16

Interval (h): 2

key

3

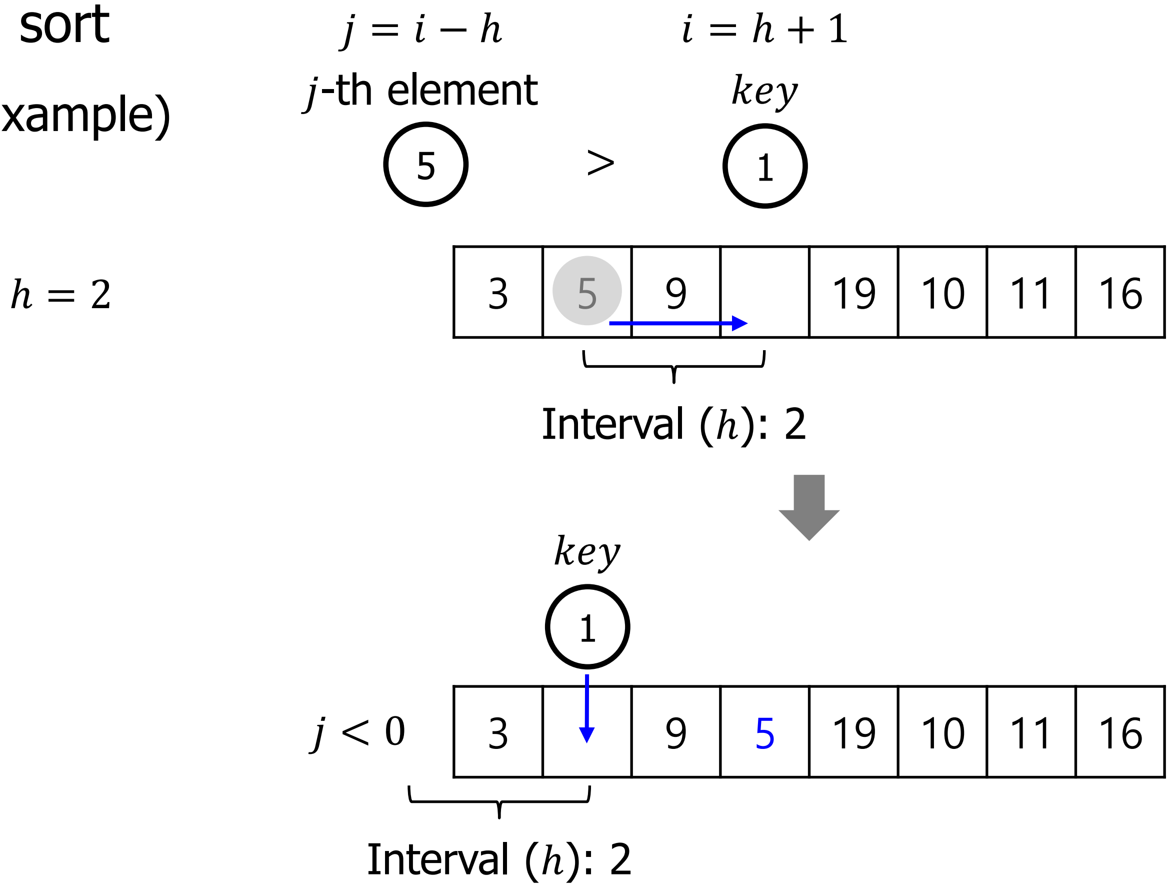
$j < 0$

	5	9	1	19	10	11	16

Interval (h): 2

Representative Advanced Sorting Algorithms

- Shell sort
  - Example)



$j = i - h$

$i = h + 1$

j-th element

key

5 > 1

$h = 2$

3	5	9		19	10	11	16

Interval (h): 2

key

1

$j < 0$

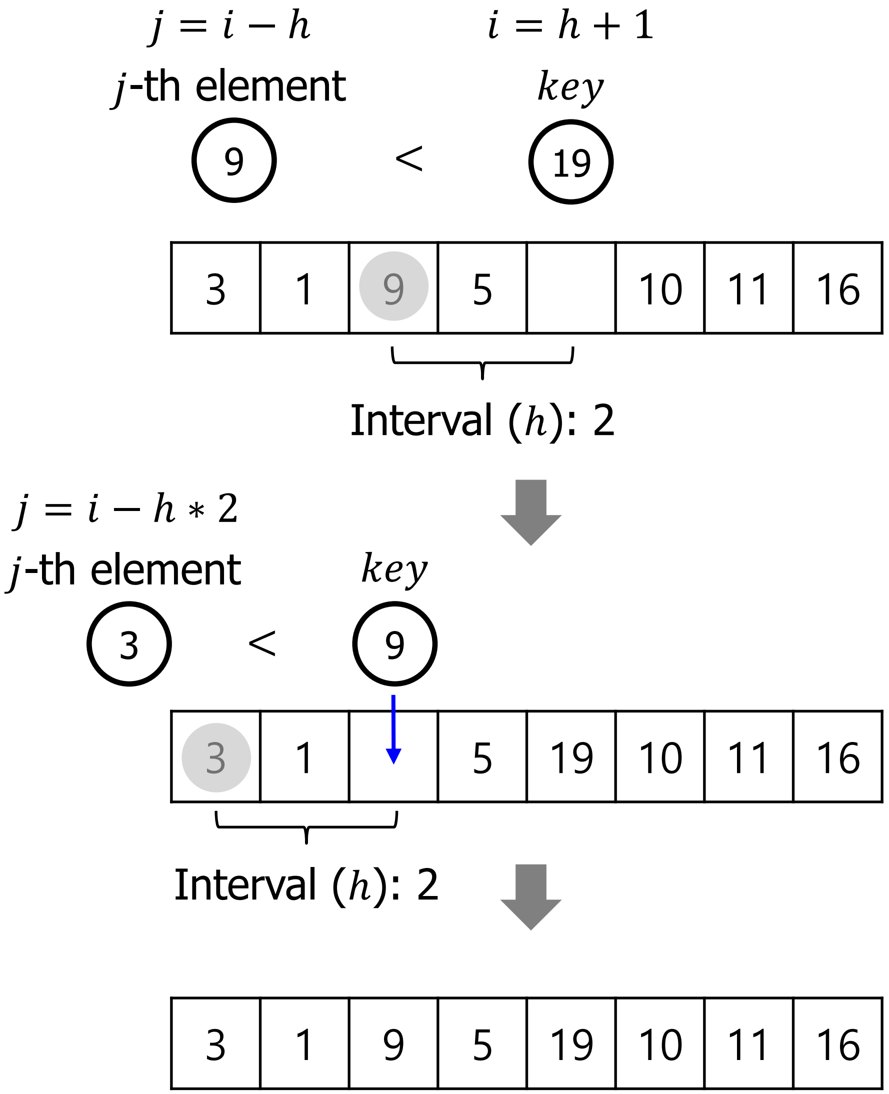
3		9	5	19	10	11	16

Interval (h): 2

Representative Advanced Sorting Algorithms

- ❖ Shell sort
  - ▪ Example)

$h = 2$



$j = i - h$ $i = h + 1$

j-th element key

9 < 19

3	1	9	5		10	11	16

Interval (h): 2

$j = i - h * 2$

j-th element key

3 < 9

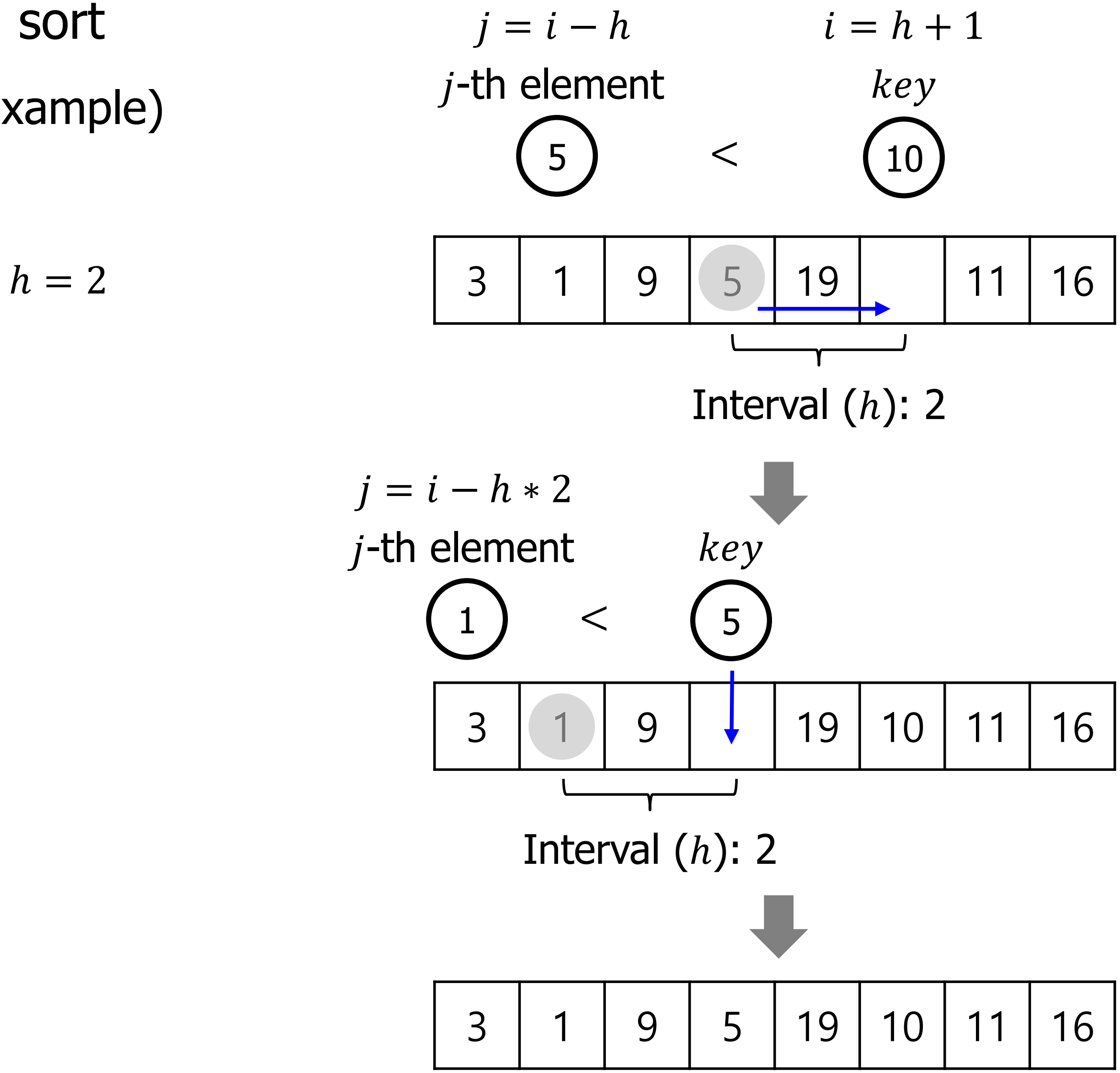
3	1		5	19	10	11	16

Interval (h): 2

3	1	9	5	19	10	11	16

Representative Advanced Sorting Algorithms

- Shell sort
  - Example)



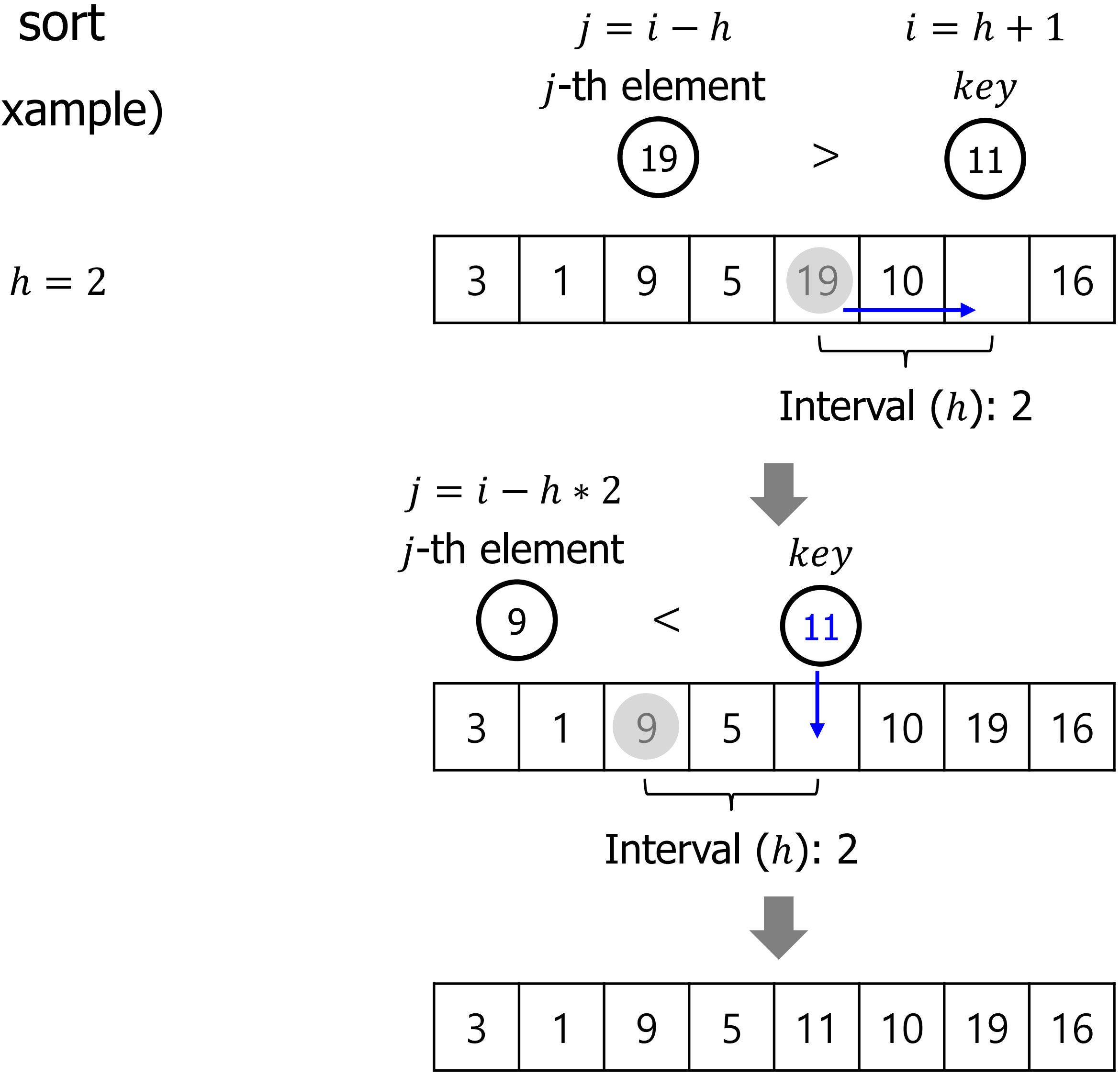
3	1	9	5	19		11	16

3	1	9		19	10	11	16

3	1	9	5	19	10	11	16

Representative Advanced Sorting Algorithms

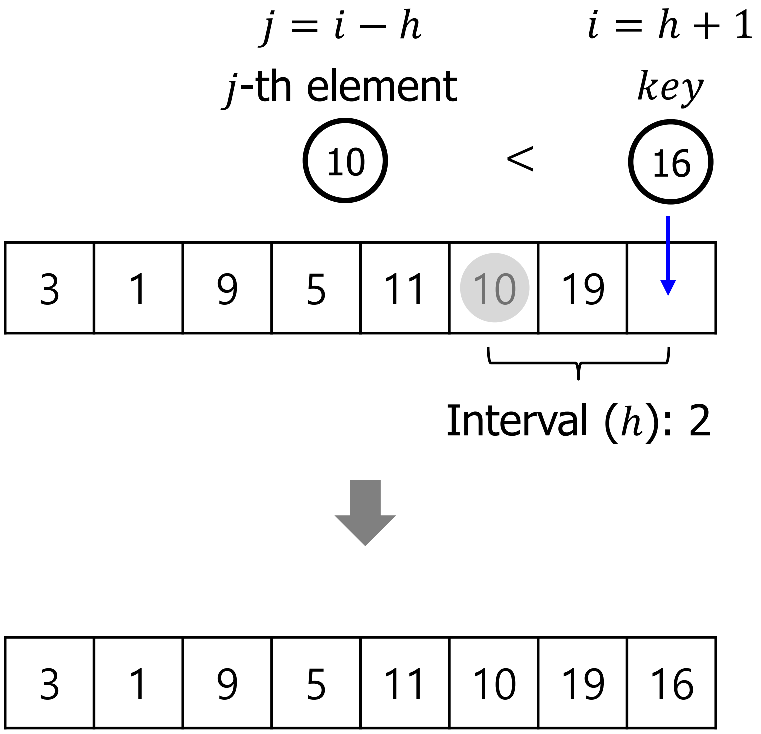
- Shell sort
  - Example)



Representative Advanced Sorting Algorithms

- ❖ Shell sort
  - Example)

$h = 2$



$j = i - h$ $\quad$ $i = h + 1$

j-th element $\quad$ key

$10 < 16$

3	1	9	5	11	10	19	

Interval (h): 2

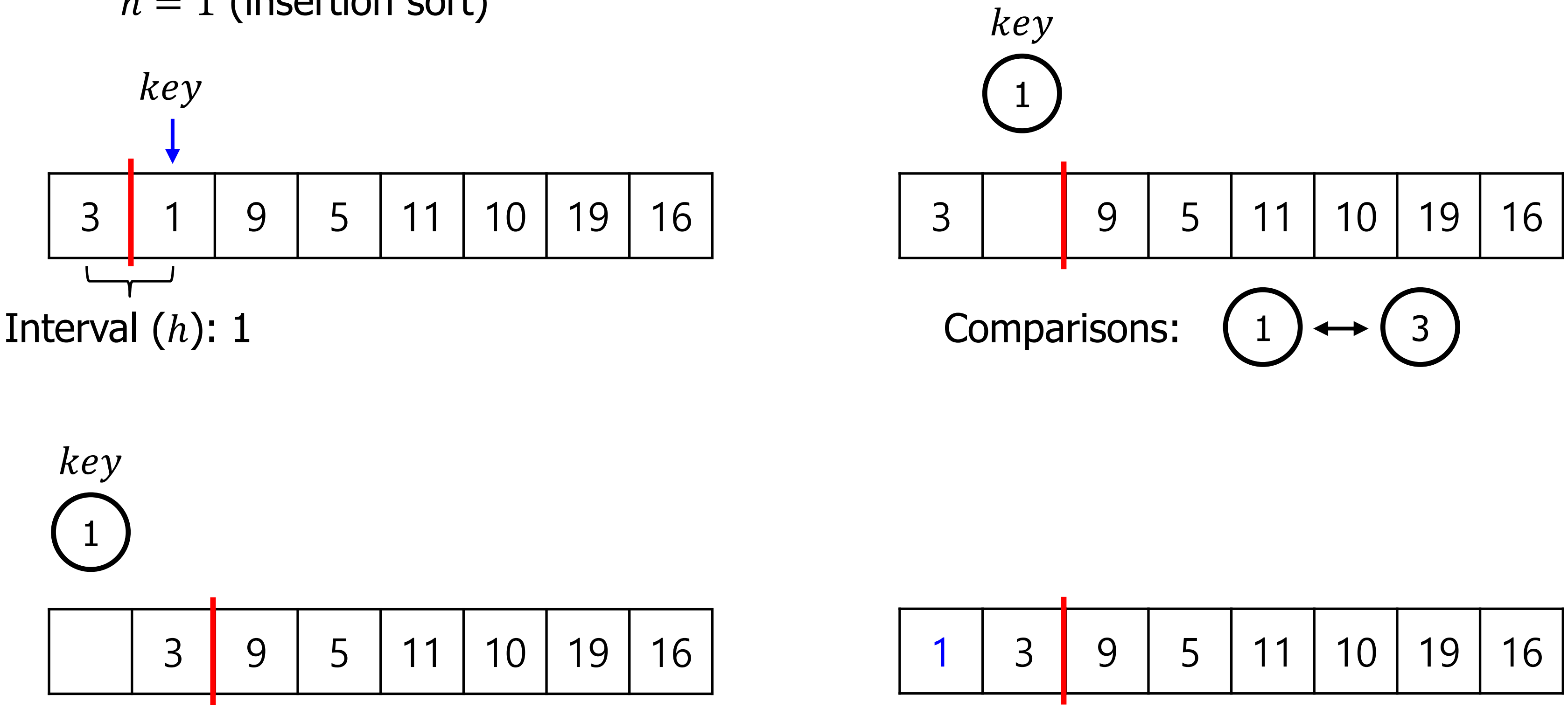
3	1	9	5	11	10	19	16

Representative Advanced Sorting Algorithms

❖ Shell sort

- Example)

$h = 1$ (insertion sort)



3	1	9	5	11	10	19	16

3		9	5	11	10	19	16

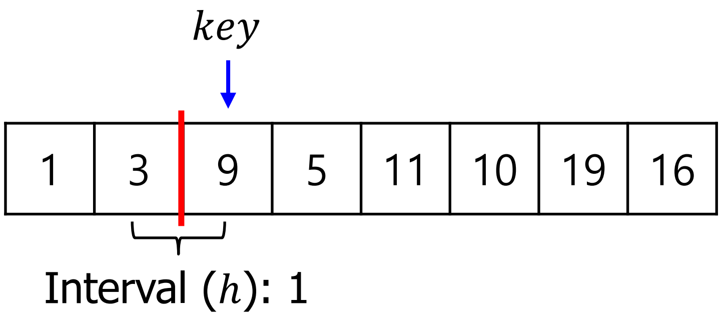
	3	9	5	11	10	19	16

1	3	9	5	11	10	19	16

Representative Advanced Sorting Algorithms

- Shell sort
  - Example)

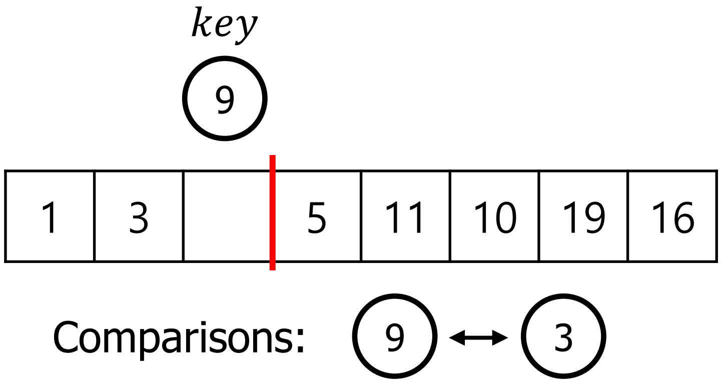
$h = 1$ (insertion sort)



key

1	3	9	5	11	10	19	16

Interval (h): 1

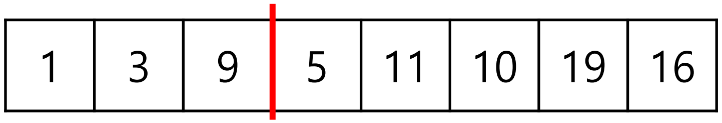


key

9

1	3		5	11	10	19	16

Comparisons: 9 ⟷ 3

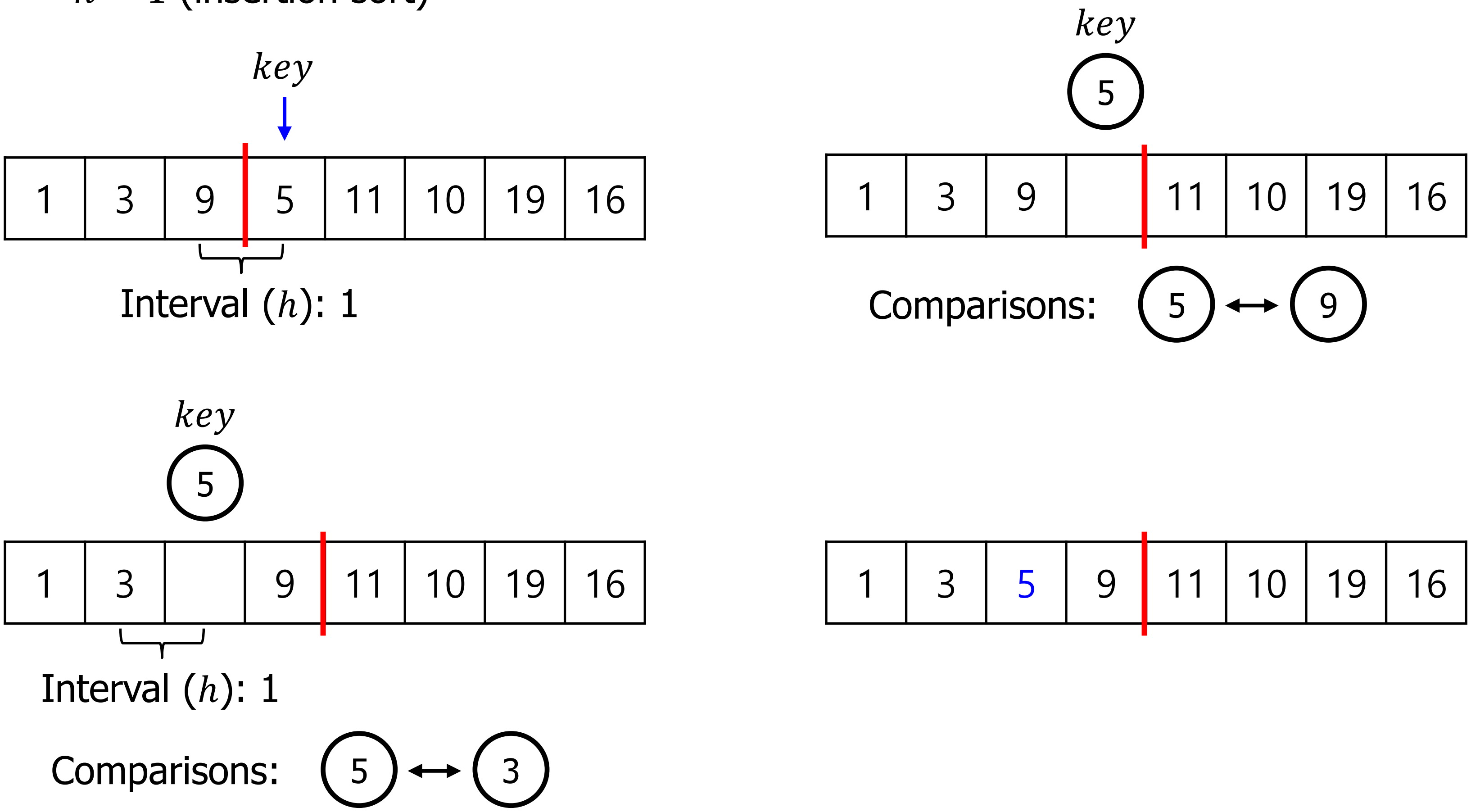


1	3	9	5	11	10	19	16

Representative Advanced Sorting Algorithms

- Shell sort
  - Example)

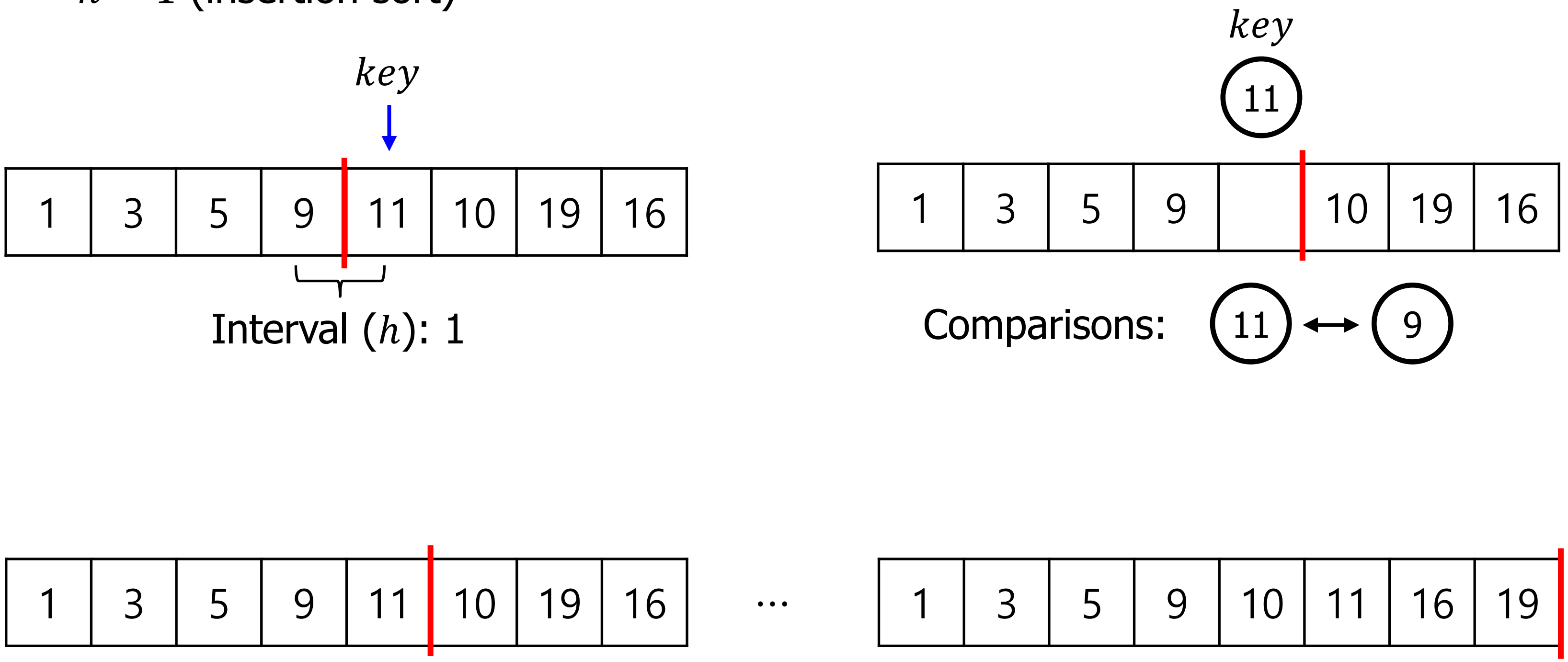
$h = 1$ (insertion sort)



Representative Advanced Sorting Algorithms

- Shell sort
  - Example)

$h = 1$ (insertion sort)



key

1	3	5	9	11	10	19	16

Interval (h): 1

key

11

1	3	5	9		10	19	16

Comparisons: 11 ⟷ 9

1	3	5	9	11	10	19	16

⋯

1	3	5	9	10	11	16	19

Representative Advanced Sorting Algorithms

- Shell sort
    - Psuedo code)

```
#include <stdio.h>

void shell_sort(int arr[], int n){
   int h, i, j, key;
   for(h = n / 2; h > 0; h /= 2){
      for(i = h; i < n; i++){
         key = arr[i];
         for(j = i; j >= h && arr[j – h] > key; j –= h)
            arr[j] = arr[j – h]
         arr[j] = key;
      }
   }
}
```

Representative Advanced Sorting Algorithms

- Merge sort
  - Divide and conquer algorithm
  - Out-of-place comparison sort
  - Time complexity: $O(n \log n)$
  - Methodology:
    1) Divide the list recursively into two sublists based on its midpoint
    2) Sort each sublist
    3) Merge pair of sorted sublists
    4) Repeat 2) – 3) steps until reconstructing the list

  - Please refer to the example in Chapter 4

Representative Advanced Sorting Algorithms

- Merge sort
  - Psuedo code)

```
#include <stdio.h>
#include <stdlib.h>

void merge_sort(int arr[], int left, int right){
   if (left < right){
      int mid = left + (right – 1) / 2;
      merge_sort(arr, left, mid);
      merge_sort(arr, mid + 1, right);
      merge(arr, left, mid, right);
   }
}
```

Representative Advanced Sorting Algorithms

- Merge sort
  - Psuedo code)

```
void merge(int arr[], int left, int mid, int right){
  int i, j, k;
  int n1 = mid – left + 1;
  int n2 = right – mid;
  int L[n1], R[n2];
  for(i = 0; i < n1; i++)
    L[i] = arr[left + i];
  for(j = 0; j < n2; j++)
    R[j] = arr[m + 1 + j];
  i = 0;
  j = 0;
  k = left;
  while(i < n1 && j < n2){
    if (L[i] <= R[j]){
      arr[k] = L[i];
      i++;
    }else{
      arr[k] = R[j];
      j++;
    }
    k++;
  }
  while(i < n1){
    arr[k] = L[i];
    i++;
    k++;
  }
  while(j < n2){
    arr[k] = R[j];
    j++;
    k++;
  }
}
```

Representative Advanced Sorting Algorithms

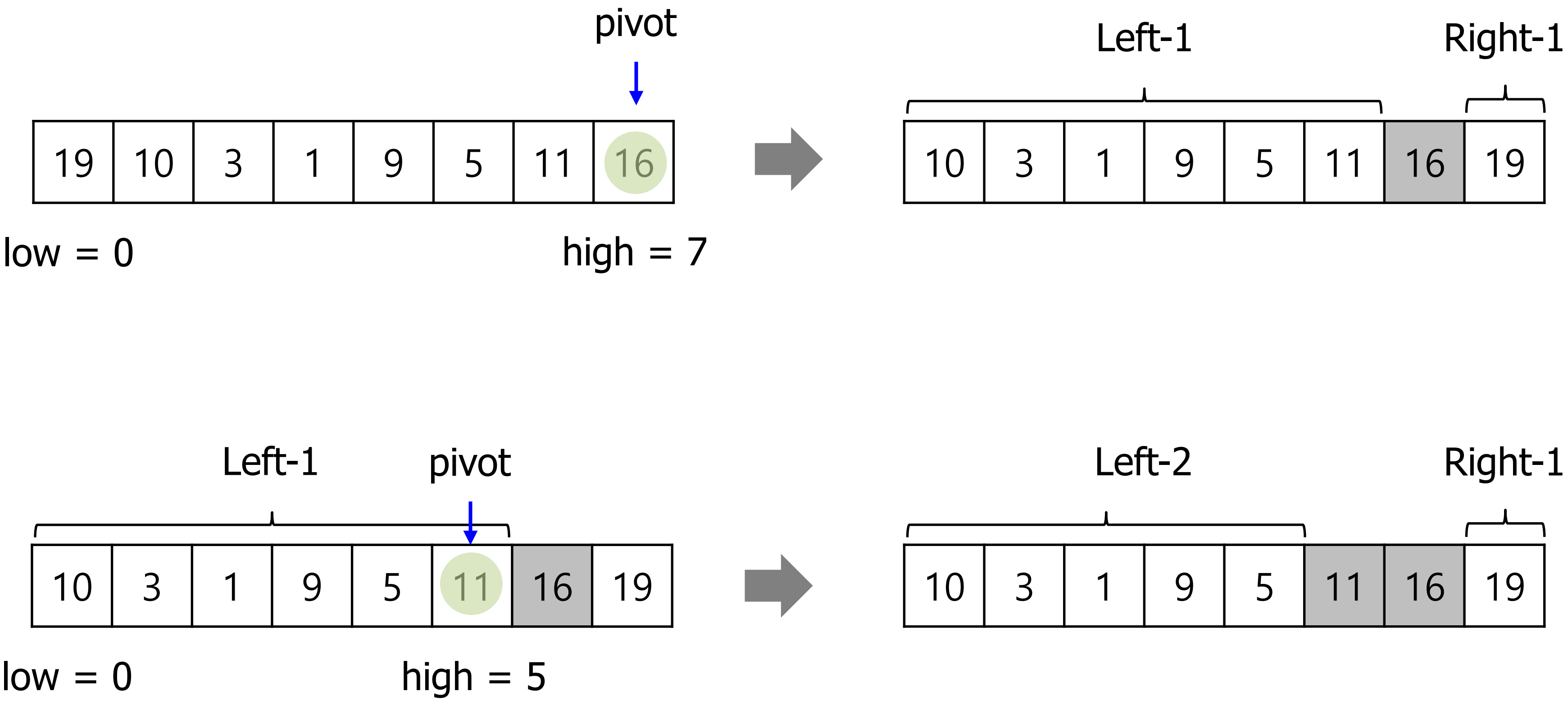
- Quick sort
  - Divide and conquer algorithm
  - In-place comparison sort
  - Time complexity: $\Theta(n \log n)$ (in worst case, $O(n^2)$
  - Methodology:
    1) Choose an element as a pivot among all elements
    2) Move smaller elements to the left and larger elements to the right subsets with the pivot as the reference point
    3) Repeat 1) – 2) steps for the left subset
    4) Repeat 1) – 2) steps for the right subset
    5) Repeat 1) – 4) steps until there are no more subsets

Representative Advanced Sorting Algorithms

- Quick sort
  - How to choose a pivot?
    - Always pick the first element as a pivot
    - Always pick the last element as a pivot
    - Pick a random element as a pivot
    - Pick the middle as the pivot

Representative Advanced Sorting Algorithms

- Quick sort
  - Example) Last element as a pivot



pivot

19	10	3	1	9	5	11	16

low = 0 high = 7

Left-1 Right-1

10	3	1	9	5	11	16	19

Left-1 pivot

10	3	1	9	5	11	16	19

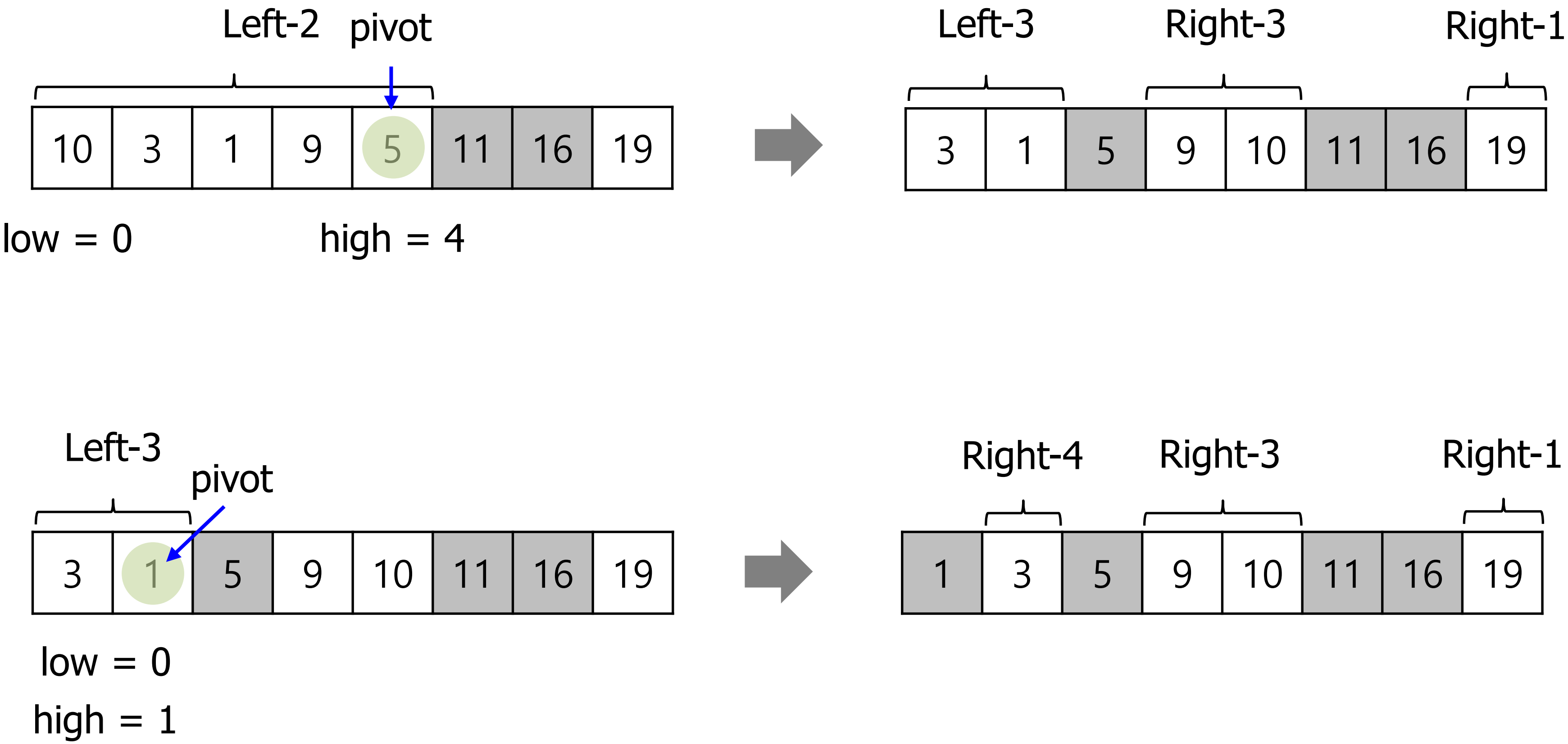
low = 0 high = 5

Left-2 Right-1

10	3	1	9	5	11	16	19

Representative Advanced Sorting Algorithms

- Quick sort
  - Example) Last element as a pivot



Left-2

pivot

10 | 3 | 1 | 9 | 5 | 11 | 16 | 19

low = 0

high = 4

Left-3

Right-3

Right-1

3 | 1 | 5 | 9 | 10 | 11 | 16 | 19

Left-3

pivot

3 | 1 | 5 | 9 | 10 | 11 | 16 | 19

low = 0

high = 1

Right-4

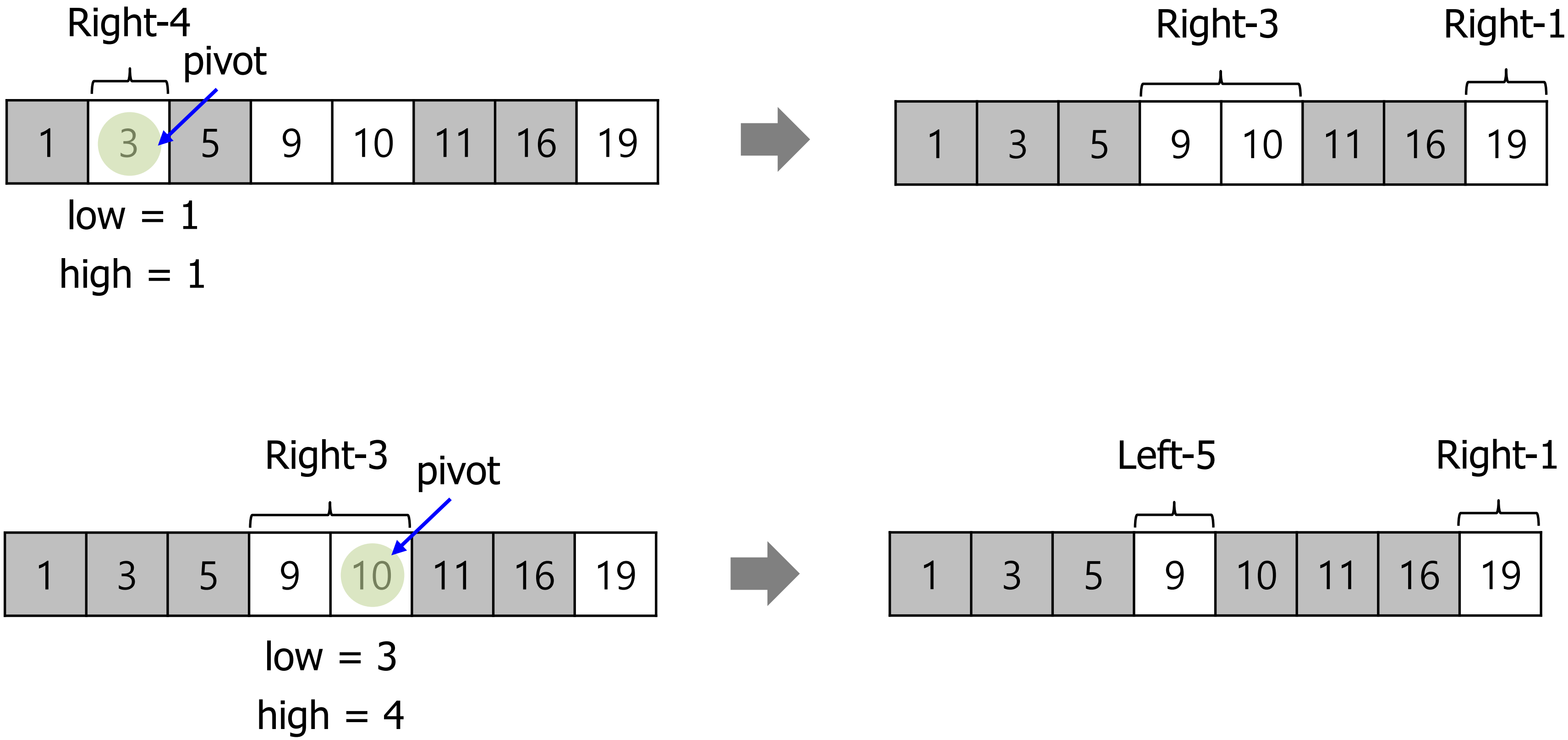
Right-3

Right-1

1 | 3 | 5 | 9 | 10 | 11 | 16 | 19

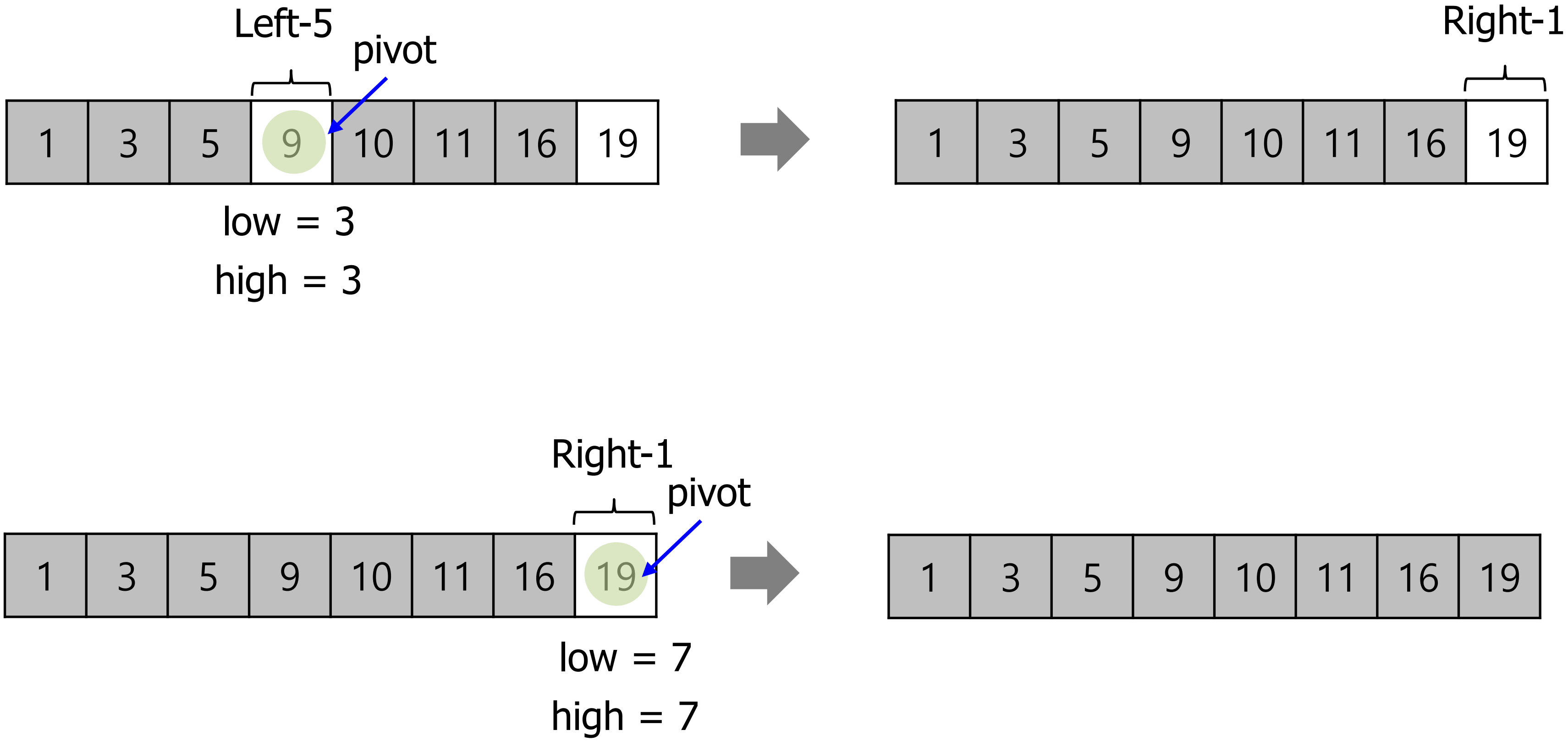
Representative Advanced Sorting Algorithms

- Quick sort
  - Example) Last element as a pivot



Representative Advanced Sorting Algorithms

- ❖ Quick sort
  - Example) Last element as a pivot



Left-5

pivot

1 3 5 9 10 11 16 19

low = 3

high = 3

Right-1

1 3 5 9 10 11 16 19

Right-1

pivot

1 3 5 9 10 11 16 19

low = 7

high = 7

1 3 5 9 10 11 16 19

Representative Advanced Sorting Algorithms

- Quick sort
  - Psuedo code)

```
#include <stdio.h>

void quick_sort(int arr[], int low, int high){
   if (low < high){
      int pivot = partition(arr, low, high);
      quick_sort(arr, low, pivot – 1);
      quick_sort(arr, pivot + 1, high);
   }
}
```

Representative Advanced Sorting Algorithms

- Quick sort
  - Psuedo code)

```
void swap(int *a, int *b){
   int temp = *a;
   *a = *b;
   *b = temp;
}
```

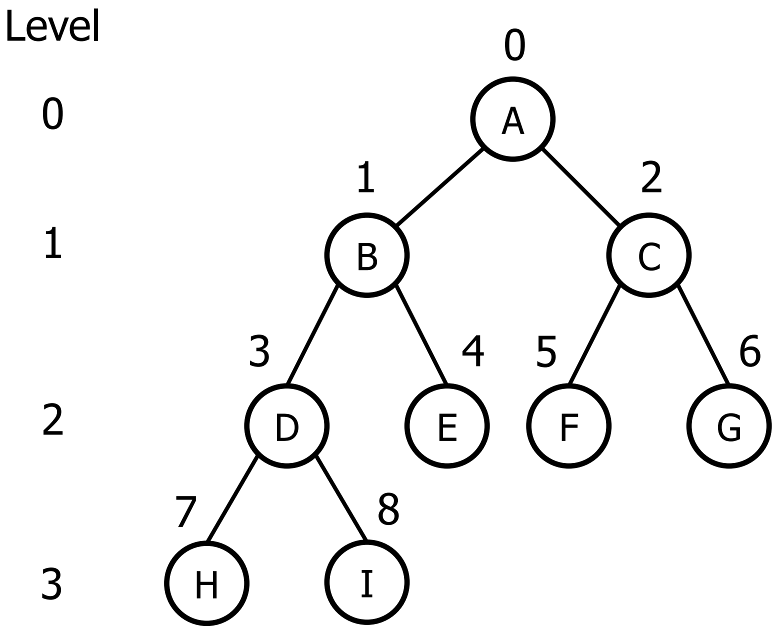
```
int partition(int arr[], int low, int high){
   int pivot = arr[high];
   int i = low – 1;
   for(int j = low; j <= high; j++){
      if (arr[j] < pivot){
         i++;
         swap(&arr[i], &arr[j]);
      }
   }
   swap(&arr[i + 1], &arr[high]);
   return i + 1;
}
```

Representative Advanced Sorting Algorithms

- Heap sort
  - Binary heap data structure
  - In-place comparison sort
  - Time complexity: $O(n \log n)$
  - Methodology:
    1) Construct a heap from the list
    2) Swap root element of the heap with last element of the heap
    3) Remove the last element of the heap
    4) Heapify the remaining elements of the heap
    5) Repeat 2) – 4) steps until the heap contains only one element

Representative Advanced Sorting Algorithms

- Heap sort
  - What is the heap?
    - Complete binary tree
      - Binary tree where all levels except the last one are fully filled
    - Max heap: each node is always greater than its children
    - Min heap: each node is always less than its children



0	1	2	3	4	5	6	7	8
A	B	C	D	E	F	G	H	I

The children of the i-th node

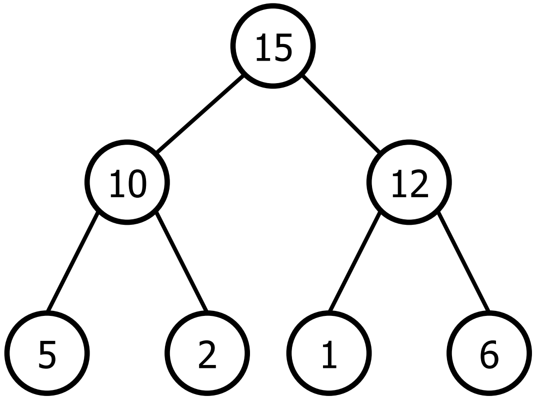
Left child: (2 * i + 1)-th node

Right child: (2 * i + 2)-th node

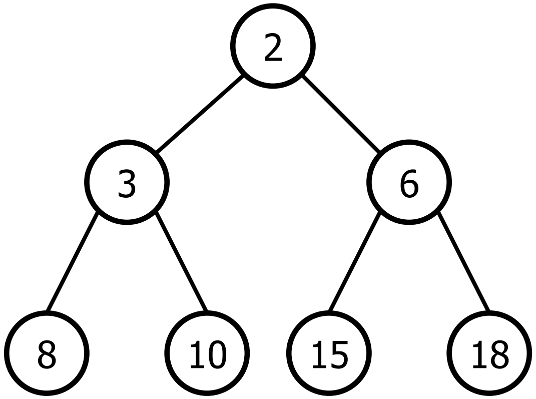
Representative Advanced Sorting Algorithms

- Heap sort
  - What is the heap? (cont’d)

Max heap



Min heap

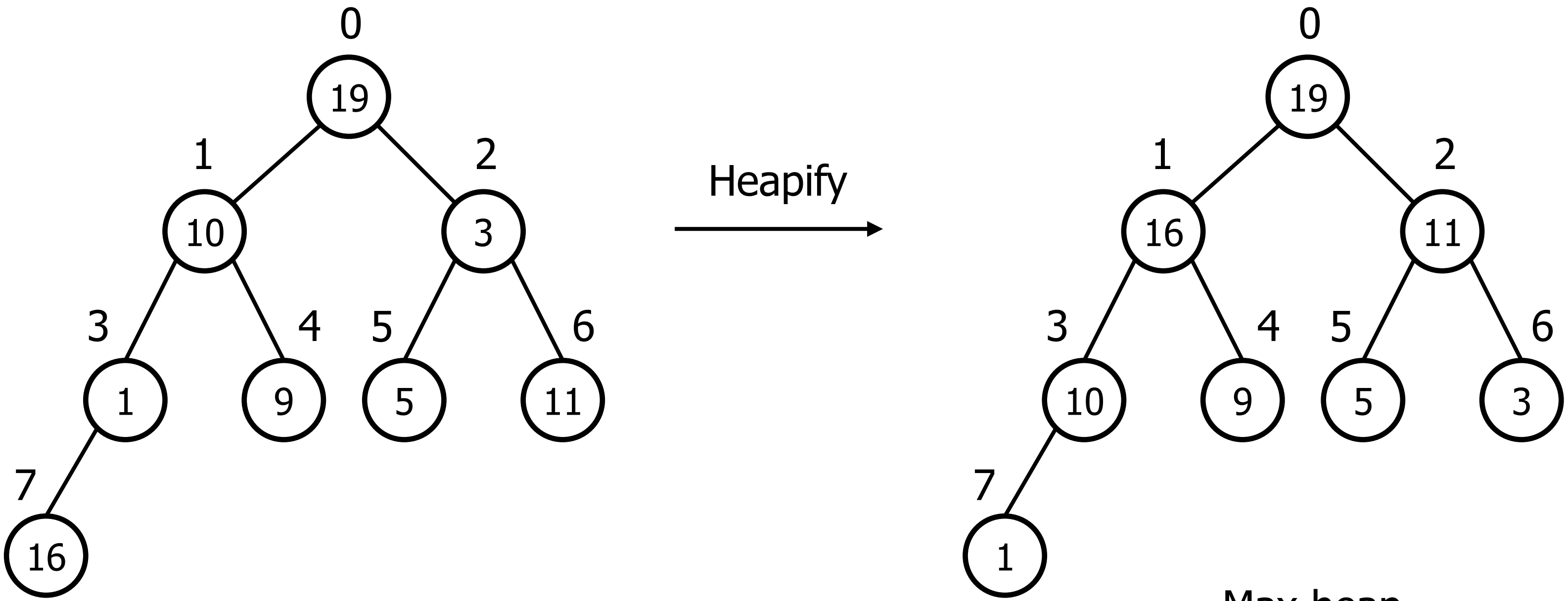


Representative Advanced Sorting Algorithms

- Heap sort
  - Example)

19	10	3	1	9	5	11	16

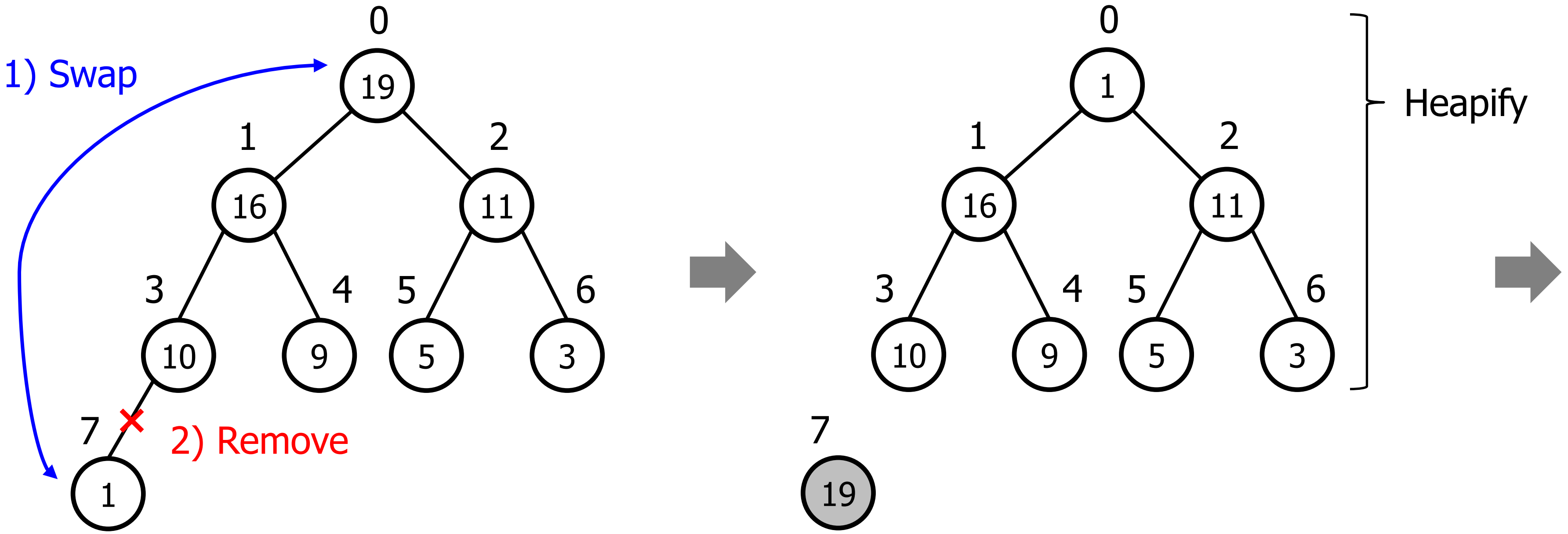
Construct a heap



Max heap

Representative Advanced Sorting Algorithms

- Heap sort
  - Example)

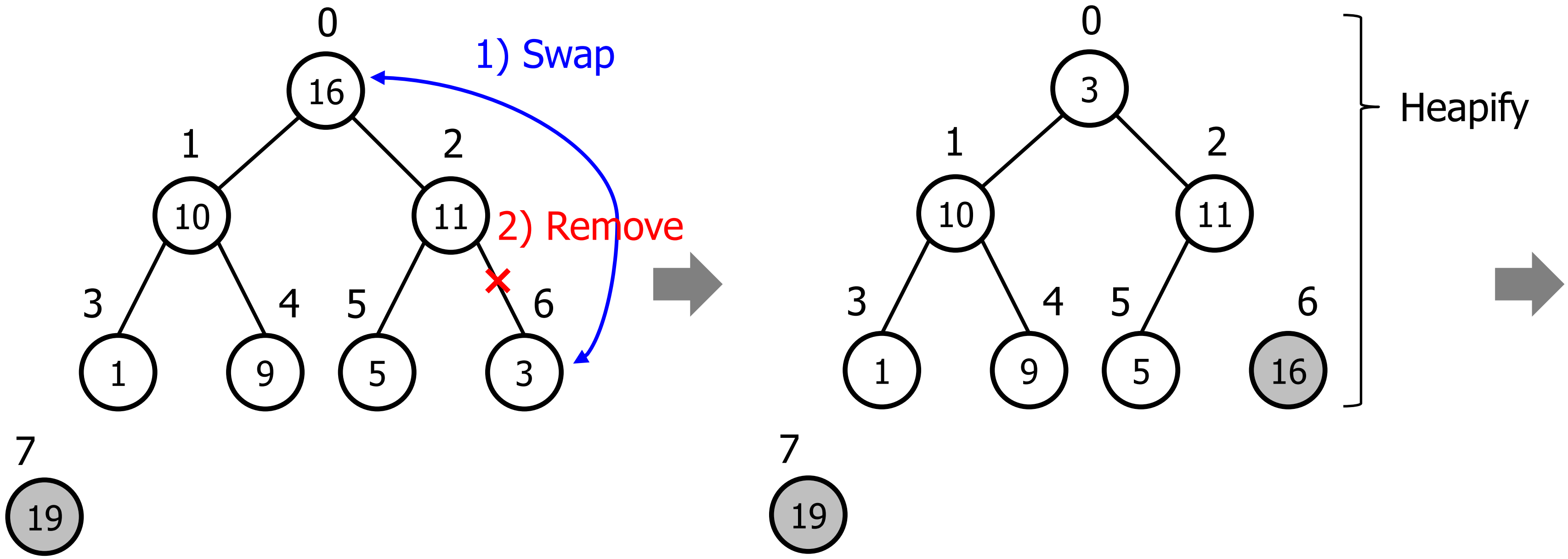


19	16	11	10	9	5	3	1

1	16	11	10	9	5	3	19

Representative Advanced Sorting Algorithms

- ❖ Heap sort
  - ▪ Example)

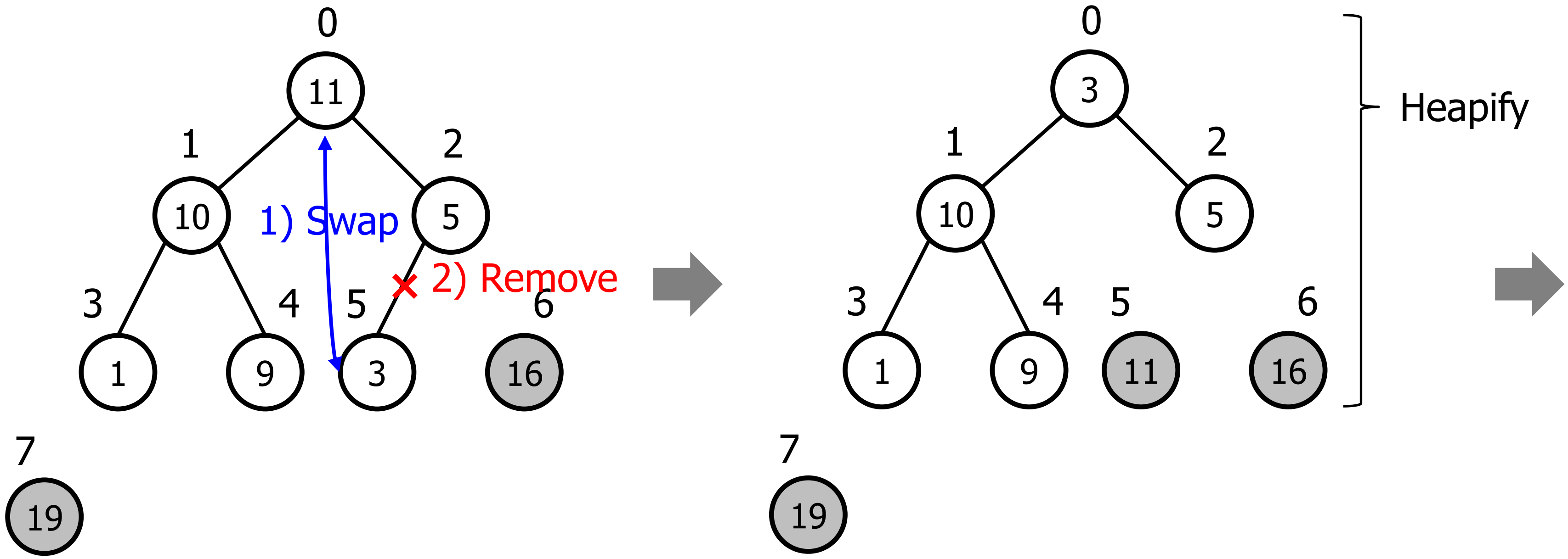


16	10	11	1	9	5	3	19

3	10	11	1	9	5	16	19

Representative Advanced Sorting Algorithms

- Heap sort
  - Example)

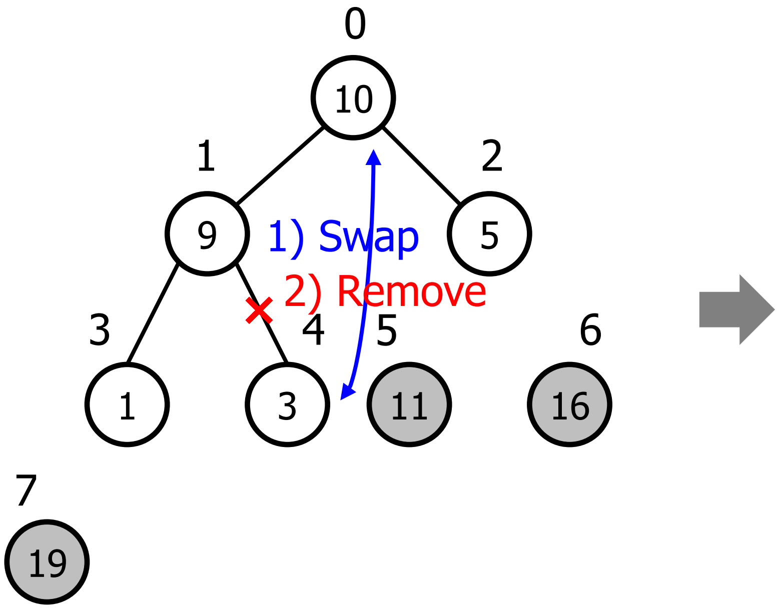


11	10	5	1	9	3	16	19

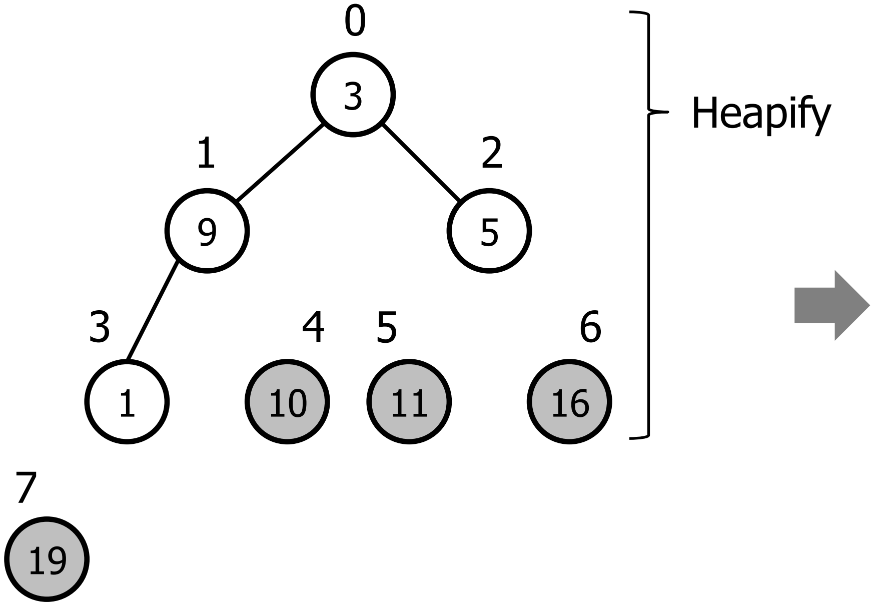
3	10	5	1	9	11	16	19

Representative Advanced Sorting Algorithms

- Heap sort
  - Example)



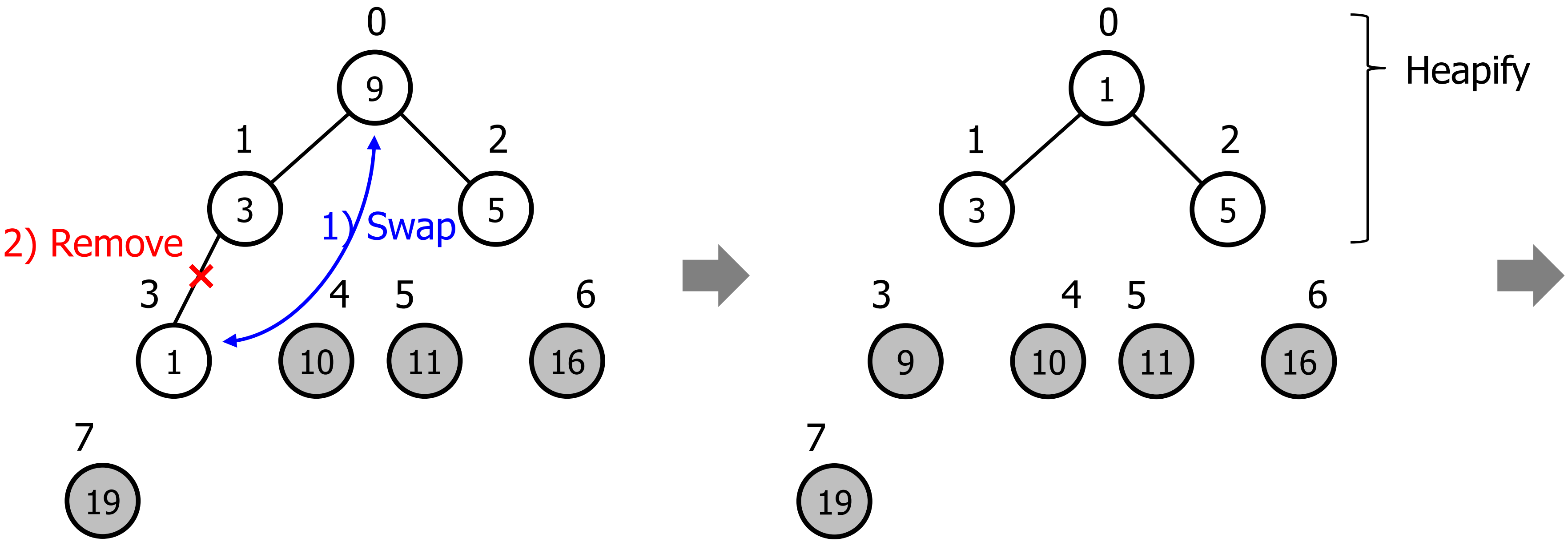
10	9	5	1	3	11	16	19



3	9	5	1	10	11	16	19

Representative Advanced Sorting Algorithms

- Heap sort
  - Example)

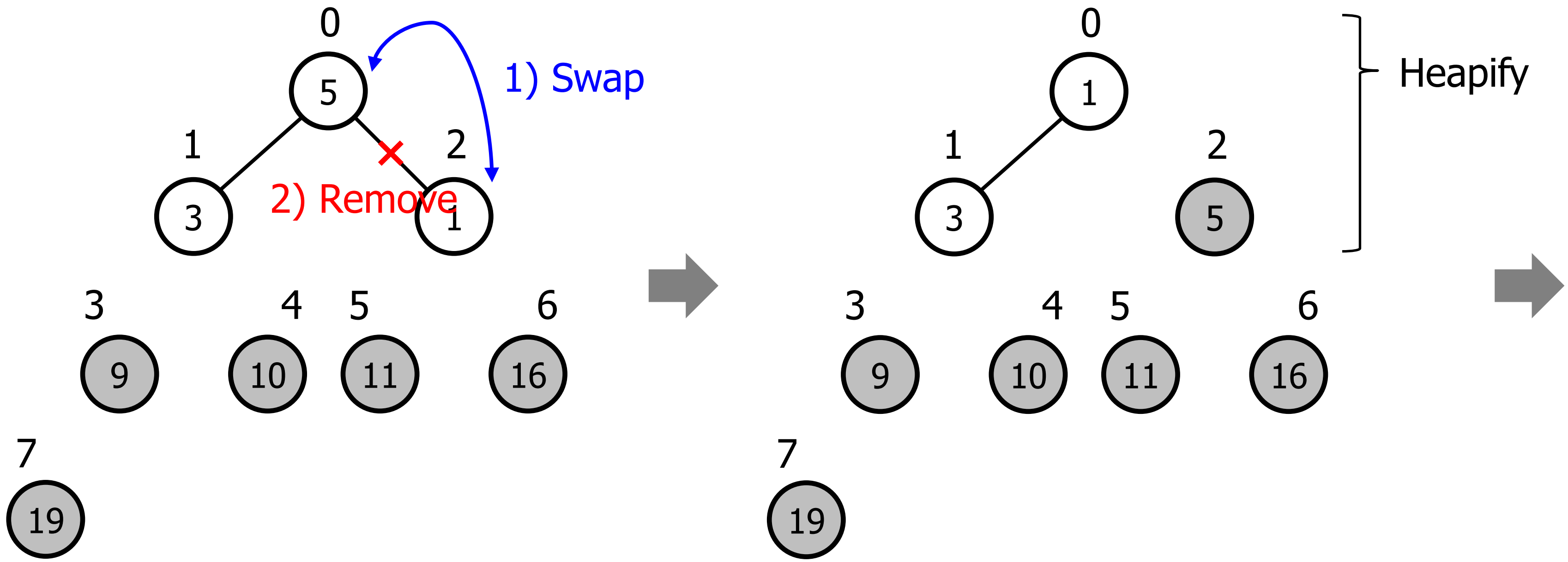


9	3	5	1	10	11	16	19

1	3	5	9	10	11	16	19

Representative Advanced Sorting Algorithms

- Heap sort
  - Example)

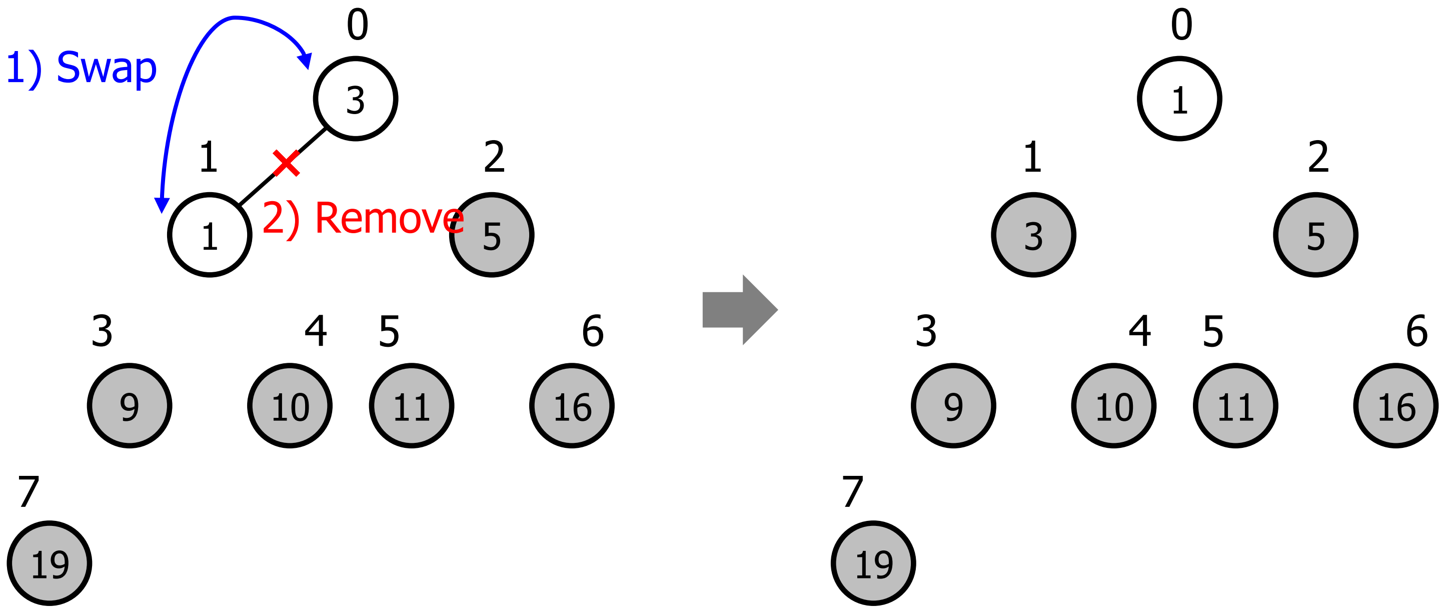


5	3	1	9	10	11	16	19

1	3	5	9	10	11	16	19

Representative Advanced Sorting Algorithms

- Heap sort
  - Example)



3	1	5	9	10	11	16	19

1	3	5	9	10	11	16	19

Representative Advanced Sorting Algorithms

- Heap sort
  - Psuedo code)

```
#include <stdio.h>

void heap_sort(int arr[], int n){
   for(int i = n / 2 – 1; i >= 0; i--)
      heapify(arr, n, i);
   for(int i = n – 1; i >= 0; i--){
      swap(&arr[0], &arr[i]);
      heapify(arr, i, 0);
   }
}

void swap(int* a, int* b){
   int temp = *a;
   *a = *b;
   *b = temp
}
```

```
void heapify(int arr[], int n, int i){
   int largest = i;
   int left = 2 * i + 1;
   int right = 2 * i + 2;
   if (left < n && arr[left] > arr[largest])
      largest = left;
   if (right < n && arr[right] > arr[largest])
      largest = right;
   if (largest != i){
      swap(&arr[i], &arr[largest]);
      heapify(arr, n, largest)
   }
}
```

Representative Advanced Sorting Algorithms

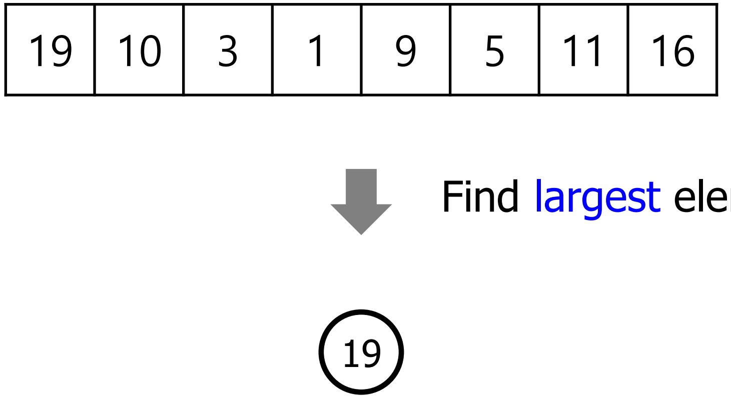
- Radix sort
  - Linear sorting algorithm
    - It sorts elements by processing them digit by digit
  - Out-of-place comparison sort
  - Time complexity: $\mathrm{O}(d * (n + b))$
    - d: the number of digits
    - n: the number of elements
    - b: the base of the number system being used

Representative Advanced Sorting Algorithms

- Radix sort
  - Methodology:
    1) Find largest element in the array to determine the number of digits
    2) Sort elements iteratively from Least Significant Digit (LSD) to Most Significant Digit (MSD)

Representative Advanced Sorting Algorithms

- Radix sort
  - Example)



19	10	3	1	9	5	11	16

Find largest element ($O(n)$)

Number of digits: 2

Ones and tens places
(LSD) (MSD)

Representative Advanced Sorting Algorithms

❖ Radix sort

- Example)

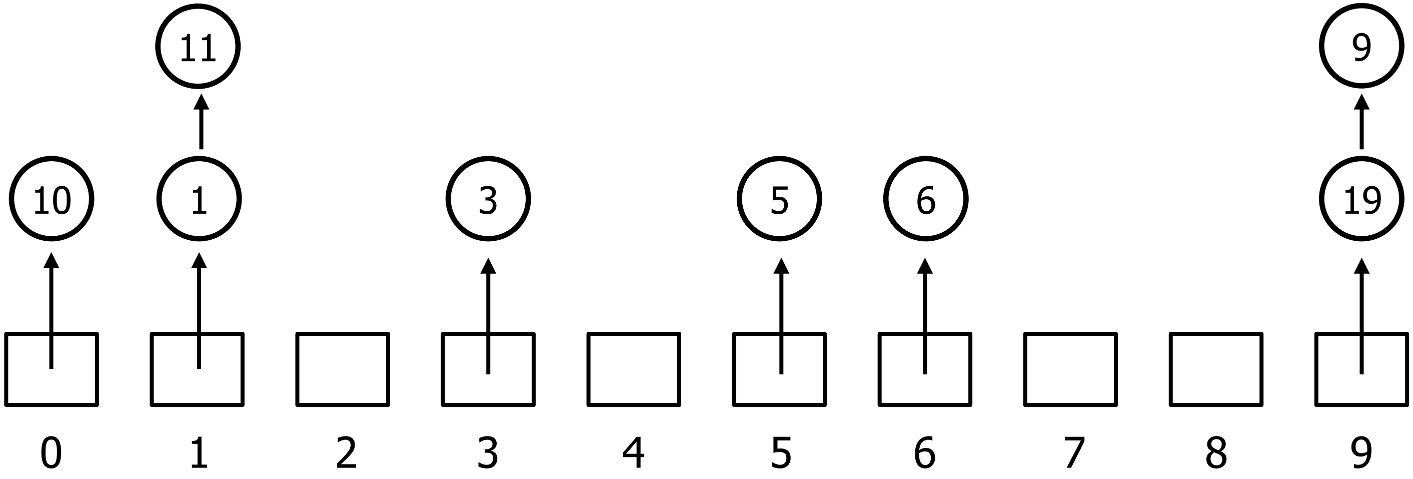
The base of the number system: Decimal (0~9) (= the number of bins)

1-iter.: ones place

19	10	3	1	9	5	11	16

⟶

10	1	11	3	5	6	19	9

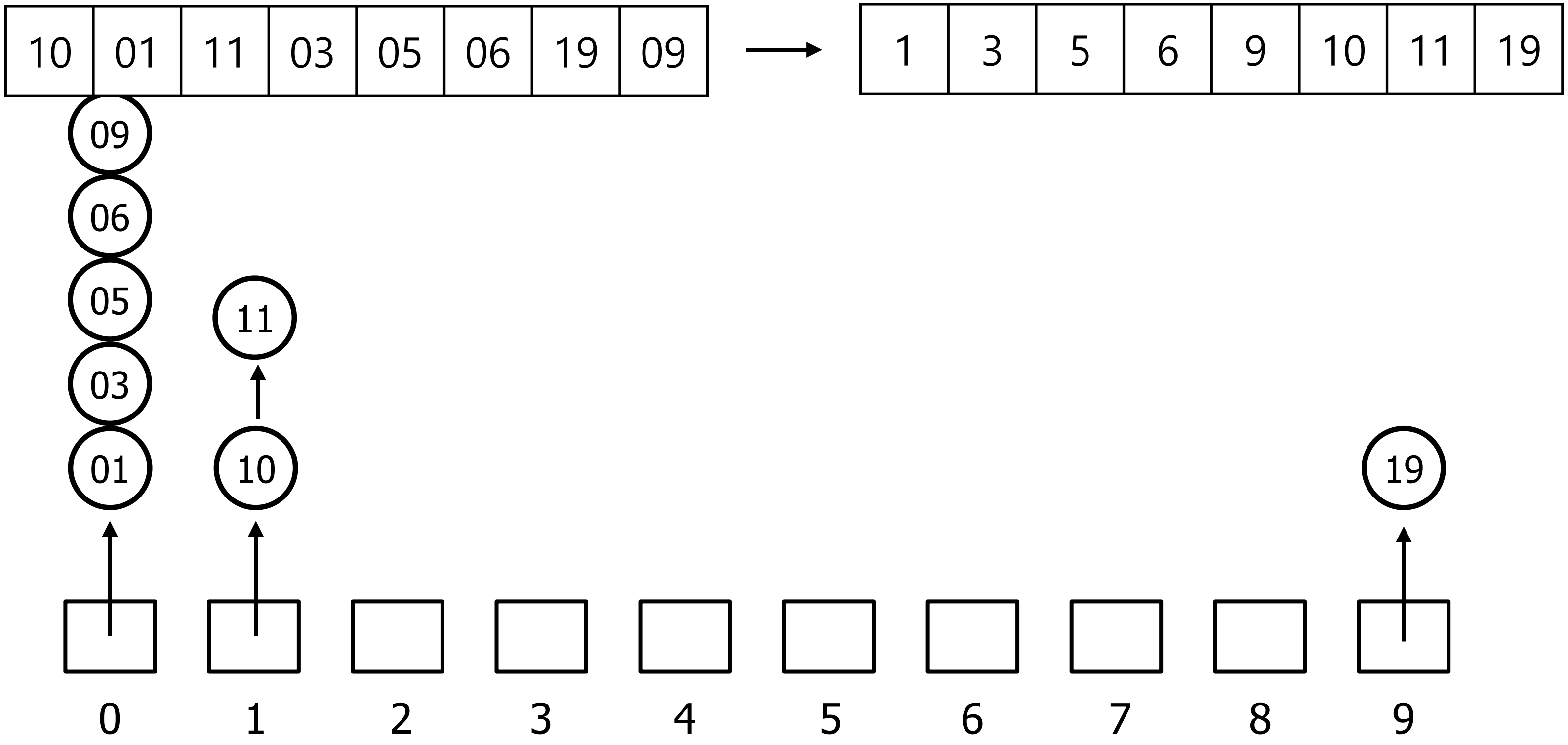


Representative Advanced Sorting Algorithms

- ❖ Radix sort
  - ▪ Example)

    The base of the number system: Decimal (0~9) (= the number of bins)

    2-iter.: tens place



10	01	11	03	05	06	19	09

→

1	3	5	6	9	10	11	19

Representative Advanced Sorting Algorithms

- Radix sort
  - Psuedo code)

```
#include <stdio.h>

void radix_sort(int arr[], int n){
   int m = getMax(arr, n);
   for(int exp = 1; m / exp > 0; exp *= 10)
      count_sort(arr, n, exp);
}


void getMax(int arr[], int n){
   int mx = arr[0];
   for(int i = 1; i < n; i++)
      if (arr[i] > mx)
         mx = arr[i];
   return mx;
}
```

```
void count_sort(int arr[], int n, int exp){
   int* output = malloc(sizeof(int) * n);
   int i, count[10] = {0};
   for(i = 0; i < n; i++)
      count[(arr[i] / exp) % 10]++;
   for(i = 1; i < 10; i++)
      count[i] += count[i – 1];
   for(i = n – 1; i >= 0; i--){
      output[count[(arr[i] / exp) % 10] – 1] = arr[i];
      count[(arr[i] / exp) % 10]--;
   }
   for(i = 0; i < n; i++)
      arr[i] = output[i];
   free(output);
}
```

Summary

- Speed-up factors of sorting algorithm
  - The number of comparison
  - The number of swaps

- Representative advanced sorting algorithms
  - Shell sort
  - Merge sort
  - Quick sort
  - Heap sort
  - Radix sort
  - Bucket sort
  - Tim sort

Questions?

SEE YOU NEXT TIME!