Lecture 7: Sorting algorithm (Part. 1)

Algorithm

Jeong-Hun Kim

Remind

- Knapsack Problems
  - 0-1 knapsack
  - Fractional knapsack
- Interval scheduling
  - Maximum number of tasks that can be processed
- Scheduling all requests
  - Minimum number of classrooms required to conduct all lectures
- Huffman encoding
  - Minimization of binary string length

Table of Contents

Part 1

DATA SORTING

Data Sorting

- Sorting problem
  - Input: A list consisting of n numbers $\langle a_1, a_2, a_3, \ldots, a_{n-1}, a_n \rangle$
  - Output: A list rearranged in ascending order $\langle a'_1, a'_2, a'_3, \ldots, a'_{n-1}, a'_n \rangle$, where $a'_1 \leq a'_2 \leq \cdots \leq a'_n$

- What is the sorting algorithm?
  - Arranging n elements in order
    - E.g., Sorted by height, sorted by name, sorted by number, etc.
  - Time complexity: typically between $O(n^2)$ and $O(n \log n)$
    - When the input satisfies specific conditions, time complexity is $O(n)$

Data Sorting

- Representative sorting algorithms
  - Basic sorting algorithms
    - Sorting algorithms that have an time complexity of $\Theta(n^2)$
      - n is the number of elements
    - E.g., Selection sort, Bubble sort, Insertion sort
  - Advanced sorting algorithms
    - Sorting algorithms that have an time complexity of $O(n \log n)$
    - E.g., Shell sort, Merge sort, Quick sort, Heap sort, Radix sort, Bucket sort, Tim sort

Part 2

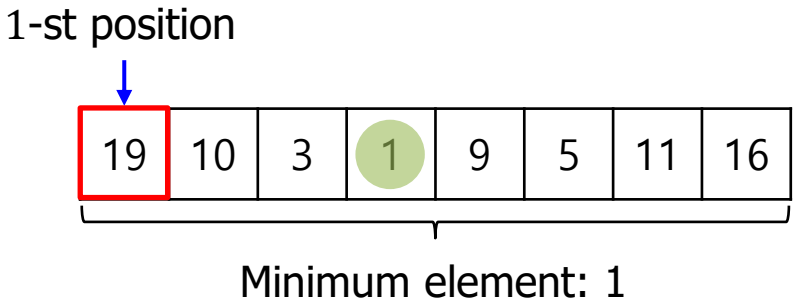
REPRESENTATIVE BASIC SORTING ALGORITHMS

Representative Basic Sorting Algorithms

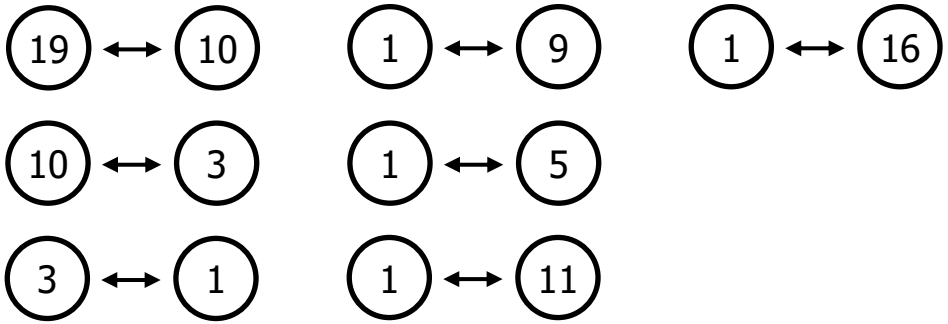
- Selection sort
  - The simplest sorting algorithm
  - In-place comparison sort
    - In-place: no need for a new list (↔ out-of-place)
    - Useful when memory space is limited
  - Time complexity: $O(n^2)$
  - Methodology:
    1) Move to the i-th position in the list
    2) Find the minimum element among elements from i to n
    3) Swap the i-th element with the minimum element
    4) Repeat 1) – 3) steps until $i = n$

Representative Basic Sorting Algorithms

- Selection sort
  - Example)

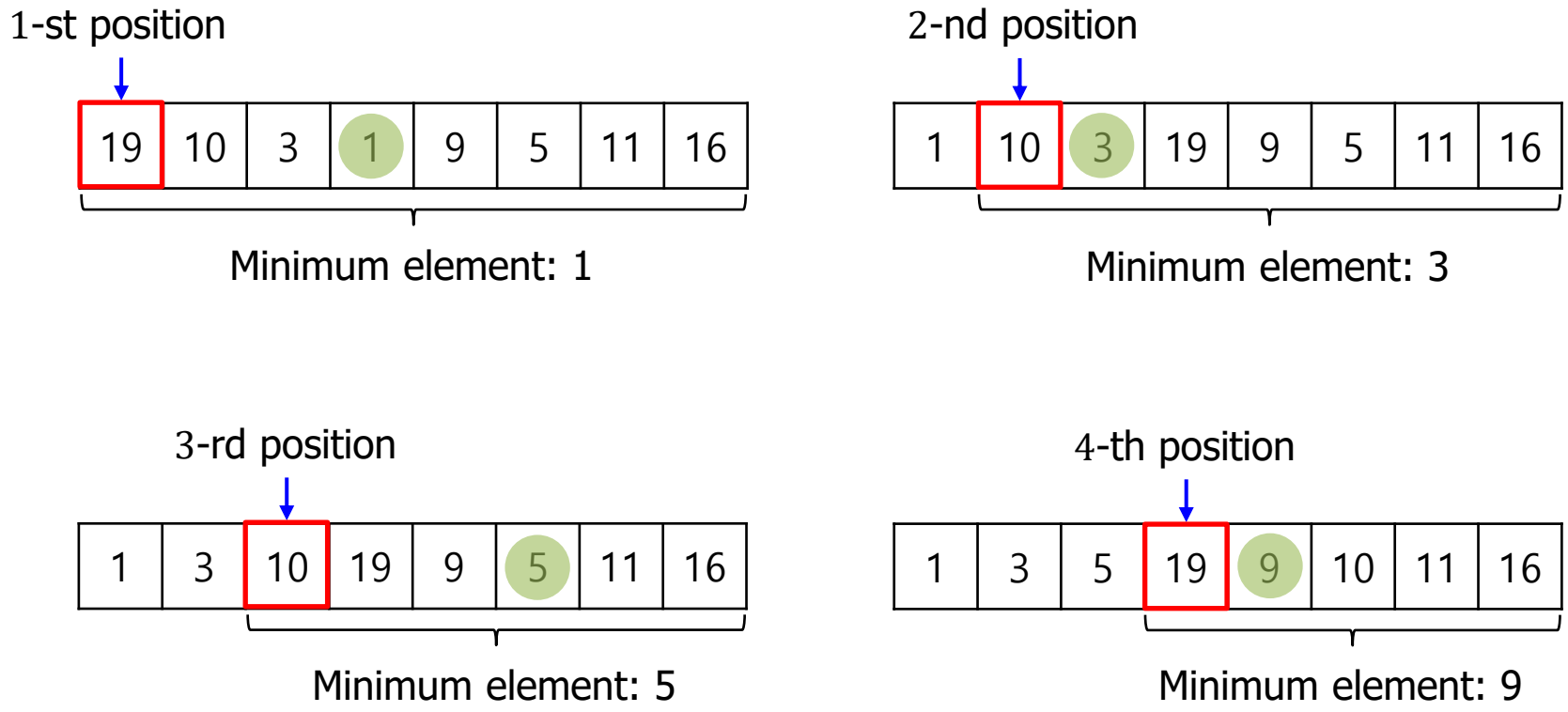


Comparisons:



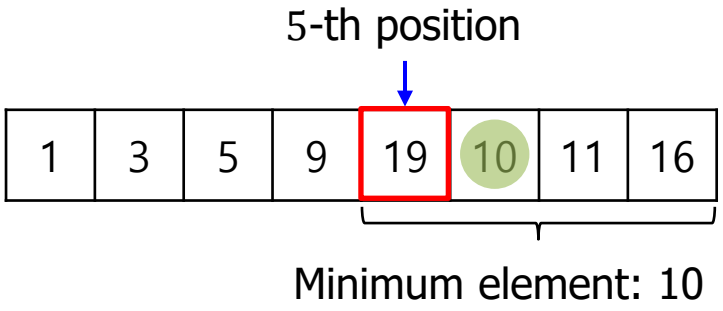
Representative Basic Sorting Algorithms

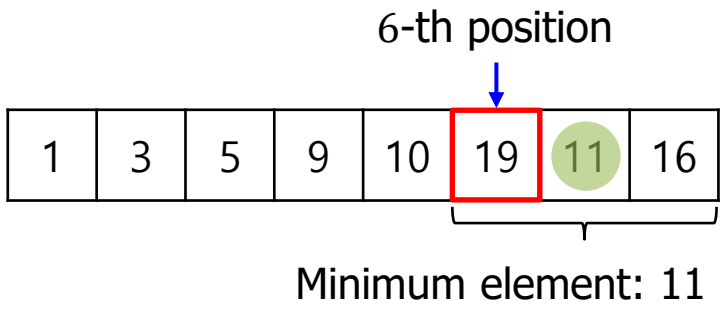
- Selection sort
  - Example (cont'd))

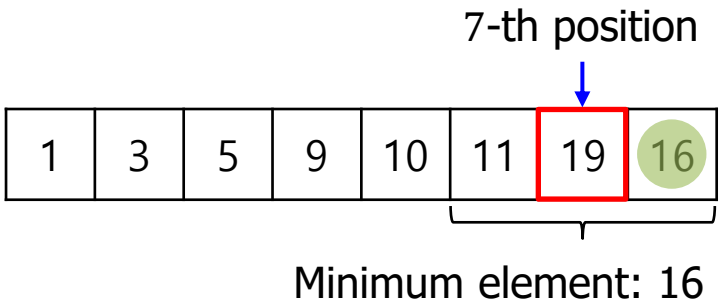


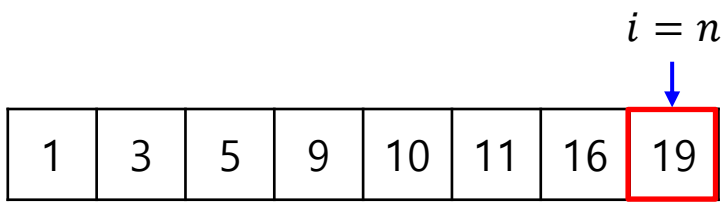
Representative Basic Sorting Algorithms

- Selection sort
  - Example (cont'd))









Representative Basic Sorting Algorithms

- Selection sort
  - Psuedo code)

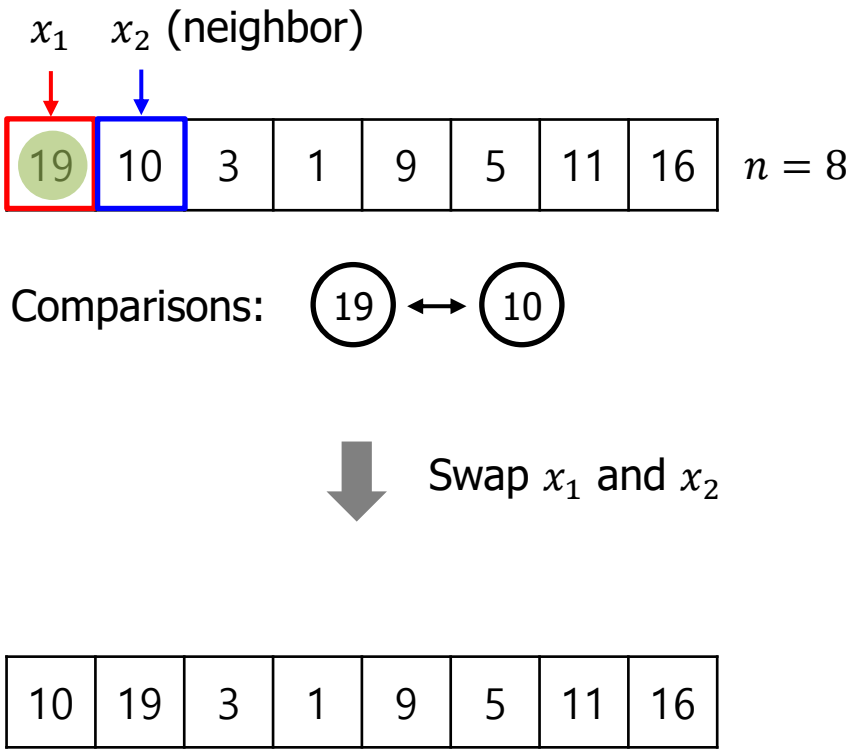
```
#include <stdio.h>

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

void selection_sort(int arr[], int n){
   int i, j, min_idx;
   for (i = 0; i < n – 1; i++){
      min_idx = i;
      for (j = i + 1; j < n; j++){
         if (arr[j] < arr[min_idx])
            min_idx = j;
      }
      if (min_idx != i)
         swap(arr[min_idx], arr[i])
   }
}
```

Representative Basic Sorting Algorithms

- Bubble sort
  - Traverse method
  - In-place comparison sort
  - It uses neighbor element
  - Time complexity: $O(n^2)$
  - Methodology:
    1) Compare i-th element x_i with its neighbor ($i+1$-th) element x_{i+1}
    2) If $x_i > x_{i+1}$, swap the positions of x_i and x_{i+1}
    3) Repeat 1-2) steps until $i + 1 = n$
    4) Decrease n by 1, then repeat steps 1) – 3)
    5) Terminate when $n = 1$

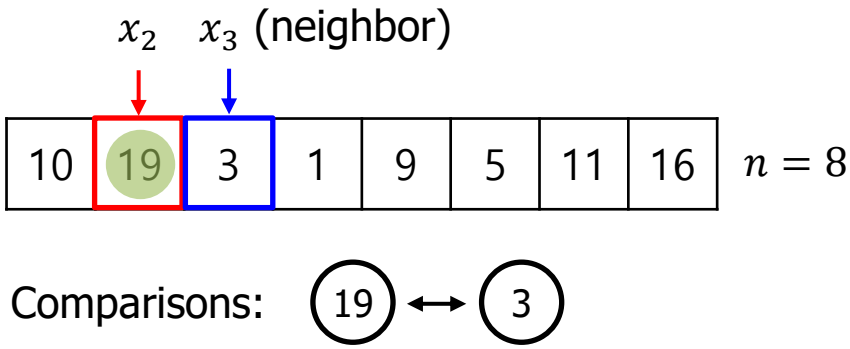
Representative Basic Sorting Algorithms

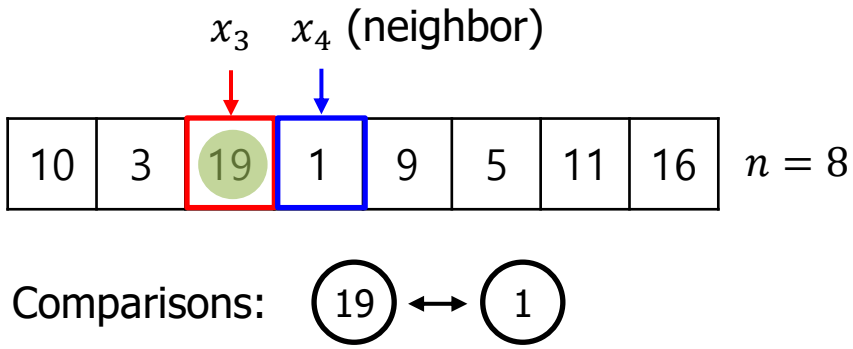
- Bubble sort
  - Example)

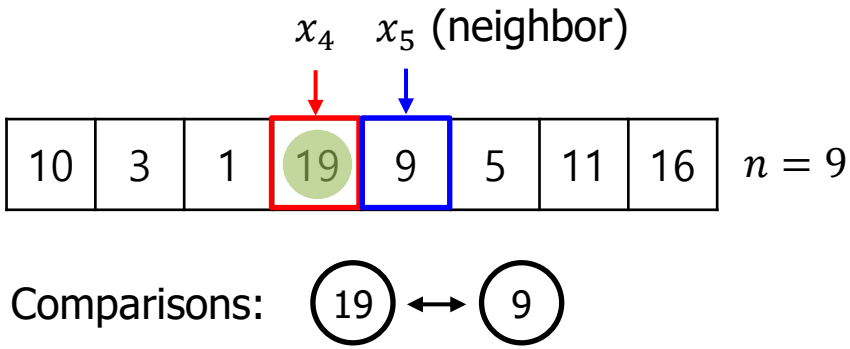


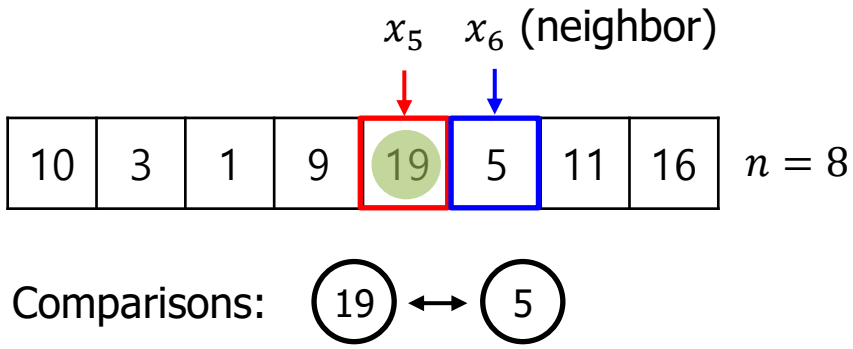
Representative Basic Sorting Algorithms

- Bubble sort
  - Example)



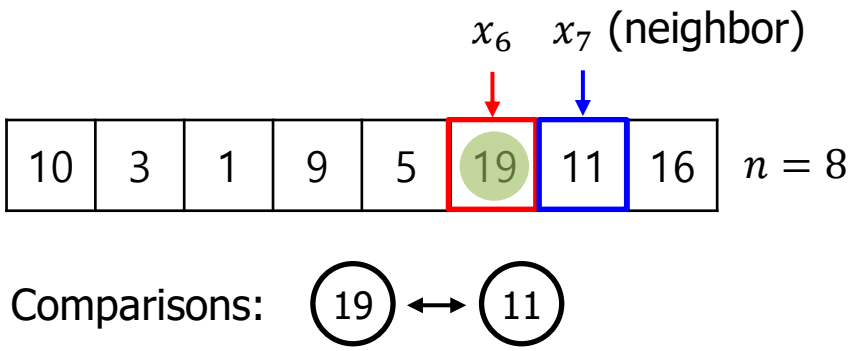


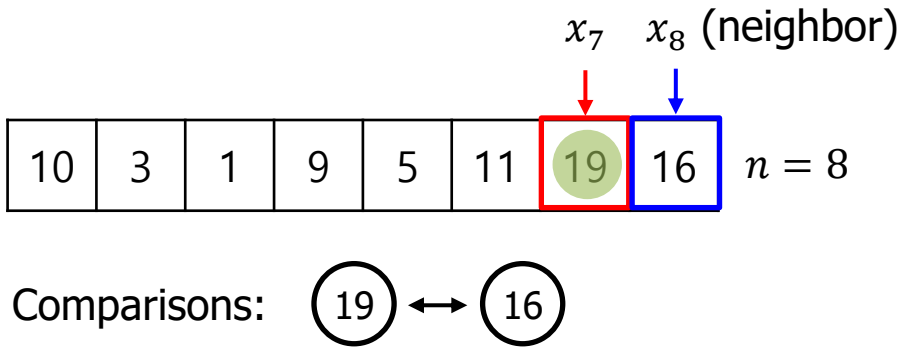




Representative Basic Sorting Algorithms

- Bubble sort
  - Example)

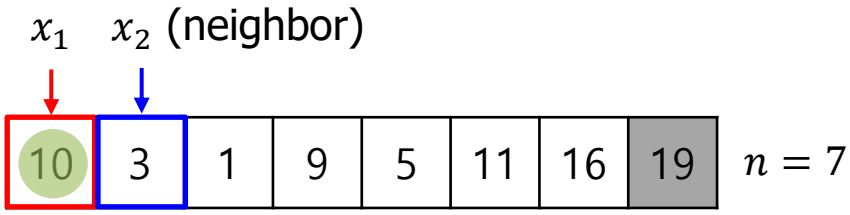




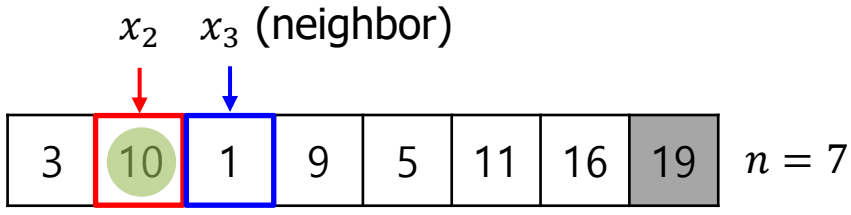
10	3	1	9	5	11	16	19

Representative Basic Sorting Algorithms

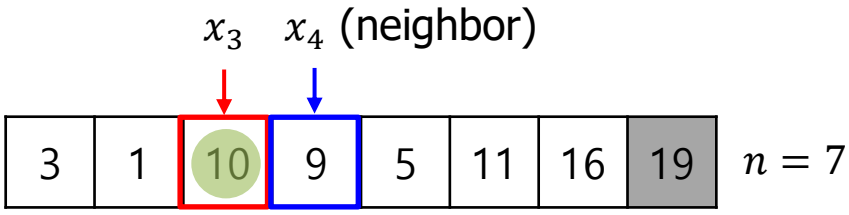
- Bubble sort
  - Example)



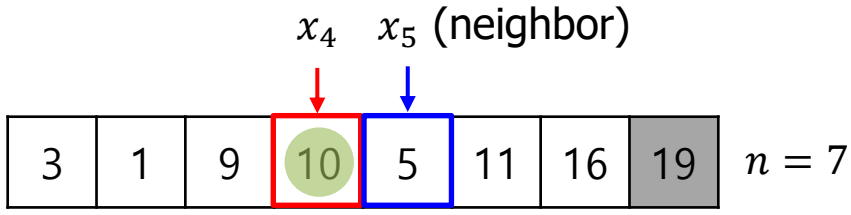
Comparisons: 10 ↔ 3



Comparisons: 10 ↔ 1



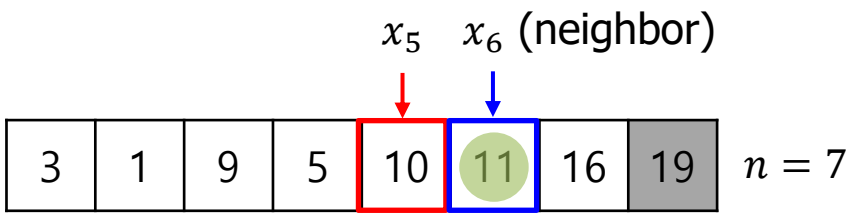
Comparisons: 10 ↔ 9



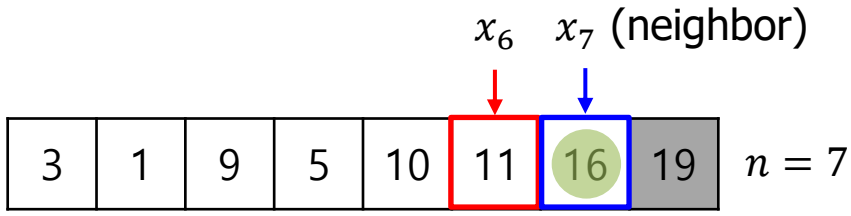
Comparisons: 10 ↔ 5

Representative Basic Sorting Algorithms

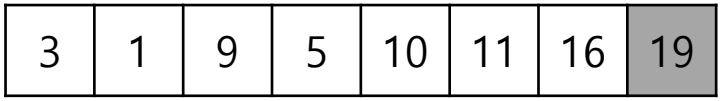
- Bubble sort
  - Example)



Comparisons: 10 ↔ 11

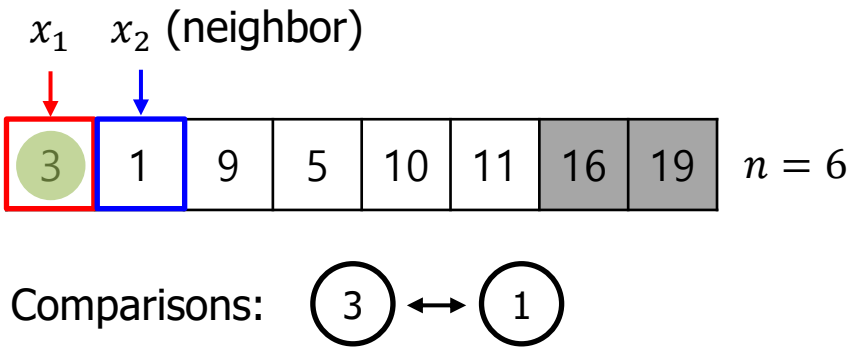


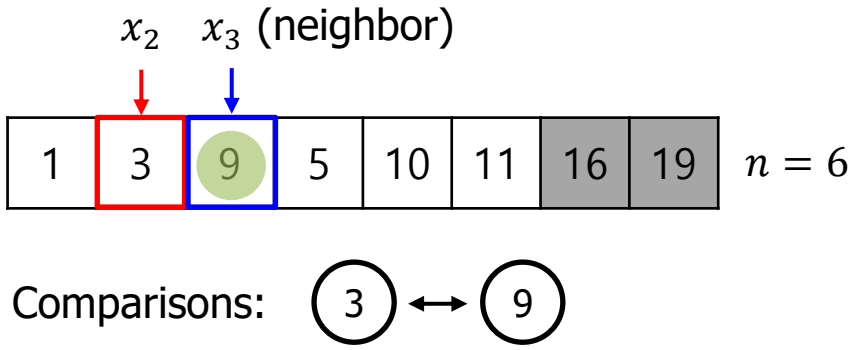
Comparisons: 11 ↔ 16

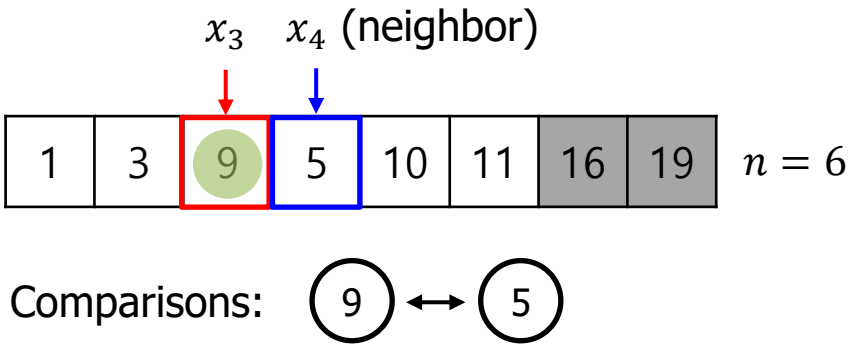


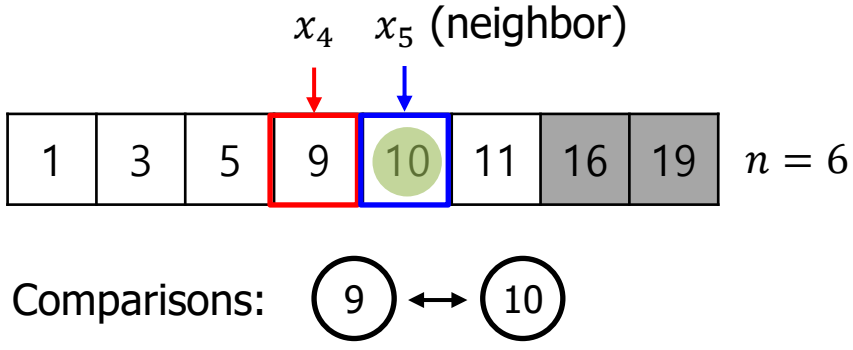
Representative Basic Sorting Algorithms

- Bubble sort
  - Example)



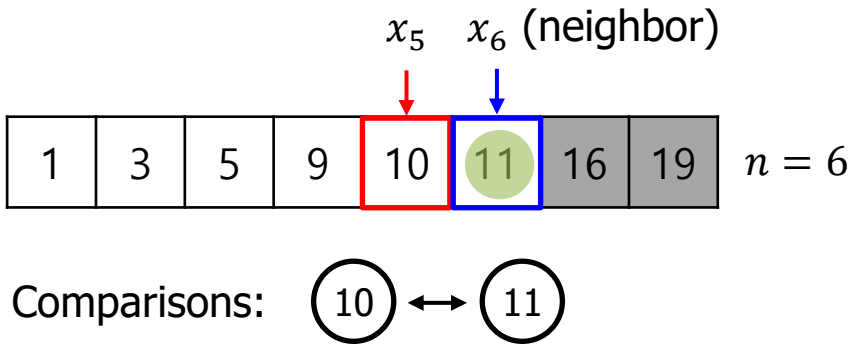


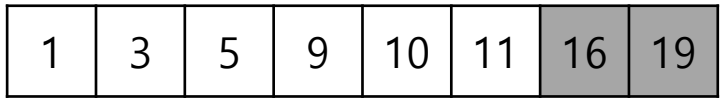




Representative Basic Sorting Algorithms

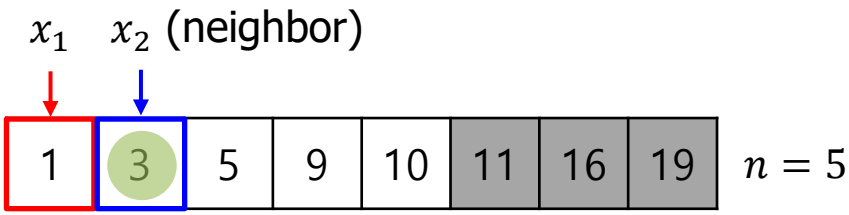
- Bubble sort
  - Example)



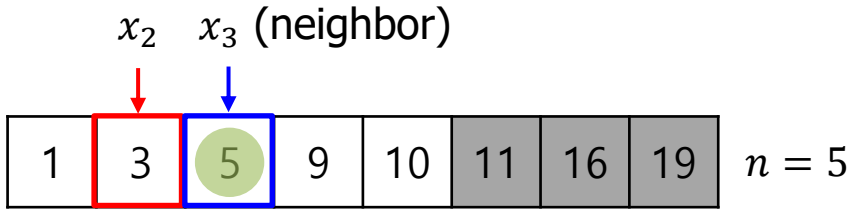


Representative Basic Sorting Algorithms

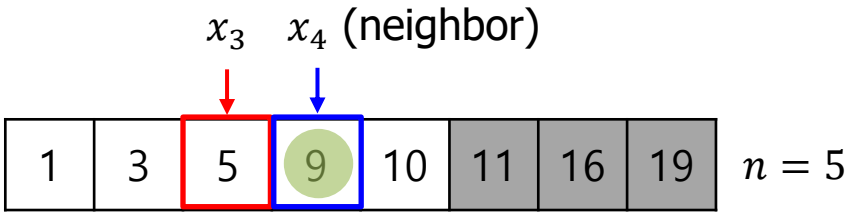
- Bubble sort
  - Example)



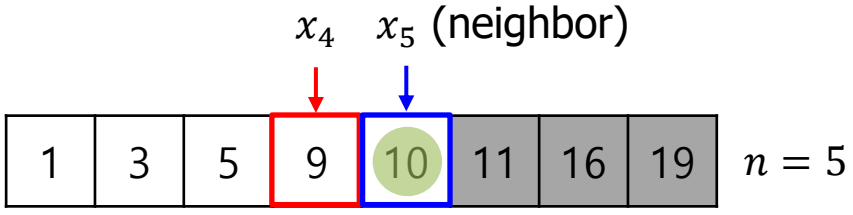
Comparisons: 1 ↔ 3



Comparisons: 3 ↔ 5



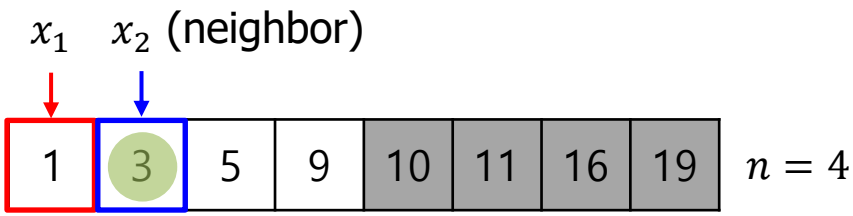
Comparisons: 5 ↔ 9



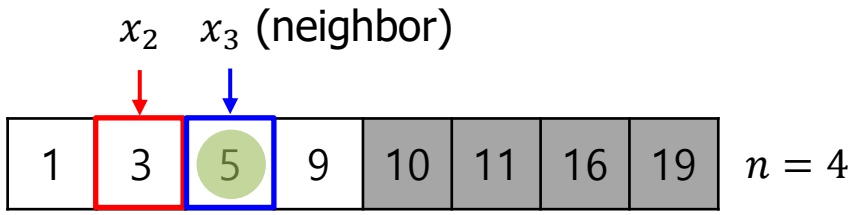
Comparisons: 9 ↔ 10

Representative Basic Sorting Algorithms

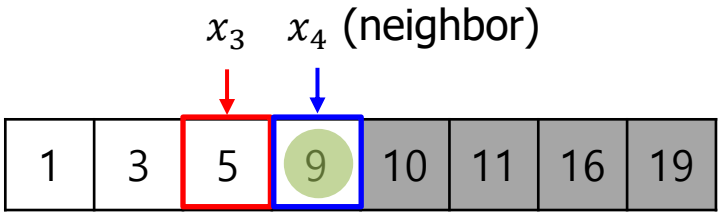
- Bubble sort
  - Example)



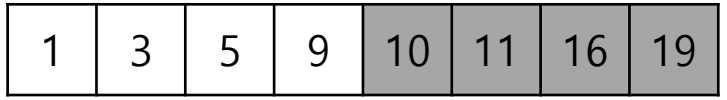
Comparisons: 1 ↔ 3



Comparisons: 3 ↔ 5

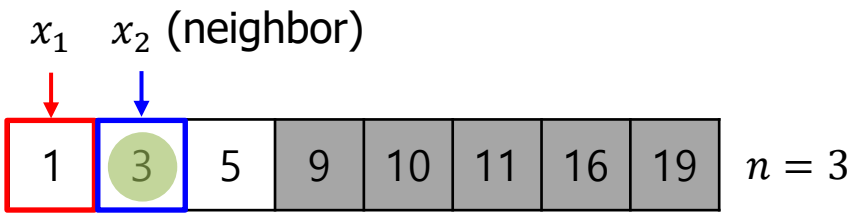


Comparisons: 5 ↔ 9

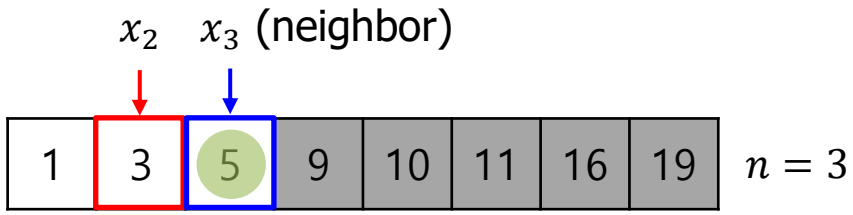


Representative Basic Sorting Algorithms

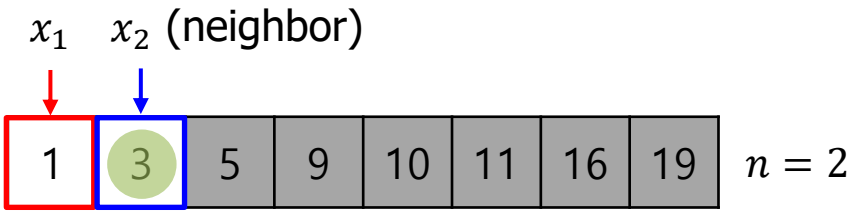
- Bubble sort
  - Example)



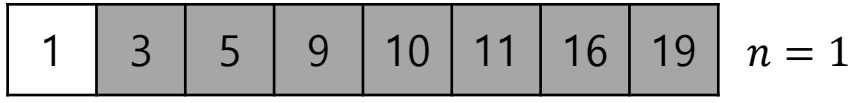
Comparisons: 1 ↔ 3



Comparisons: 3 ↔ 5



Comparisons: 1 ↔ 3



Representative Basic Sorting Algorithms

- Bubble sort
  - Psuedo code)

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

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

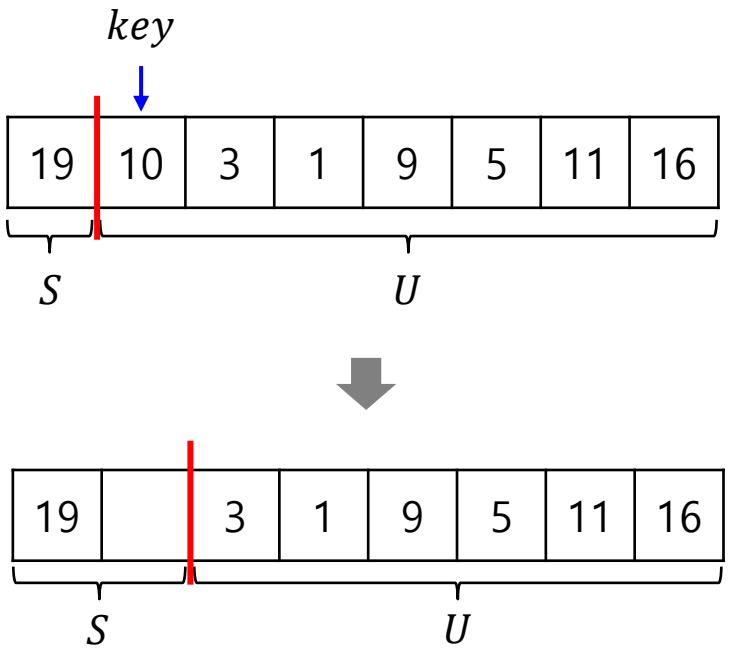
```
void bubble_sort(int arr[], int n){
   int i, j;
   bool swapped;
   for (i = 0; i < n – 1; i++){
      swapped = false;
      for (j = 0; j < n – i - 1; j++){
         if (arr[j] < arr[j + 1]){
            swap(&arr[j], &arr[j + 1]);
            swapped = true;
         }
      }
      if (swapped == false)
         break;
   }
}
```

Representative Basic Sorting Algorithms

- Insertion sort
  - It uses two virtual subsets:
    - Sorted subset S
    - Unsorted subset U
  - In-place comparison sort
  - Time complexity: $O(n^2)$
  - Methodology:
    1) $i = |S|$
    2) $key = (i+1)$-th element x_{i+1}
    3) Find the insertion position of key:
       - Compare i-th element x_i and key
       - If $x_i > key$, then $x_{i+1} = x_i$ and i is decreased by 1
       - Otherwise, $x_{i+1} = key$
       - Repeat 3) step
    4) Repeat 1) – 4) steps until $U = \emptyset$

Representative Basic Sorting Algorithms

- Insertion sort
  - Example)

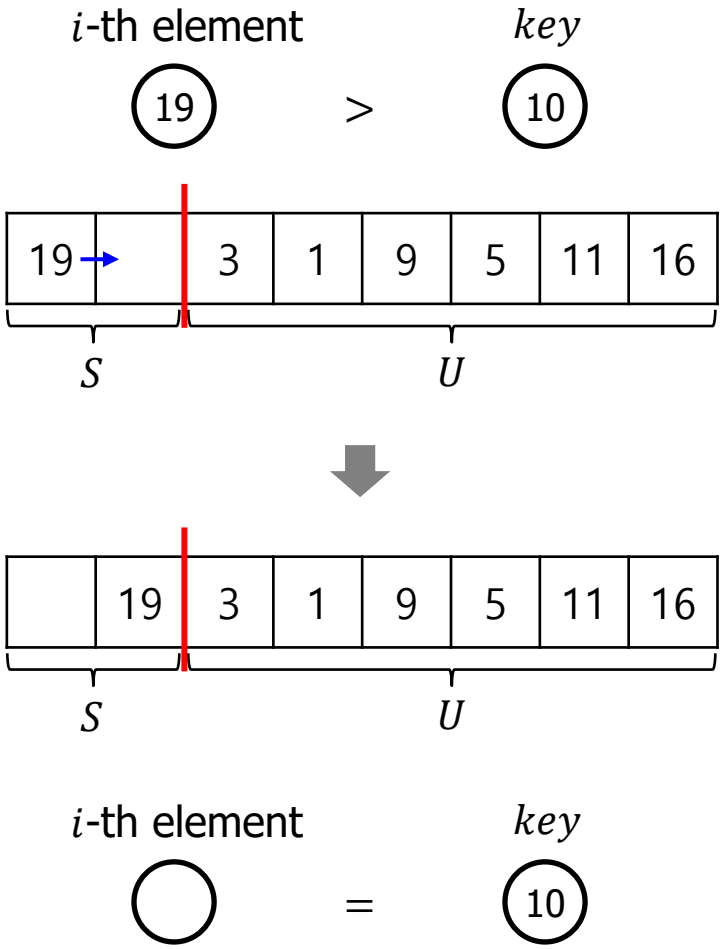


i-th element key

Comparisons: 19 ↔ 10

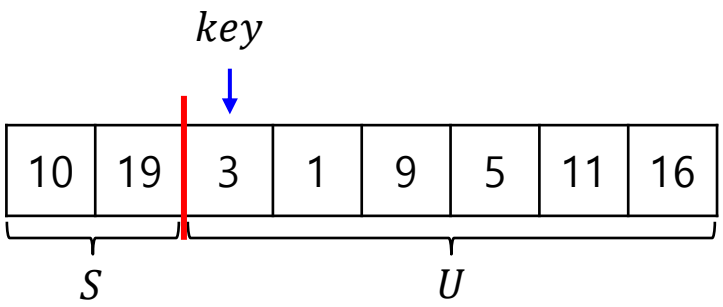
Representative Basic Sorting Algorithms

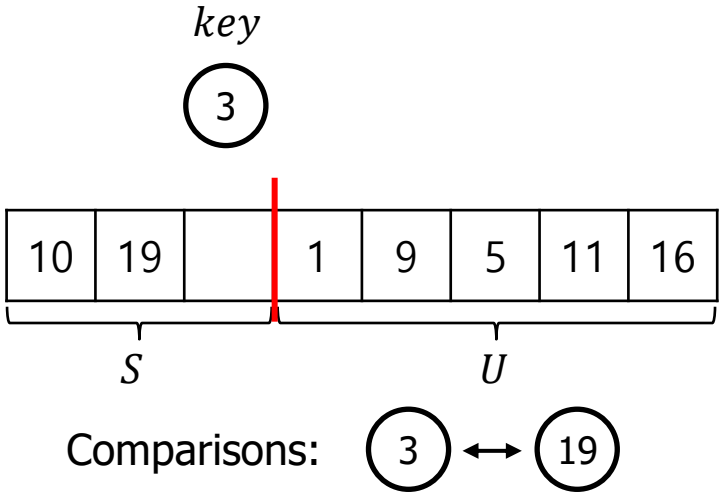
- Insertion sort
  - Example)

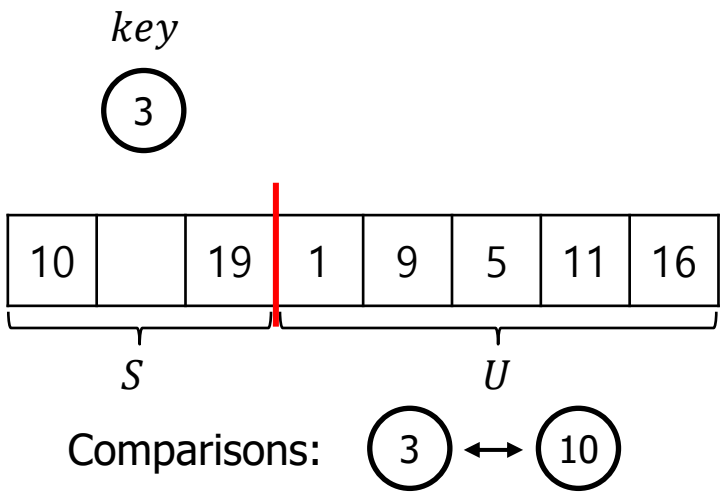


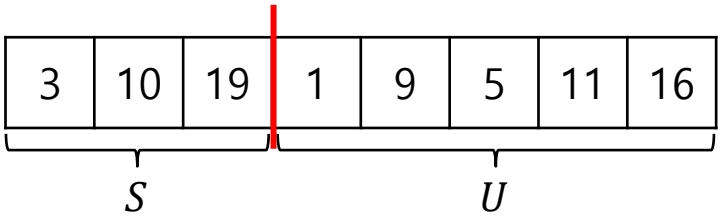
Representative Basic Sorting Algorithms

- Insertion sort
  - Example)



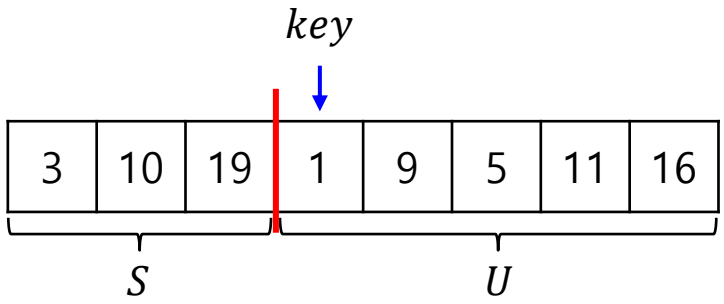


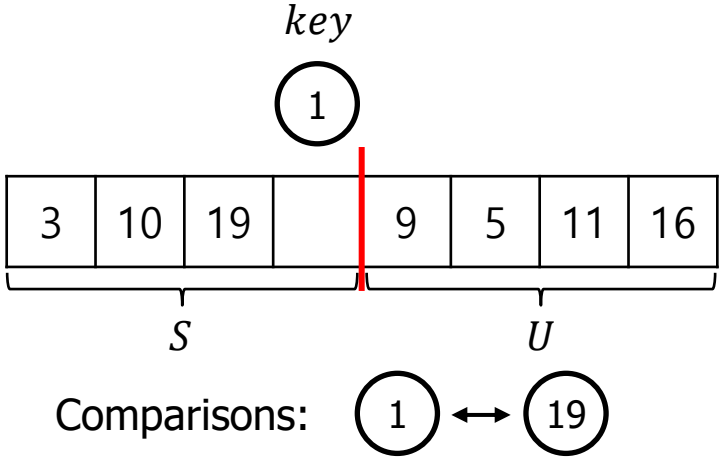


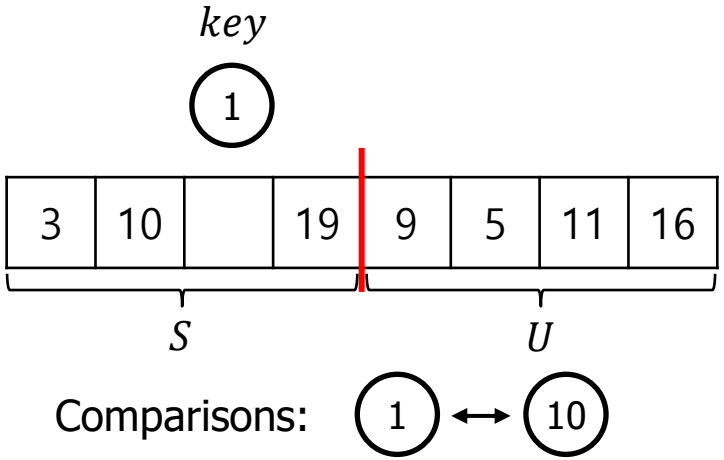


Representative Basic Sorting Algorithms

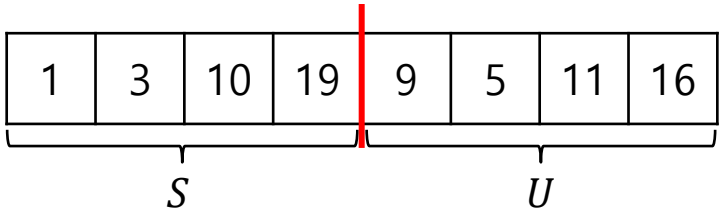
- Insertion sort
  - Example)





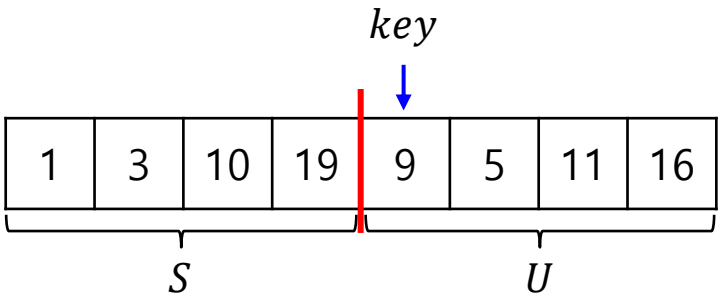


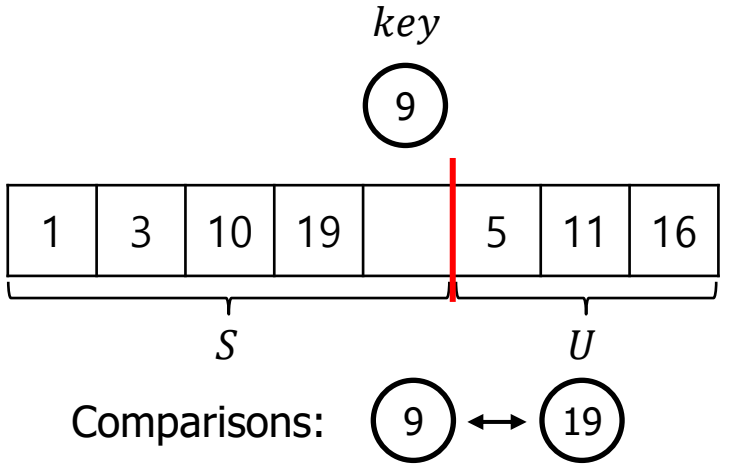
…

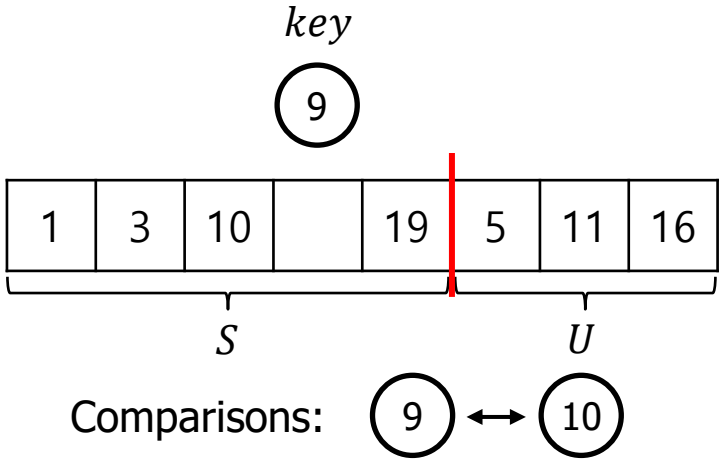


Representative Basic Sorting Algorithms

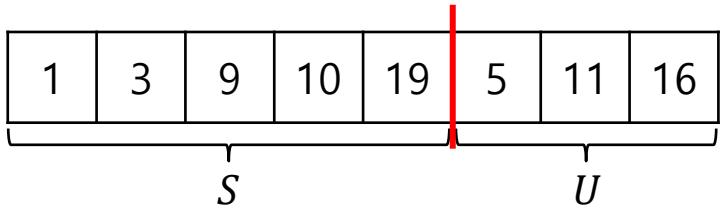
- Insertion sort
  - Example)





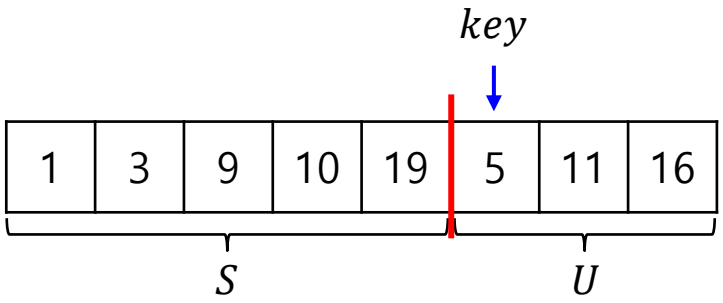


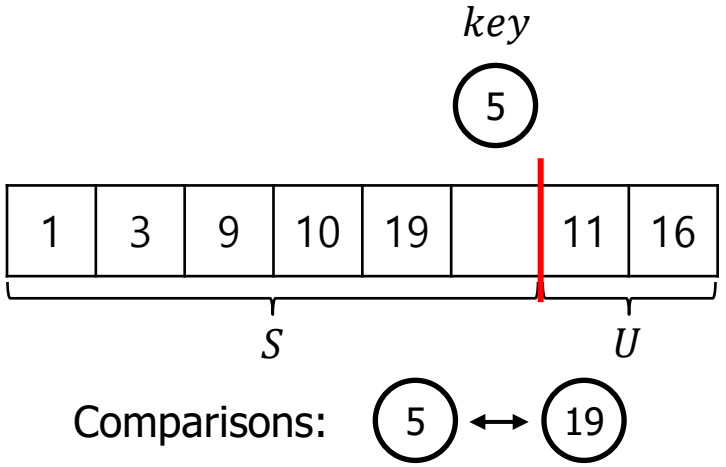
…

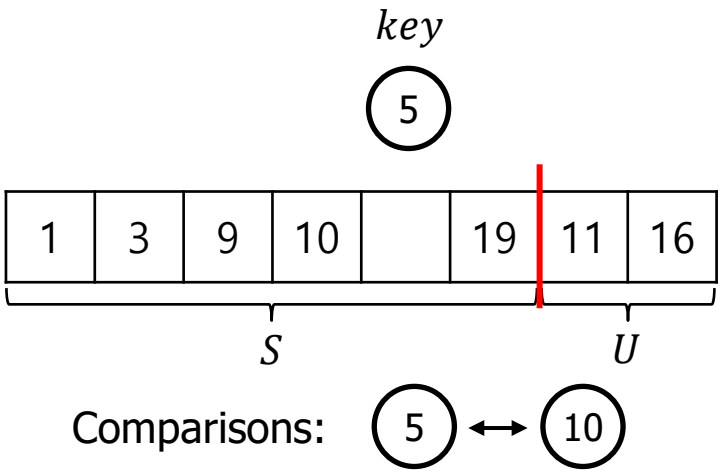


Representative Basic Sorting Algorithms

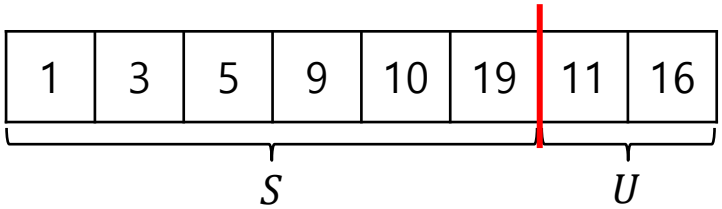
- Insertion sort
  - Example)





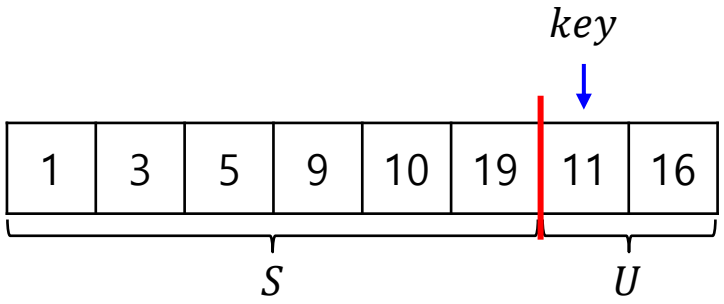


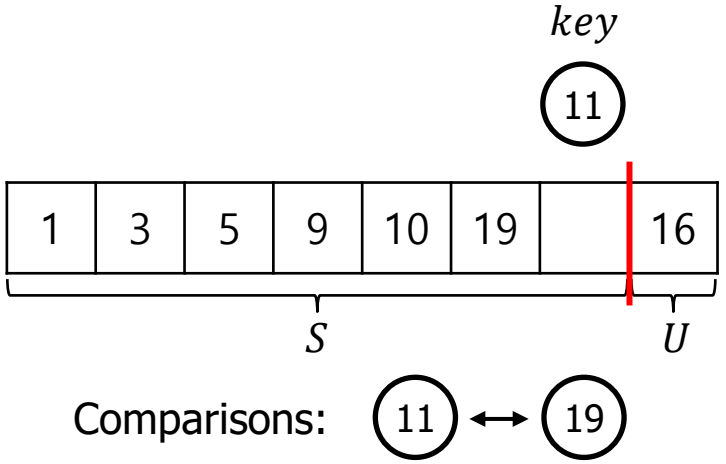
…

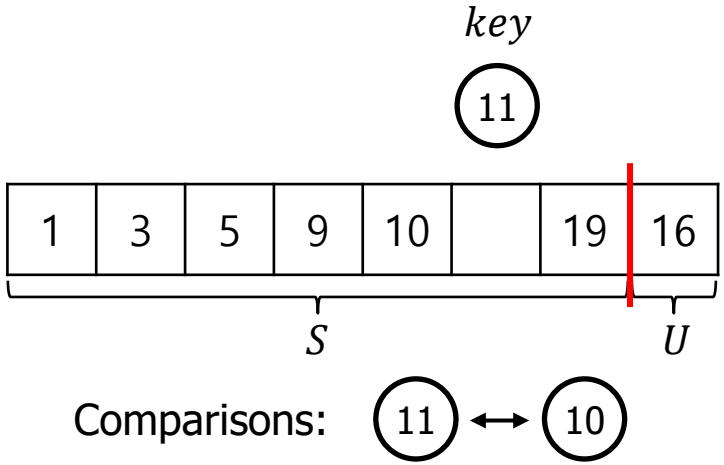


Representative Basic Sorting Algorithms

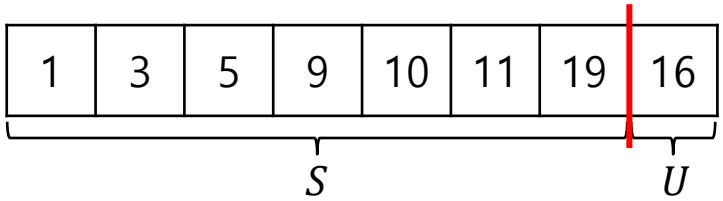
- Insertion sort
  - Example)





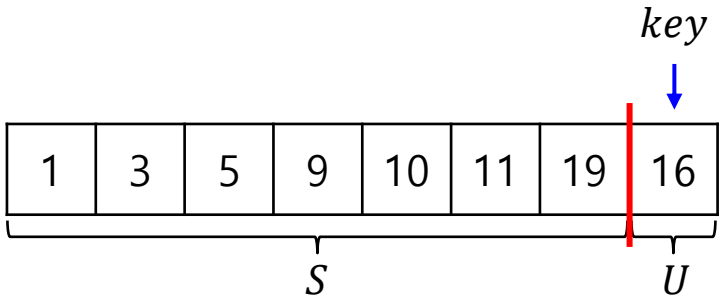


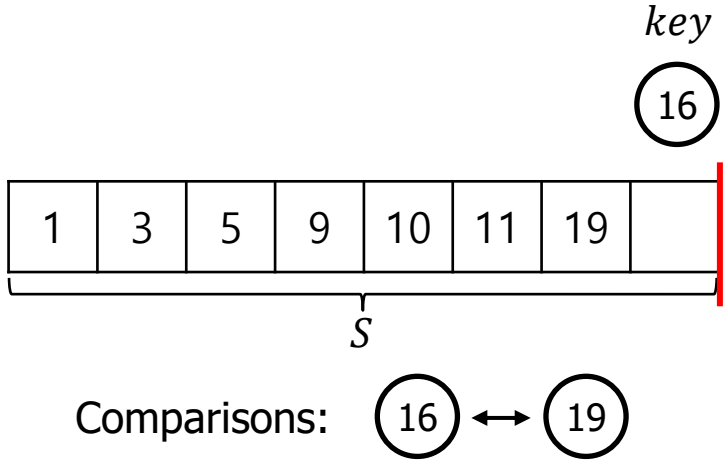
…

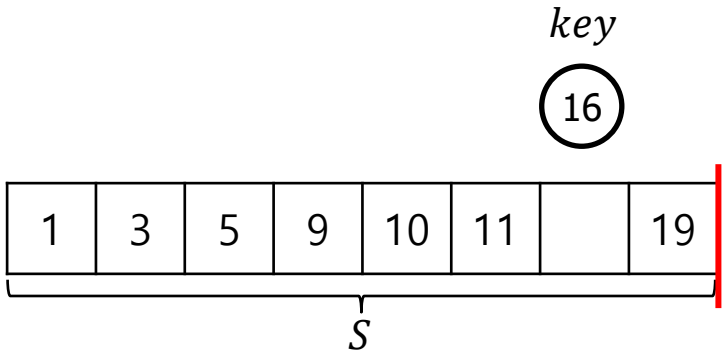


Representative Basic Sorting Algorithms

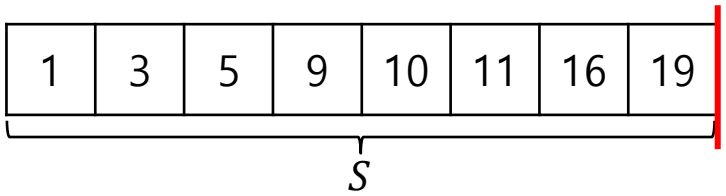
- Insertion sort
  - Example)







…



Representative Basic Sorting Algorithms

- Insertion sort
  - Psuedo code)

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

void insertion_sort(int arr[], int n){
   int i, j, key;
   for (i = 1; i < n; i++){
      key = arr[i];
      j = i – 1;
      while(j >= 0 && arr[j] > key){
         arr[j + 1] = arr[j];
         j = j – 1;
      }
      arr[j + 1] = key;
   }
}
```

Summary

- 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.

Questions?

SEE YOU NEXT TIME!