ADT and Problem Solving
Lecture 4

Recursion & Adv. Sorting

Pree Thiengburanathum, PhD.

Agenda

- Iterative vs Recursive
- Advance Sorting

Iterative

- Iterative approach is a repetition process until the condition fails
- They are often simple referred as loops
- Loops are used such as ***for*** loop, ***while*** loop, ***do*** loop.
- By this time you should have master this programming approach.

```
public static void countdown(int n) {
    while (n > 0) {
      System.out.println(n);
      n = n-1;
    }
    System.out.println("Blastoff!");
  }
```

Recursion

- Recursion is the process of defining a problem (or the solution to a problem) in terms of (a simpler version of) itself.
- A function calls repeatedly

- **Advantages:**
  - Performs better in solving problems based on tree structures.
  - Reduce the time to write and debug code, makes code smaller

- **Disadvantages:**
  - Use more memory than iterative method.
  - Slower than iterative due to the overhead of maintaining the stack.

Recursion

- Recursion is basically divided into two cases:
- *Base case:*
  - It stops the function to call itself when specific condition matches. Whereas there can be more than one base cases.
- *General case:*
  - It call function from within that function.

Example of recursion: Factorial

$n! = n \times (n-1) \times (n-2) \times \ldots \times 2 \times 1$

```java
int calfactorial (int n){
int fac = 1;
for (int i = 2; i <= n; i++) {
    fac = fac * i;
}
return fac;
}
```

```java
public static long factorial(int n) {
    if (n == 1) return 1;
    return n * factorial(n-1);
}
```

Example of recursion: Euclid's algorithm

- The *greatest common divisor* (gcd) of two positive integers is the largest integer that divides evenly into both of them. For example, the *gcd*(102, 68) = 34.

- *gcd*(32, 5)?
- 32 = 5 x 6 + 2
- 5 = 2 x 2 + 1
- 2 = 1 x 2 + 0

- *gcd* (3, 7)
- 7 = 3x2+1
- 1 = 1x1+0

Euclid’s algorithm: iterative vs recurisve

```java
public static int gcd_1(int p, int q) {
    while (q != 0) {
        int temp = q;
        q = p % q;
        p = temp;
    }
    return p;
}
```

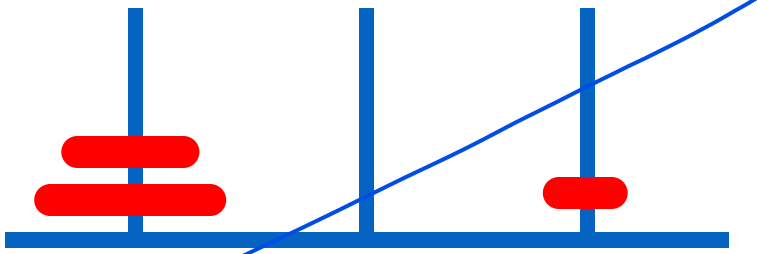
```java
public static int gcd(int p, int q) {
    if (q == 0) return p;
    else return gcd(q, p % q);
}
```

Towers of Hanoi

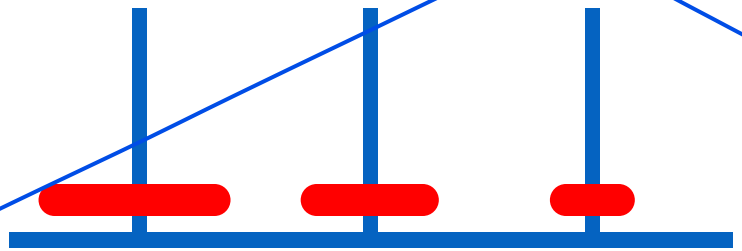
- The *Towers of Hanoi* is a puzzle made up of three vertical pegs and several disks that slide on the pegs
- The disks are of varying size, initially placed on one pole with the largest disk on the bottom with increasingly smaller ones on top
- The goal is to move all of the disks from one pole to another under the following rules:
  - We can move only one disk at a time
  - We cannot move a larger disk on top of a smaller one

Towers of Hanoi

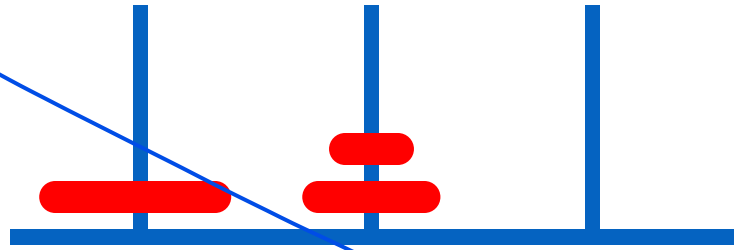
Original Configuration



Move 1

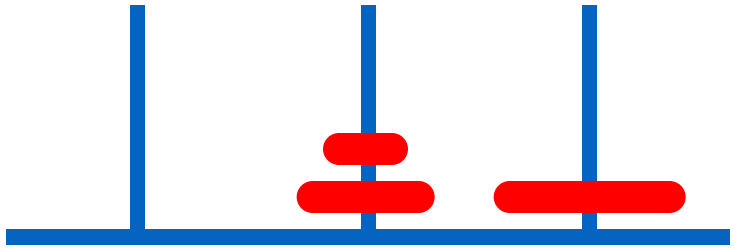


Move 2

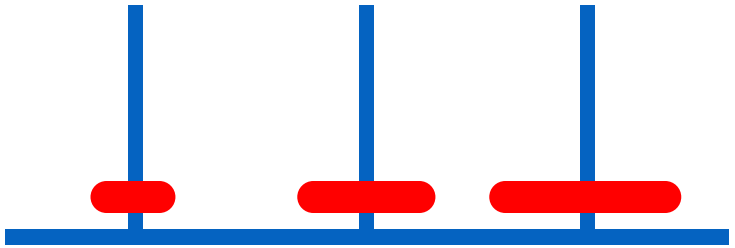


Move 3

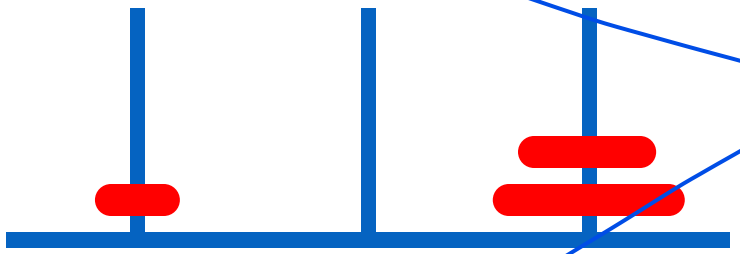
Towers of Hanoi



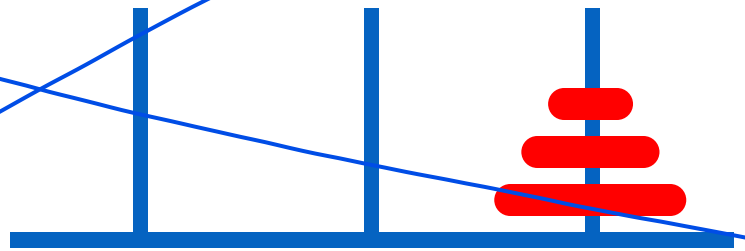
Move 4



Move 5



Move 6



Move 7 (done)

Class exercise : recursion

- Printing problem
  - Write a simple Java program that print number from 20 to 1 for both iterative and recursive version.

Output

20

19

18

..

..

1

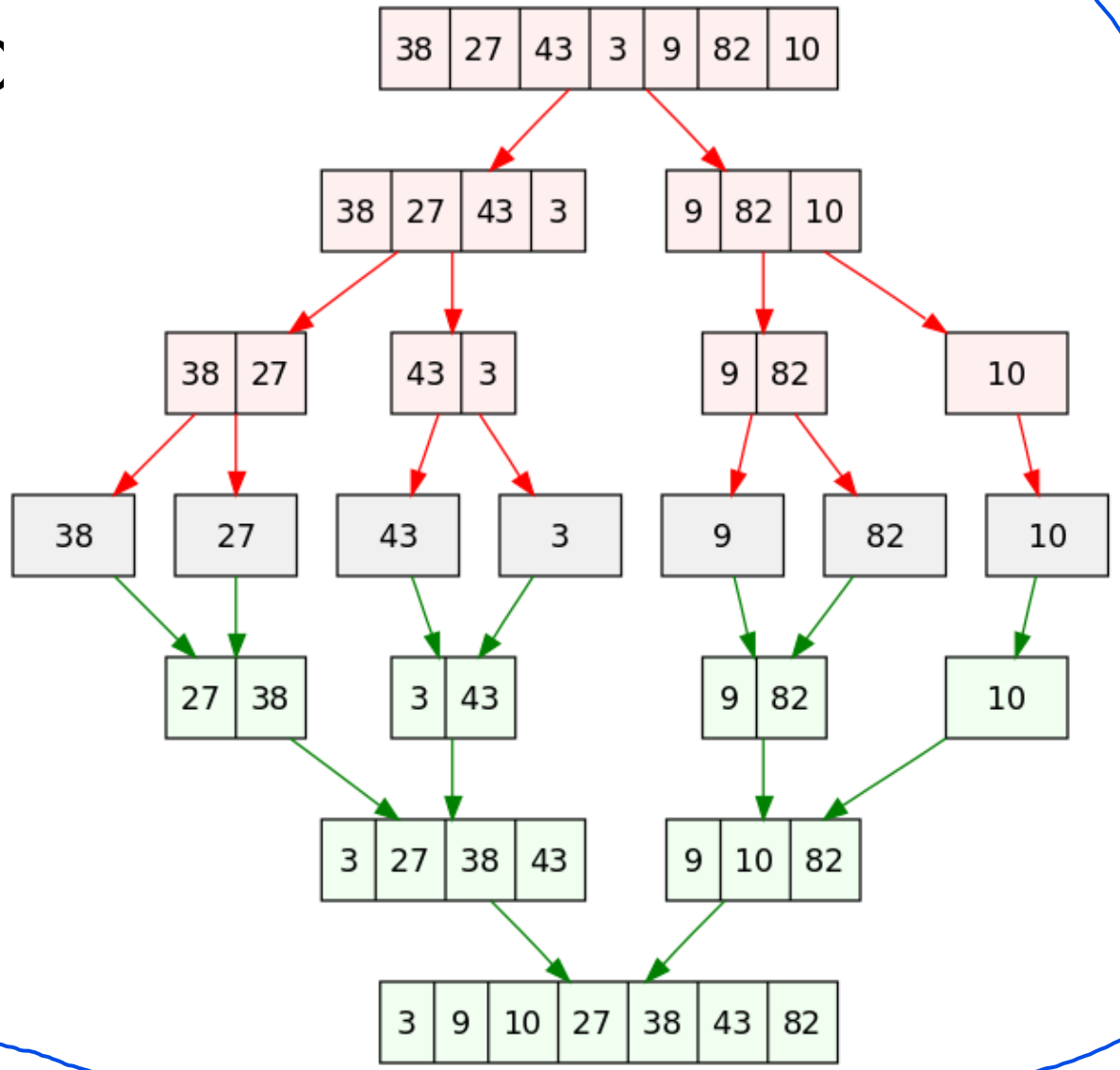
Merge sort visualization

- https://www.hackerearth.com/practice/algorithms/sorting/merge-sort/visualize/

Merge Sort (revisit)

- Highly efficient sorting algorithm that uses a divide-and-conquer approach.
- **Partition the Array**: Rearrange the array elements so that elements less than the pivot are on the left side, and elements greater than the pivot are on the right side. The pivot element will be placed in its correct sorted position in this process.
- **Recursively Apply Quick Sort**: Apply the same process (choose a pivot, partition the array) to the left and right sub-arrays until the entire array is sorted.

Merge Sc


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82

Workshop 2 – Tracing of Sorting Algorithms

- trace of merge_sort([3, 6, 8, 10, 1, 2, 1]) by drawing.
- Work in your team (group)
  - Submit via the Moodle website in class.

- Example Array: [38, 27, 43, 3, 9, 82, 10]
- **Step 1: Divide the Array**
  - Split the array into two halves:
  - Left: [38, 27, 43, 3]
  - Right: [9, 82, 10]
- **Step 2: Recursively Divide Each Half**
  - Left [38, 27, 43, 3]:
    - Split into:
      - Left: [38, 27]
      - Right: [43, 3]
    - Split [38, 27] into:
      - Left: [38]
      - Right: [27]
    - Split [43, 3] into:
      - Left: [43]
      - Right: [3]
  - Right [9, 82, 10]:
    - Split into:
      - Left: [9, 82]
      - Right: [10]
    - Split [9, 82] into:
      - Left: [9]
      - Right: [82]
- **Step 3: Merge Sorted Sublists**
  - Merge [38] and [27] to get [27, 38].
  - Merge [43] and [3] to get [3, 43].
  - Merge [27, 38] and [3, 43] to get [3, 27, 38, 43].
  - Merge [9] and [82] to get [9, 82].
  - Merge [9, 82] and [10] to get [9, 10, 82].
- **Step 4: Final Merge**
  - Merge [3, 27, 38, 43] and [9, 10, 82] to get the final sorted array: [3, 9, 10, 27, 38, 43, 82].

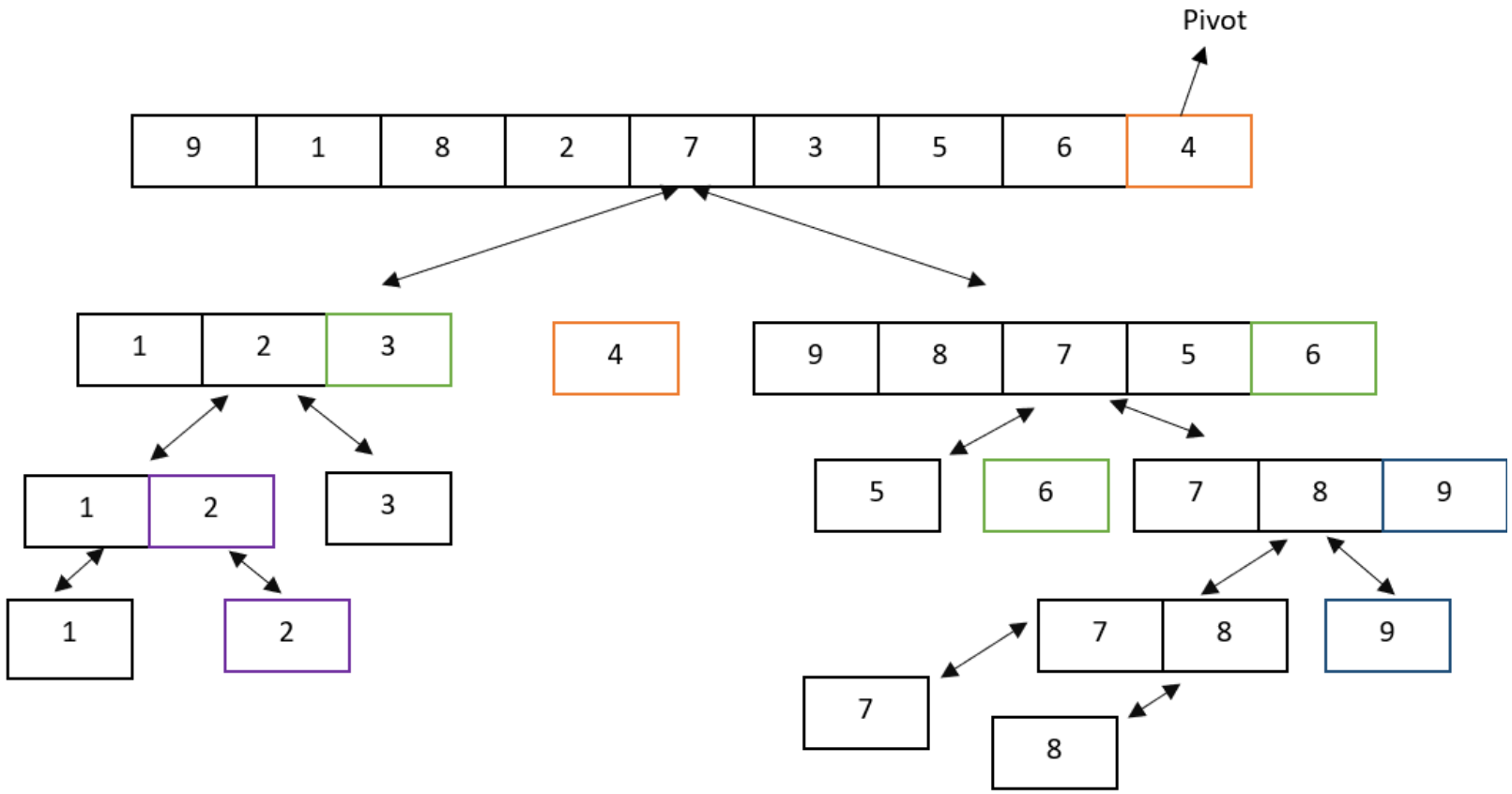
Quick sort Visualization

- https://www.hackerearth.com/practice/algorithms/sorting/quick-sort/visualize/
- https://opendsa-server.cs.vt.edu/embed/quicksortAV

Quick Sort (revisit)

- A popular and efficient sorting algorithm that uses the **divide-and-conquer** strategy.
- Developed by Tony Hoare in 1959 and is widely used because of its performance in practical scenarios.

Steps of Quicksort:



Why quick sort is faster than bubble sort

- Quick Sort works by dividing the array into smaller subarrays and sorting those subarrays recursively.
- Reduces the problem size rapidly, and after partitioning, each subarray is much smaller, which makes the sorting process more efficient.
- Bubble Sort does not divide the problem but repeatedly loops through the entire array, making adjacent comparisons and swaps, leading to many redundant operations.