Algorithms & Data Structures

Sorting Algorithms

Lecture-4

Outline

- Several sorting algorithms:
  - Bubble Sort
  - Selection Sort
  - Insertion Sort
  - Shell Sort
- For each algorithm:
  - Basic Idea
  - Example
  - Implementation
  - Algorithm Analysis

Sorting

Introduction

- What is Sorting?

Arranging things into ascending or descending order is called sorting.

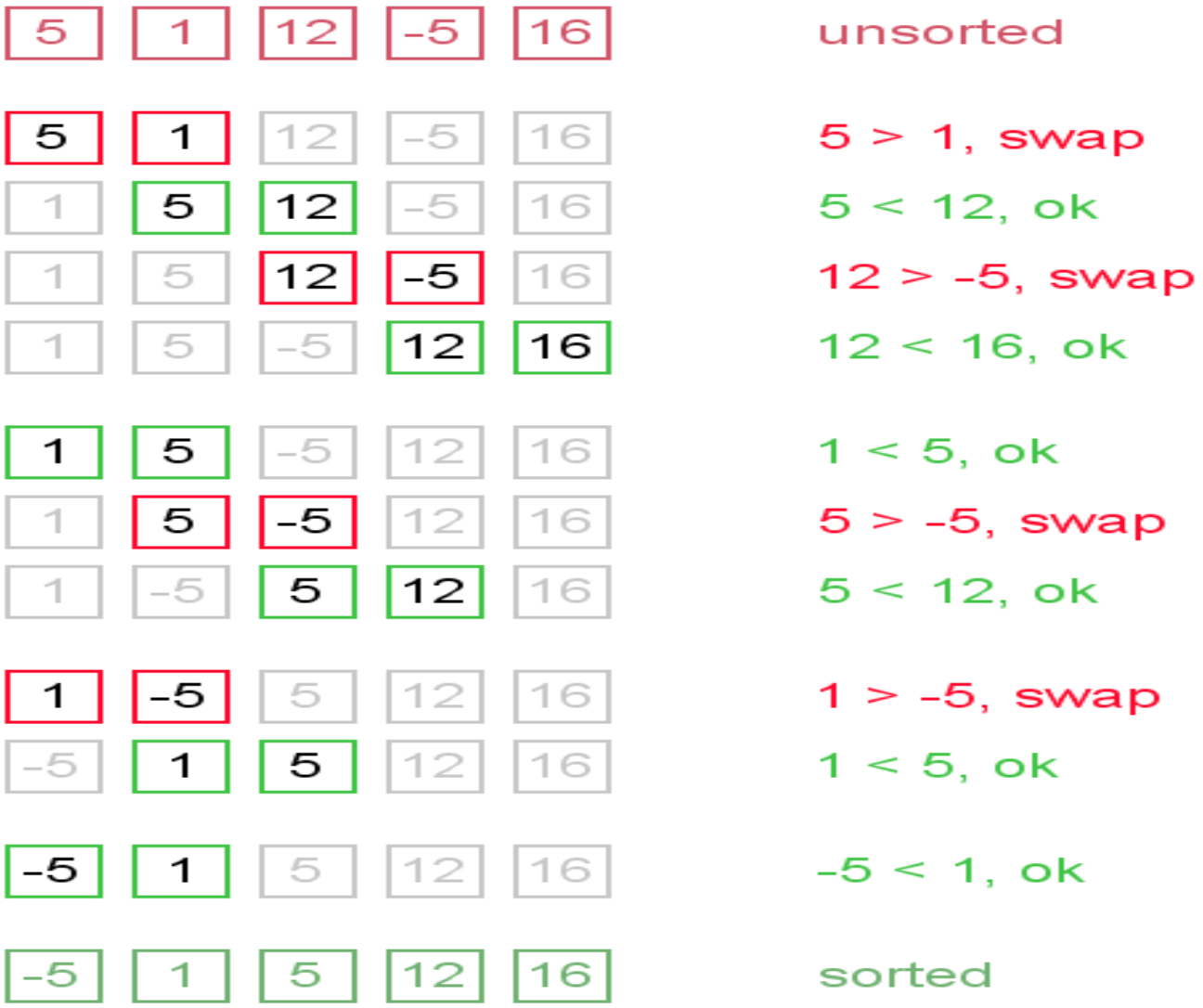
- Why do we need to sort data ?:
  - -- to arrange names in alphabetical order
  - -- arrange students by grade, etc.
  - -- preliminary step to searching data.
- Basic steps involved:
  - -- compare two items
  - -- swap the two items or copy one item.

Bubble Sort

- Compare two items
- If one on the left is greater, swap them else move right.
- Move one position right.
- Continue until the end.
- The rightmost item is the greatest.
- Go back and start another pass from the left end, going towards right, comparing and swapping whenever appropriate.
- Stop one item short of the end of the line.
- Continue until all items are sorted.

Bubble Sort

Bubble Sort Example 1



Bubble Sort

- Simplest sorting algorithm
- Idea:
  - 1. Set flag = false
  - 2. Traverse the array and compare pairs of two consecutive elements
    - 1.1 If E1 E2 -> OK (do nothing)
    - 1.2 If E1 > E2 then Swap(E1, E2) and set flag = true
  - 3. repeat 1. and 2. while flag=true.

Bubble Sort: Algorithm

```
public static void bubbleSort(int[] a)
{
    int temp;
    for(int i=0;i<arr.length-1;i++)
        for(int j=i+1;j<arr.length;j++)
            if(arr[i] > arr[j])
              {
                temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
    }
```

Bubble Sort: Analysis

- Efficiency:
  -- if N is number of items,
  -- N-1 comparisons in first pass, N-2 in second pass, so on and so forth.
  -- The formula is : (N-1) + (N-2) +…+ 1 = N* (N-1)/2.
  -- hence runs in $O(N^2)$ time.

Bubble Sort Example 2

8	6	34	2	51	32	21	original
6	**8**	34	2	51	32	21	pass 1
6	**8**	**34**	2	51	32	21	
6	8	**2**	**34**	51	32	21	
6	8	2	**34**	**51**	32	21	
6	8	2	34	**32**	**51**	21	
6	8	2	34	32	**21**	**51**	

Repeat the process until the list is sorted

Bubble Sort

6	8	2	34	32	21	51	
6	**8**	2	34	32	21	51	pass 2
6	**2**	**8**	34	32	21	51	
6	2	**8**	**34**	32	21	51	
6	2	8	**32**	**34**	21	51	
6	2	8	32	**21**	**34**	51	

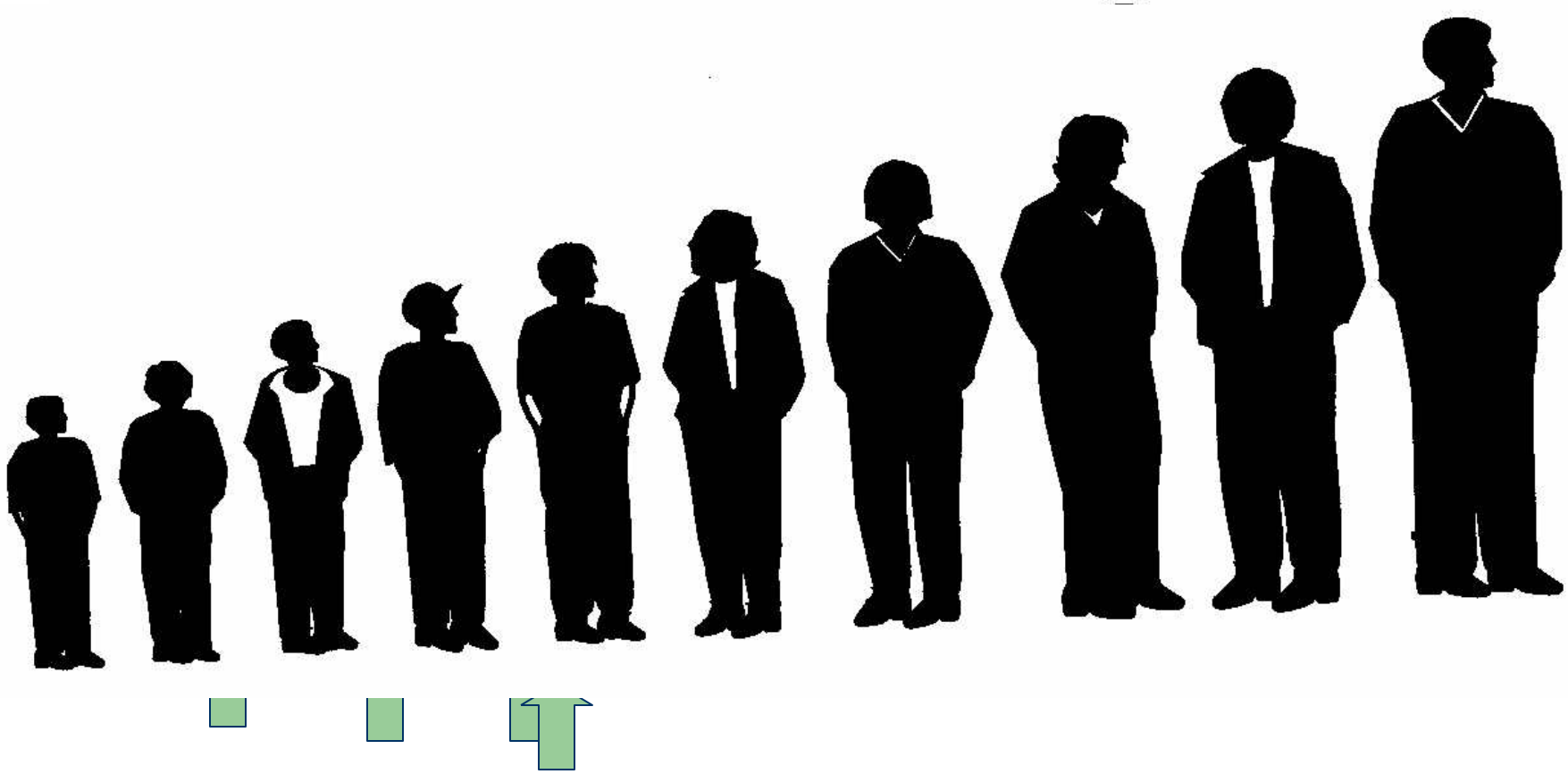
Repeat the process until the list is sorted

Selection Sort

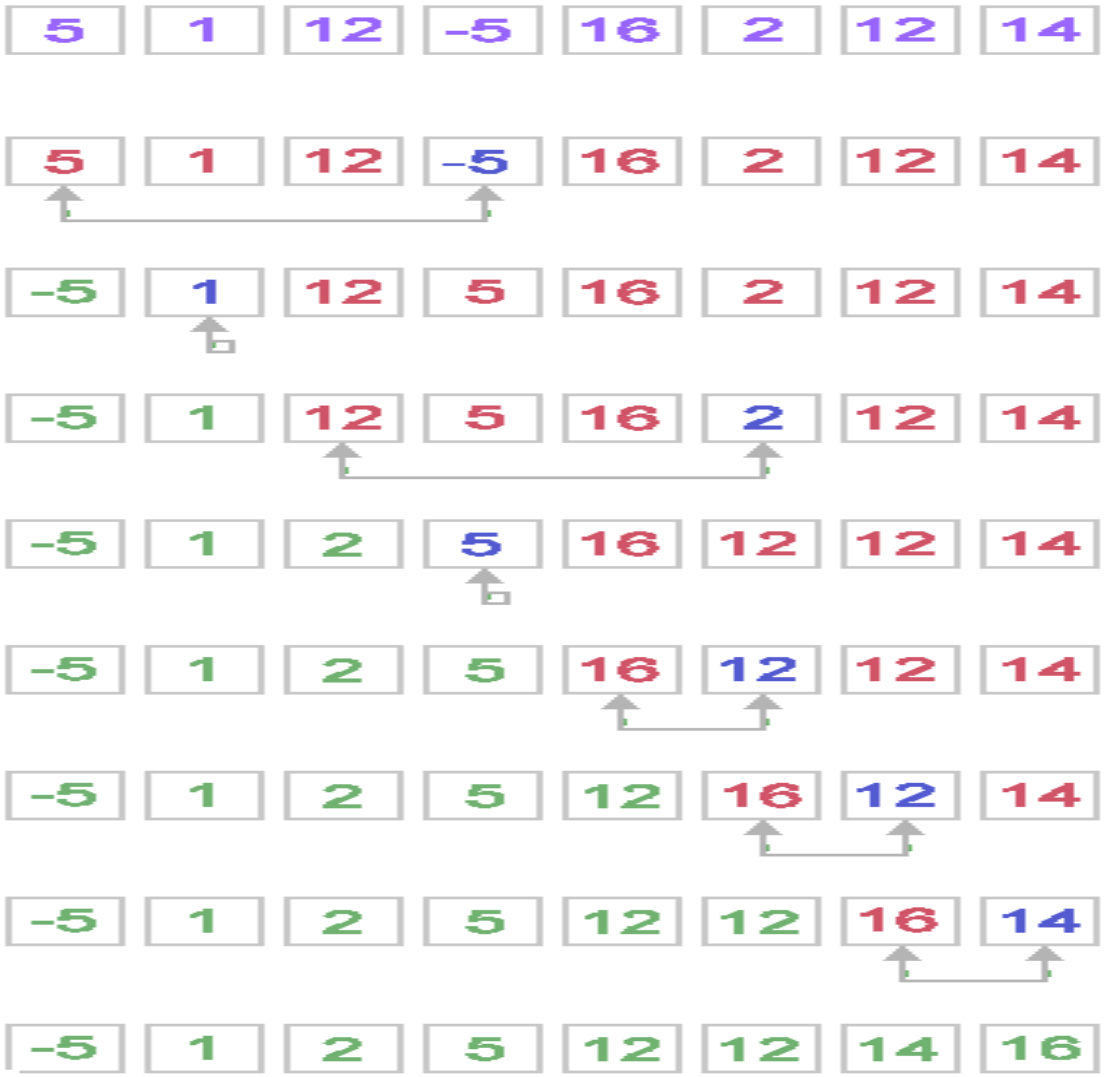
- In terms of an array A, the selection sort finds the smallest element in the array and exchanges it with A[0]. Then, ignoring A[0], the sort finds the next smallest and swaps it with A[1] and so on.
- Start at the left end.
- Mark the leftmost item, say as min.
- Compare this item with the next item, the shortest among two now becomes min.
- Keep comparing till the end, the final min item is swapped at placed at position 0.
- Then start with position 1 and repeat the process.

Selection Sort

min = out



Selection Sort – Example 1



Selection Sort : Algorithm

```
public void selectionSort(int[] arr) {
    int i, j, minIndex, tmp;
    int n = arr.length;
    for (i = 0; i < n - 1; i++) {
        minIndex = i;
        for (j = i + 1; j < n; j++)
            if (arr[j] < arr[minIndex])
                minIndex = j;
        if (minIndex != i) {
            tmp = arr[i];
            arr[i] = arr[minIndex];
            arr[minIndex] = tmp;
        }
    }
}
```

Selection Sort – Example 2

8	6	34	2	51	32	21	original
2	6	34	8	51	32	21	pass 1
2	**6**	34	8	51	32	21	pass 2
2	**6**	**8**	34	51	32	21	pass 3
2	**6**	**8**	**21**	51	32	34	pass 4
2	**6**	**8**	**21**	**32**	51	34	pass 5
2	**6**	**8**	**21**	**32**	**34**	51	pass 6

Selection Sort: Number of Comparisons

Elements in unsorted	Comparisons to find min
n	n-1
n-1	n-2
…	…
3	2
2	1
1	0
	n(n-1)/2

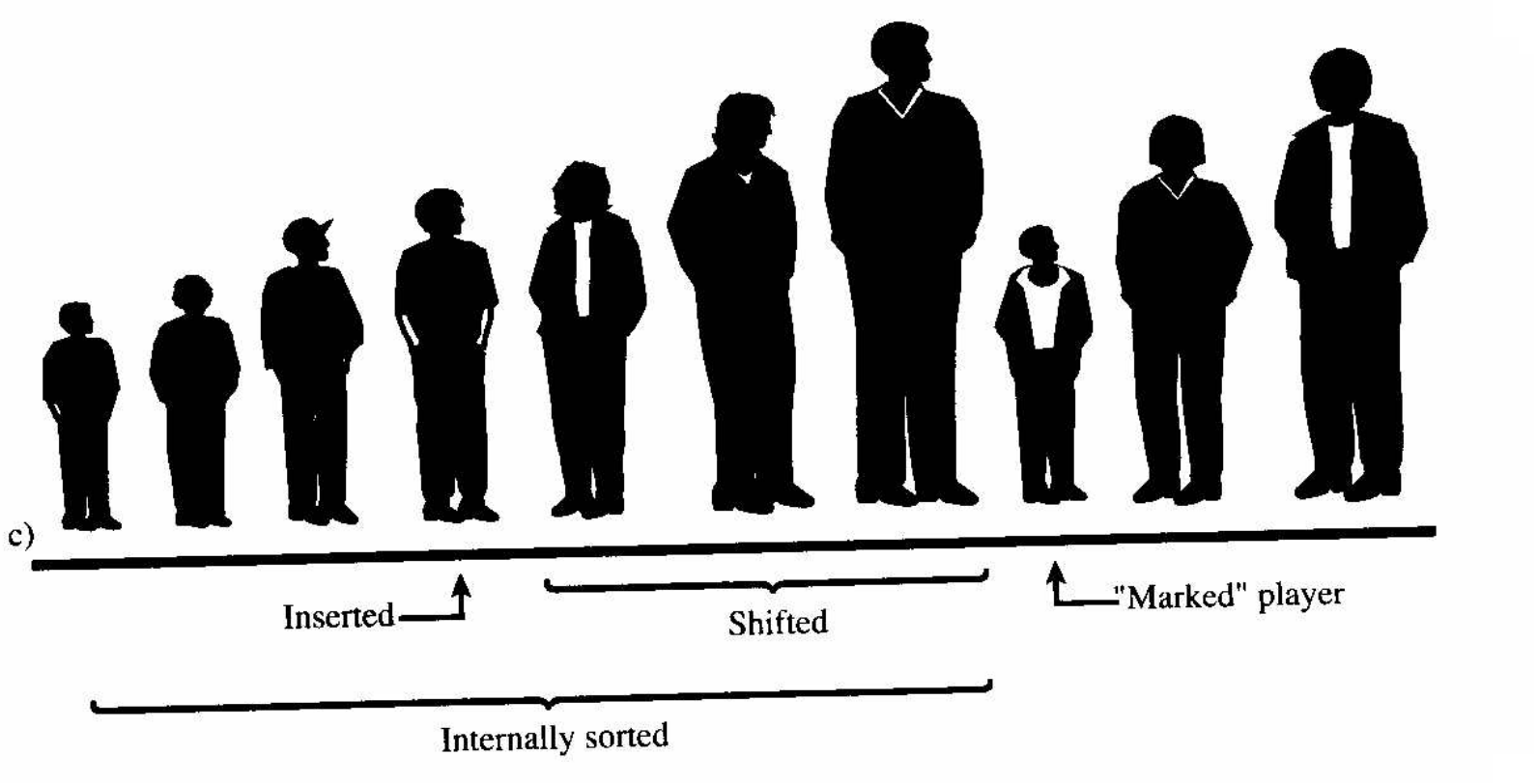
Selection Sort - Complexity Analysis

- Efficiency:
  - -- Number of comparisons is same, N*(N-1)/2.
  - -- runs in $O(N^2)$ time.
- How do you compare it to Bubble sort?
  - **Faster than bubble sort because of fewer swaps.**

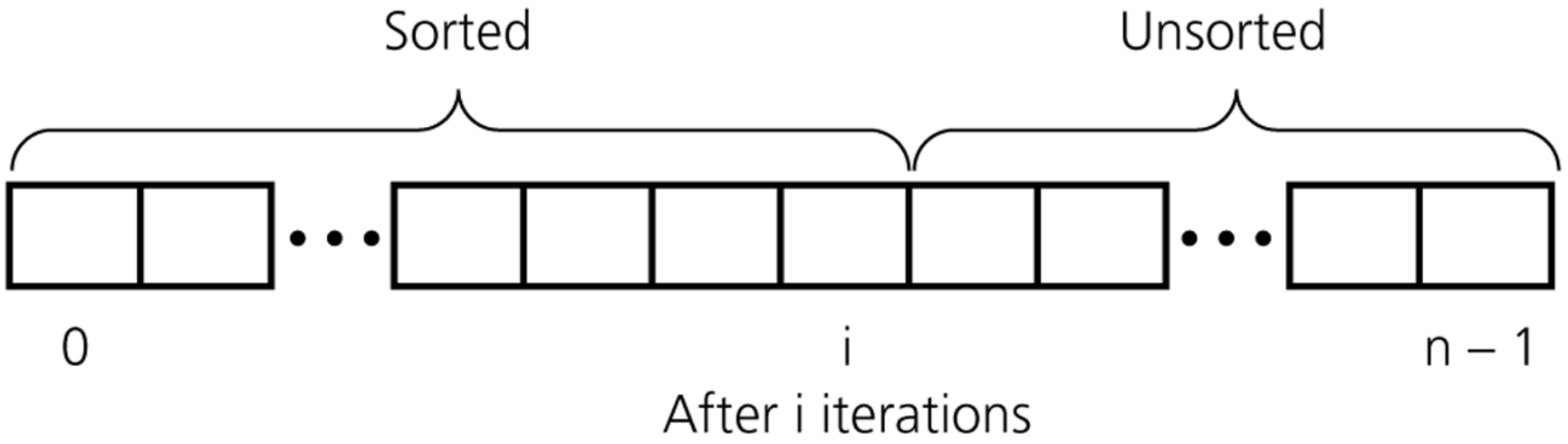
Insertion Sort

1. We have two group of items:
   - sorted group, and
   - unsorted group
2. Initially, all items in the unsorted group and the sorted group is empty.
   - We assume that items in the unsorted group unsorted.
   - We have to keep items in the sorted group sorted.
3. Pick any item from, then insert the item at the right position in the sorted group to maintain sorted property.
4. Repeat the process until the unsorted group becomes empty.

Insertion Sort



Marked player

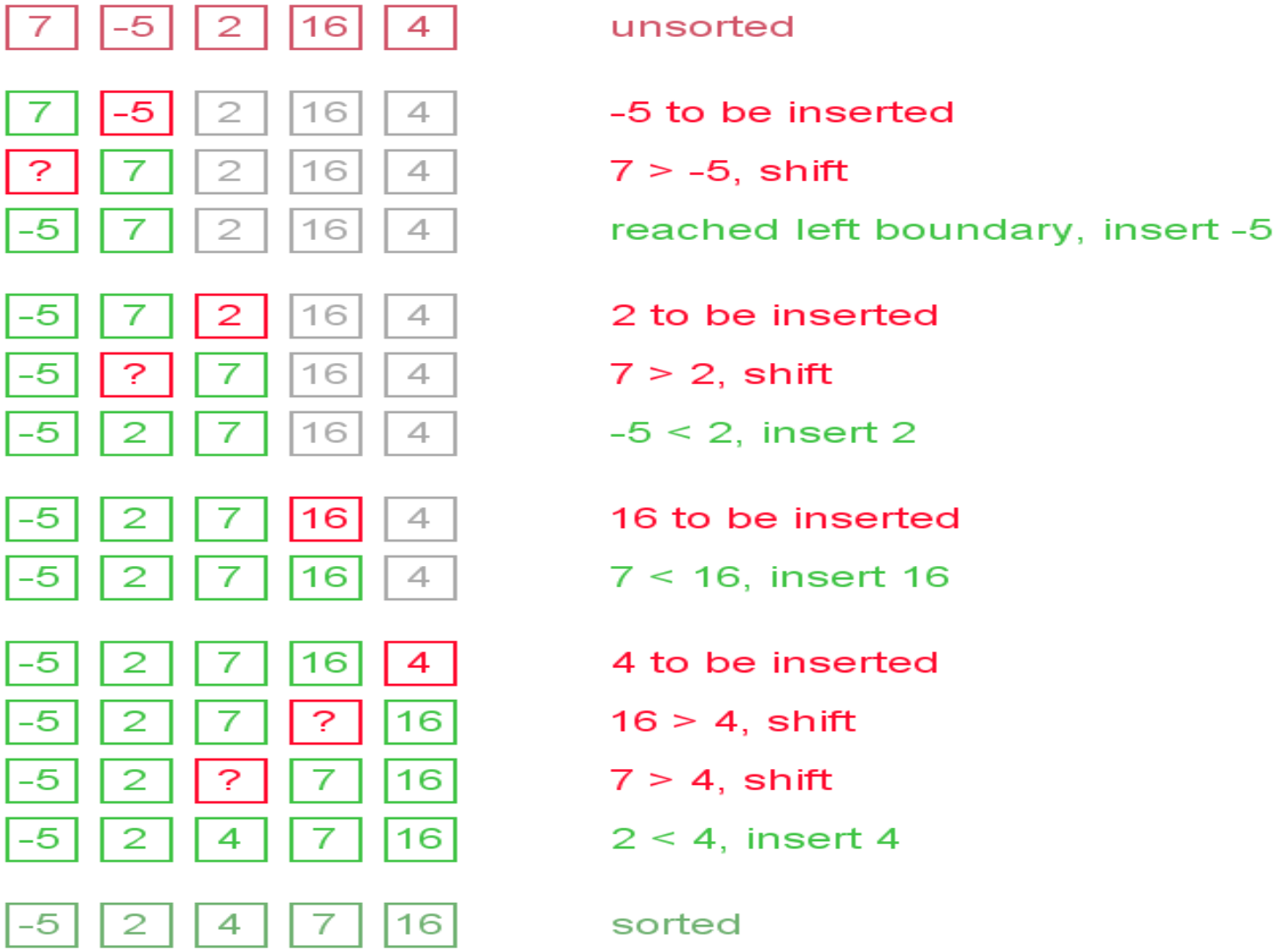


An insertion sort partitions the array into two regions

Insertion Sort: Idea

- Idea: sorting cards.
  - 8 | 5 9 2 6 3
  - 5 8 | 9 2 6 3
  - 5 8 9 | 2 6 3
  - 2 5 8 9 | 6 3
  - 2 5 6 8 9 | 3
  - 2 3 5 6 8 9 |

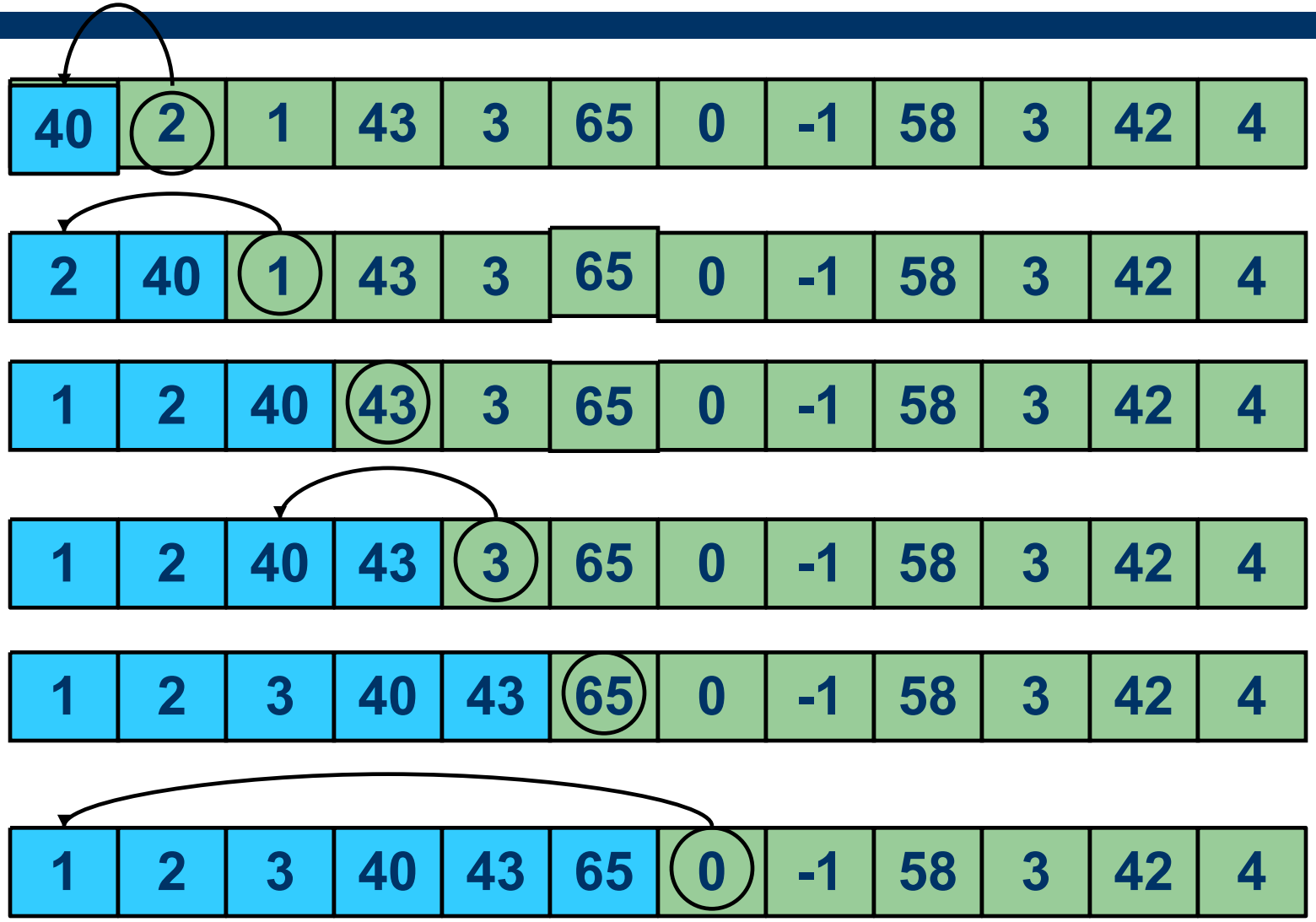
Insertion Sort: Example 1



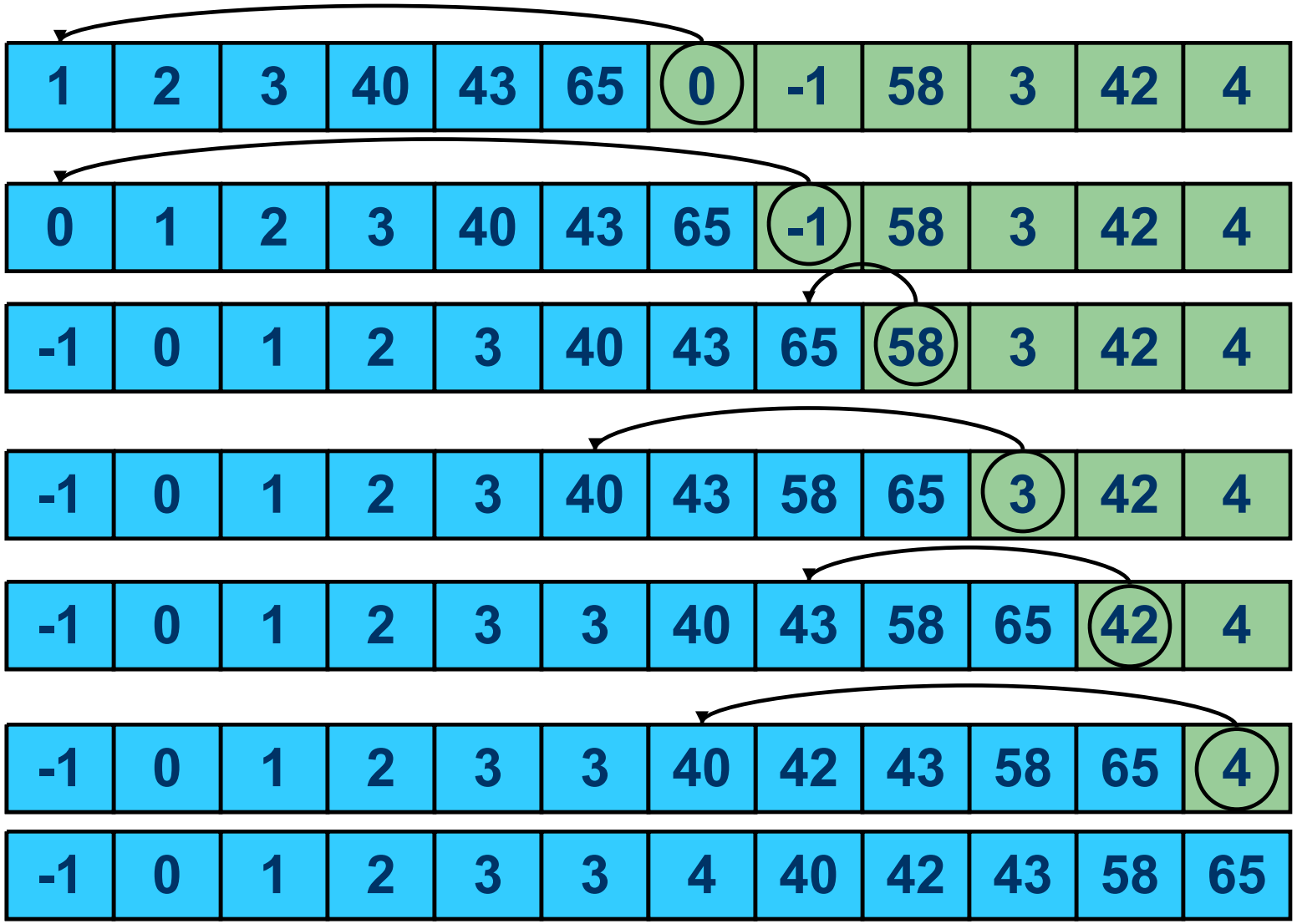
Insertion Sort - Algoritm

```
void insertionSort(int[] arr) {
    int i, j, newValue;
    for (i = 1; i < arr.length; i++) {
        newValue = arr[i];
        j = i;
        while (j > 0 && arr[j - 1] > newValue) {
            arr[j] = arr[j - 1];
            j--;
        }
        arr[j] = newValue;
    }
}
```

Insertion Sort: Example 2



Insertion Sort: Example 2



Insertion Sort: Complexity Analysis

- On the first pass, compares at the most one item, two on second pass and so on. N-1 comparisons in the last pass.
- On each pass an average of only half of the maximum number of items are actually compared before the insertion point is found.
- N*(N-1)/4 comparisons.
- Since there is no swapping, it is twice as faster than bubble sort and faster than selection Sort.
- Runs in $O(N^2)$ time.

Comparison: Complexity Analysis

- Bubble sort is useful for small amounts of data.
- Selection sort can be used when amount of data is small and swapping is time-consuming.
- Insertion sort is the best when the list is almost sorted.

Comparison of Quadratic Sorts

Quadratic Sorts	Comparisons		Exchanges	
	Best	Worst	Best	Worst
Selection Sort	$O(n^2)$	$O(n^2)$	$O(1)$	$O(n)$
Bubble Sort	$O(n)$	$O(n^2)$	$O(1)$	$O(n^2)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(1)$	$O(n^2)$