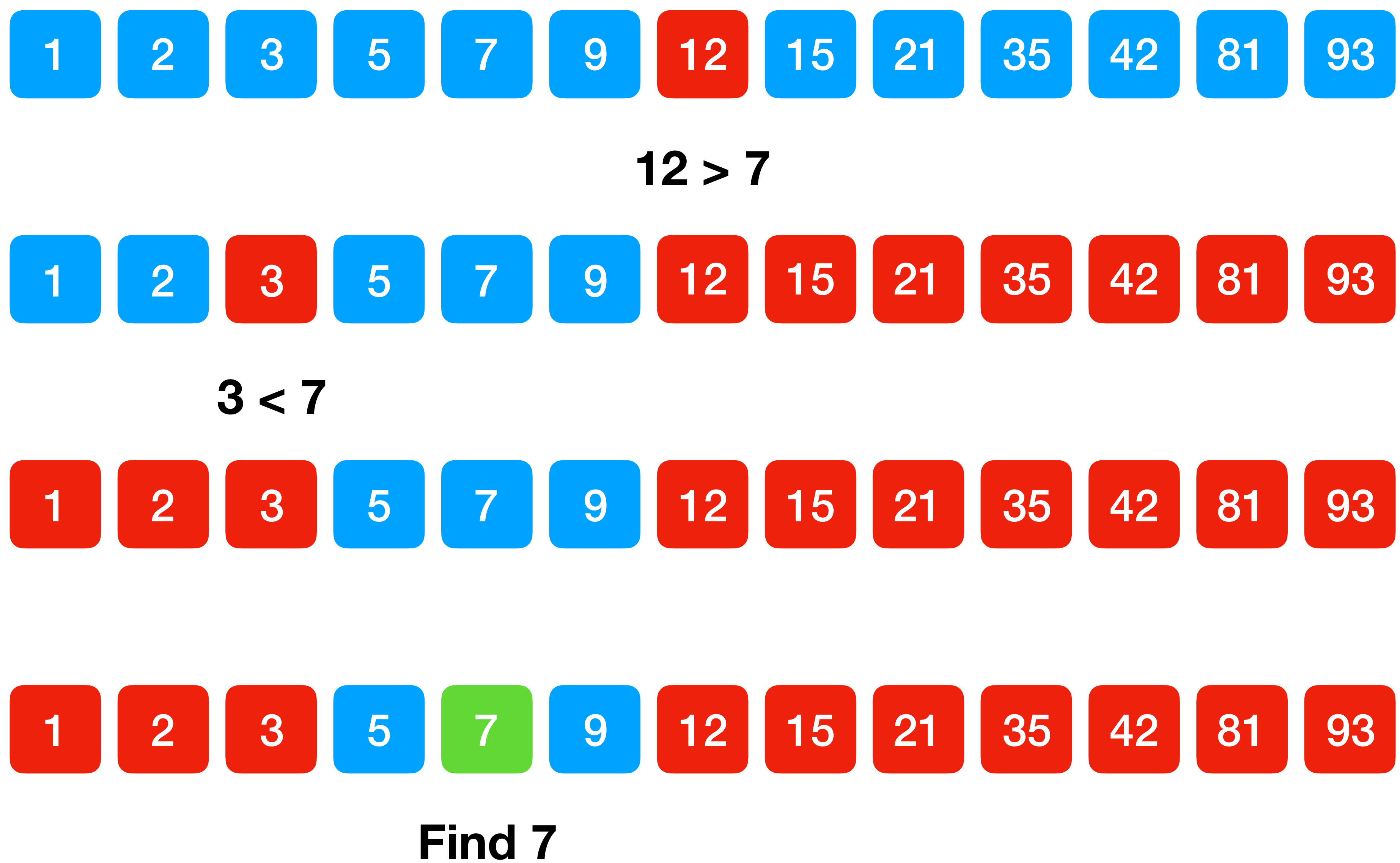
# Java Sorting and Search

2019 Lecture 5

# Review of search

```java
public static int indexOf(int[] a, int key)
{
    int lo = 0;
    int hi = a.length - 1;
    while (lo <= hi)
    {
        // Key is in a[lo..hi] or not present.
        int mid = lo + (hi - lo) / 2; // do this to avoid overflow
        if (key < a[mid])
        {
            hi = mid - 1;
        }
        else if (key > a[mid])
        {

            lo = mid + 1;
        }
        else
        {
            return mid;
        }
    }
    return -1;
}
```

# Search for 7


1 2 3 5 7 9 12 15 21 35 42 81 93
12 > 7
1 2 3 5 7 9 12 15 21 35 42 81 93
3 < 7
1 2 3 5 7 9 12 15 21 35 42 81 93
1 2 3 5 7 9 12 15 21 35 42 81 93
Find 7

# Effective Search

- With chaos order, search has to be done linearly
- With order search, we can apply algorithm like binary search to find the target efficiently
- The key pre condition of binary search is that the list has to be sorted
- Sort algorithm therefore is very important

# Sorting

- The process to make a random ordered list to a ordered list
- Speed is the most important aspect of sorting
- Some sorting method require extra memory allocation
- For sorting algorithm that require no extra space we call it in place sorting

# Need to know for the test

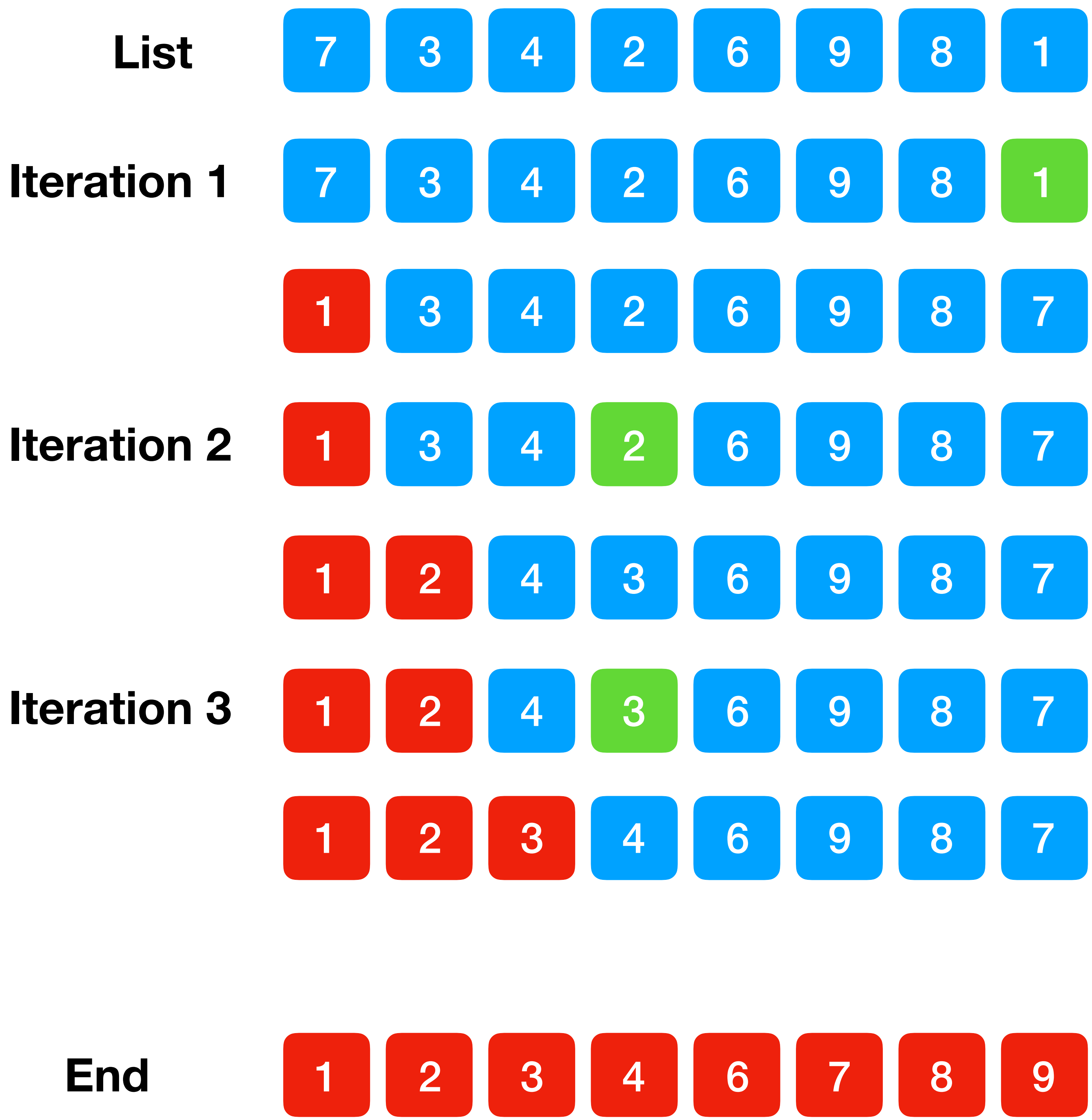
- Selection Sort
- Insertion Sort
- Merge Sort

# Selection sort

- Slowest but most easy to understand
- For each iteration, find the next smallest item in the remaining list and put to the front.

# Before selection sort

- Write program to find the minimum value in a list and return its index.

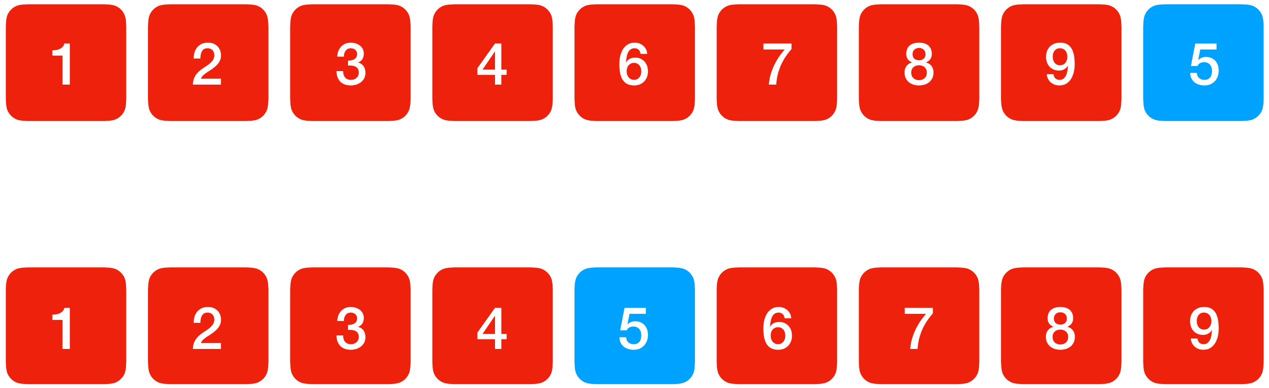

List
7 3 4 2 6 9 8 1
Iteration 1
7 3 4 2 6 9 8 1
1 3 4 2 6 9 8 7
Iteration 2
1 3 4 2 6 9 8 7
1 2 4 3 6 9 8 7
Iteration 3
1 2 4 3 6 9 8 7
1 2 3 4 6 9 8 7
End
1 2 3 4 6 7 8 9

```java
public static void selectionSort(int[] a) {

    int length = a.length;

    for (int i = 0; i < length - 1; ++i) {

        int minIndex = i;
        for (int j = i + 1; j < length; ++j) {
            if (a[j] < a[minIndex]) {
                minIndex = j;
            }
        }

        // swap to the top
        int tmp = a[i];
        a[i] = a[minIndex];
        a[minIndex] = tmp;
    }
}
```

# Before insertion sort

- Put a new item at the end of a sorted list.
- Move the new item to the correct position

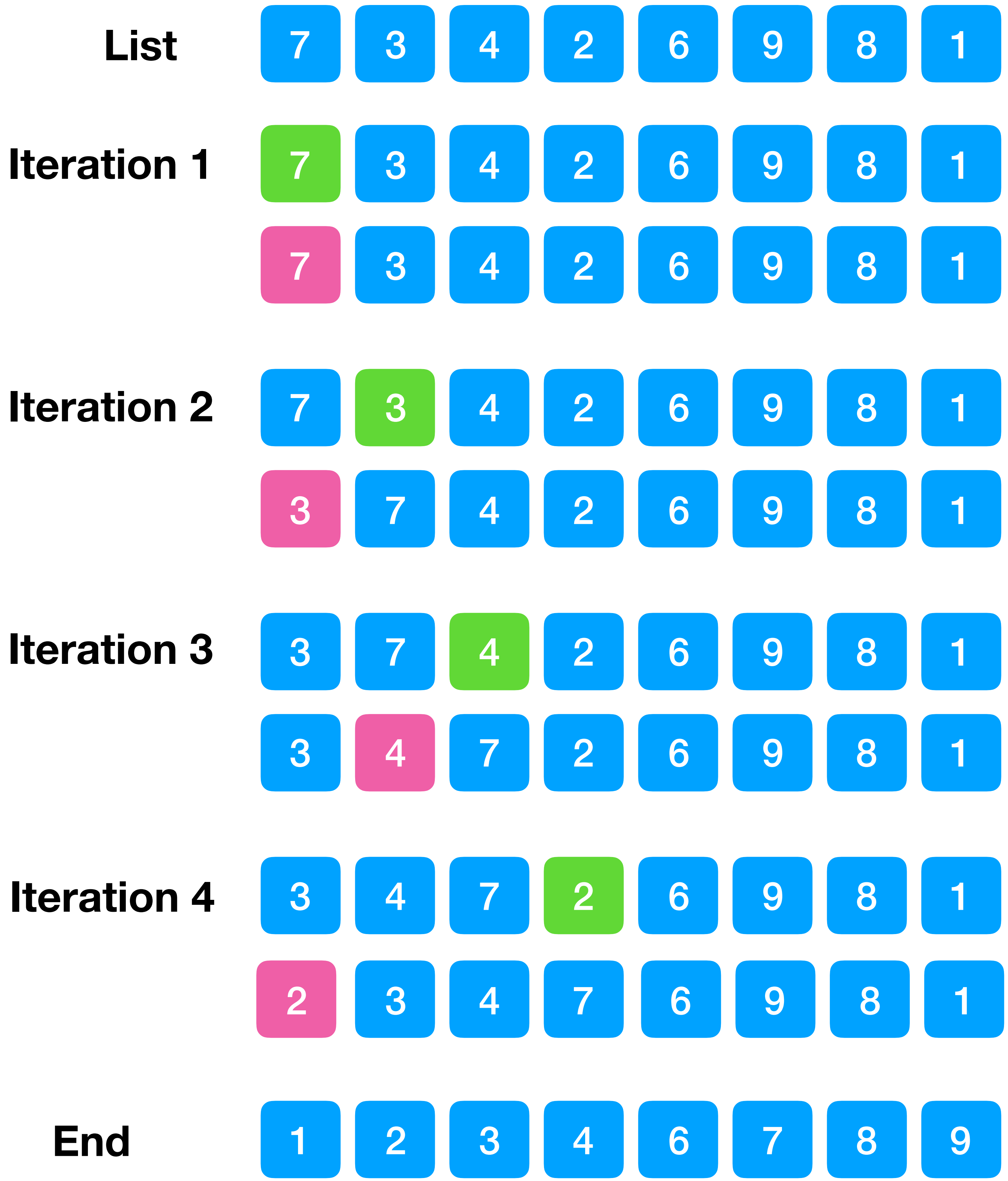
List



```java
private static int[] testTarget = new int[10];
private static int size = 0;
private static int next = 0;

public static boolean insert(int value) {
    if (size == testTarget.length) {
        return false;
    }

    testTarget[next] = value;
    for (int i = next; i > 0; i--) {
        if (testTarget[i] < testTarget[i - 1]) {
            int temp = testTarget[i - 1];
            testTarget[i - 1] = testTarget[i];
            testTarget[i] = temp;
        } else {
            break; // since all previous list are sorted
        }
    }

    next++;
    size++;
    return true;
}
```

# Insertion sort

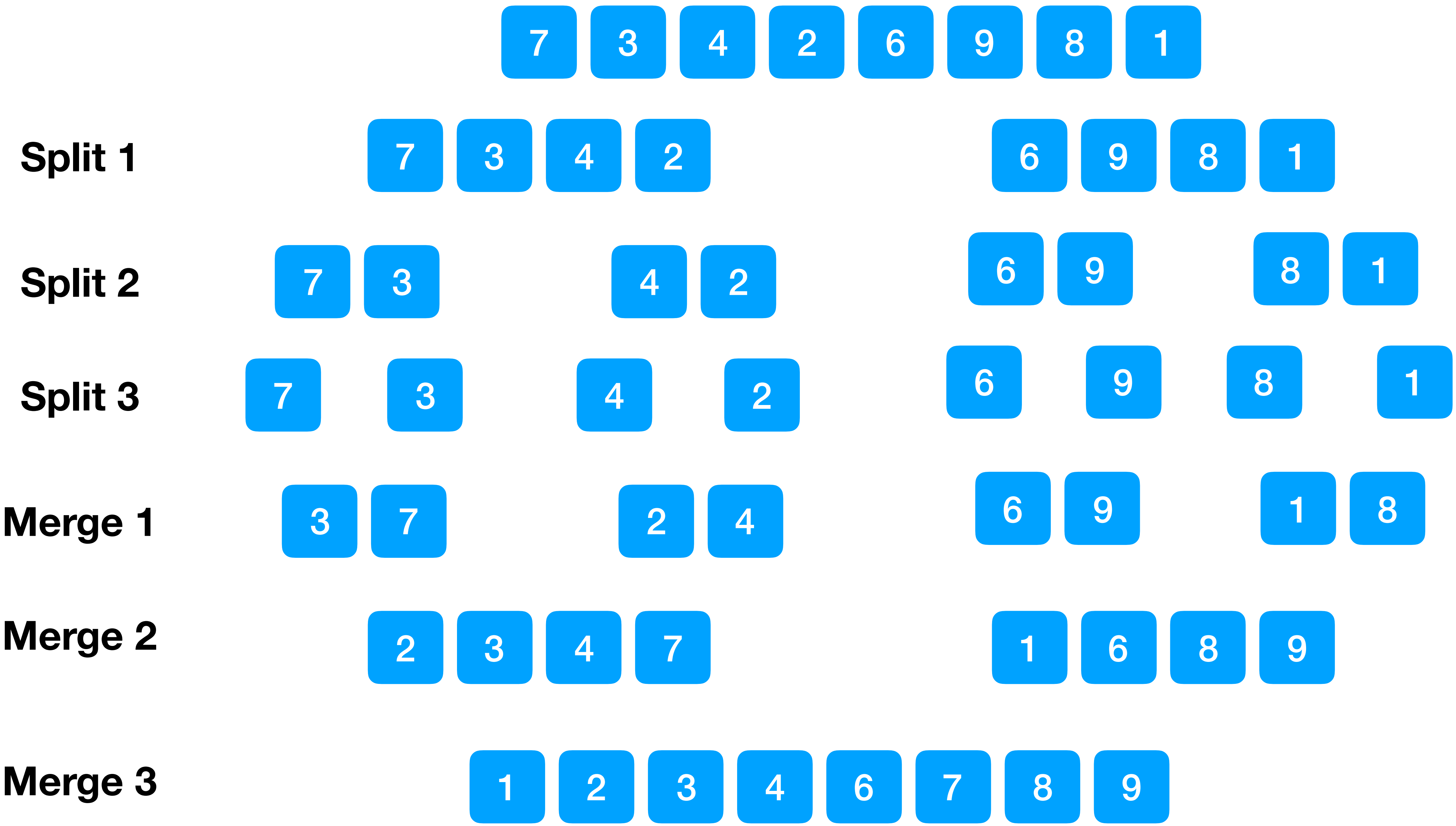
- Most common used in place sort for short list
- For each iteration, treat the next item as new insertion value, and move to it’s right location


List 7 3 4 2 6 9 8 1
Iteration 1 7 3 4 2 6 9 8 1
7 3 4 2 6 9 8 1
Iteration 2 7 3 4 2 6 9 8 1
3 7 4 2 6 9 8 1
Iteration 3 3 7 4 2 6 9 8 1
3 4 7 2 6 9 8 1
Iteration 4 3 4 7 2 6 9 8 1
2 3 4 7 6 9 8 1
End 1 2 3 4 6 7 8 9

```java
public static void insertionSort(int[] a) {
    int length = a.length;

    for (int i = 1; i < length; ++i) {
        int insertValue = a[i];

        for (int j = i; j > 0;j--) {
            if (a[j] < a[j - 1]) {
                int temp = a[i - 1];
                a[i - 1] = a[i];
                a[i] = temp;
            } else {
                break; // since all previous list are sorted
            }
        }
    }
}
```

# Merge sort

- Fastest sorting method that you need to know for this class
- Require additional memory to finish the sort
- Using recursion method
- Divide and concur solution.
    - split the list into smaller list
    - And merge them one by one


7 3 4 2 6 9 8 1
Split 1
7 3 4 2
6 9 8 1
Split 2
7 3
4 2
6 9
8 1
Split 3
7
3
4
2
6
9
8
1
Merge 1
3 7
2 4
6 9
1 8
Merge 2
2 3 4 7
1 6 8 9
Merge 3
1 2 3 4 6 7 8 9

# Merge two sorted list

```java
public int[] merge(int[] arr1, int[] arr2) {
    int[] result = new int[arr1.length + arr2.length];

    int index1 = 0;
    int index2 = 0;

    int indexResult = 0;

    while (index1 < arr1.length  && index2 < arr2.length) {
        if (arr1[index1] <= arr2[index2]) {
            result[indexResult] = arr1[index1];
            index1++;
        }
        else {
            result[indexResult] =  arr2[index2];
            index2++;
        }
        indexResult++;
    }

    /* Copy remaining elements of L[] if any */
    while (index1 < arr1.length)
    {
        result[indexResult] = arr1[index1];
        index1++;
        indexResult++;
    }

    /* Copy remaining elements of R[] if any */
    while (index2 < arr2.length)
    {
        result[indexResult] =  arr2[index2];
        index2++;
        indexResult++;
    }

    return result;
}
```

# Merge sort

```java
private void split(int arr[], int start, int end)
{
    if (start >= end) {
        return;
    } else {
        // Find the middle point
        int mid = (start + end) / 2;

        // Sort left
        split(arr, start, mid);
        // Sort right
        split(arr, mid + 1, end);

        // Merge the sorted halves
        merge(arr, start, mid, end);
    }
}
```

```java
private void merge(int arr[], int start, int mid, int end)
{
    int leftSize = mid - start + 1;
    int rightSize = end - mid;

    /* Create temp arrays */
    int L[] = new int[leftSize];
    int R[] = new int[rightSize];

    /* Copy data to temp arrays */
    for (int i = 0; i < leftSize; ++i)
        L[i] = arr[start + i];
    for (int j = 0; j < rightSize; ++j)
        R[j] = arr[mid + 1 + j];

    int i = 0, j = 0;

    int currentWalker = start;
    while (i < leftSize && j < rightSize)
    {
        if (L[i] <= R[j])
        {
            arr[currentWalker] = L[i];
            i++;
        }
        else
        {
            arr[currentWalker] = R[j];
            j++;
        }
        currentWalker++;
    }

    /* Copy remaining elements of L[] if any */
    while (i < leftSize)
    {
        arr[currentWalker] = L[i];
        i++;
        currentWalker++;
    }

    /* Copy remaining elements of R[] if any */
    while (j < rightSize)
    {
        arr[currentWalker] = R[j];
        j++;
        currentWalker++;
    }
}
```