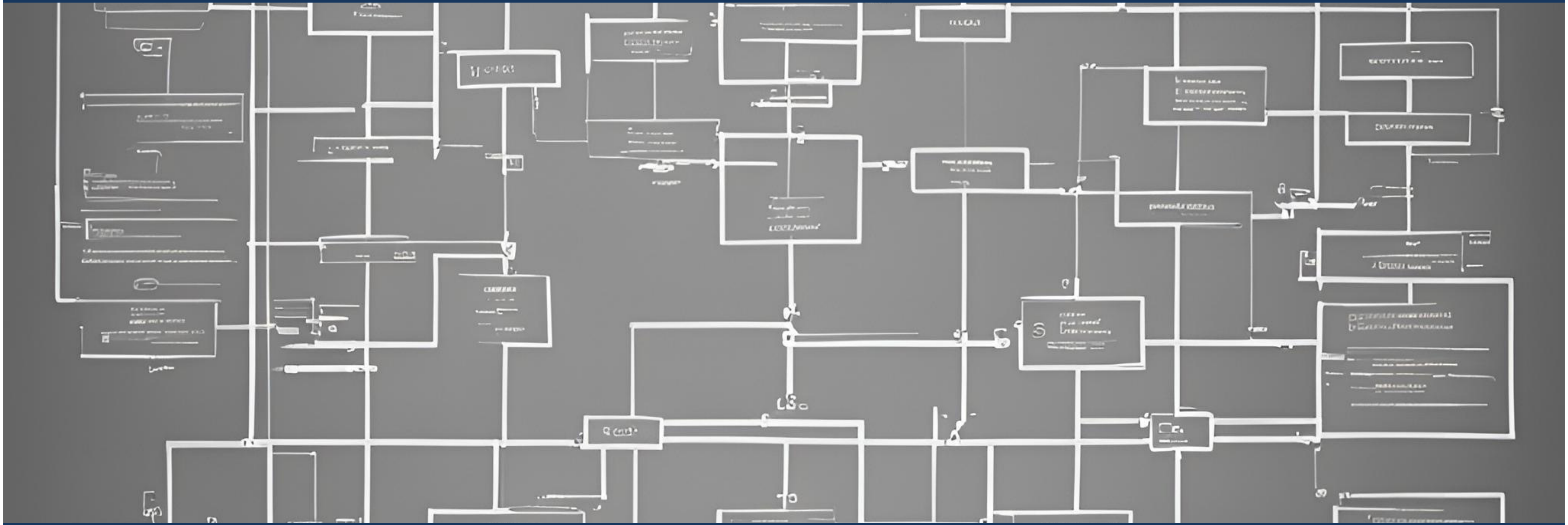
ITM 517 Algorithm

Ja-Hee Kim

Divided and Conquer



Introduction

Divide and conquer

- Divide and Conquer is an algorithmic paradigm. A typical Divide and Conquer algorithm solves a problem using following three steps.
    - **Divide**: Break the given problem into sub-problems of same type.
    - **Conquer**: Recursively solve these sub-problems
    - **Combine**: Appropriately combine the answers

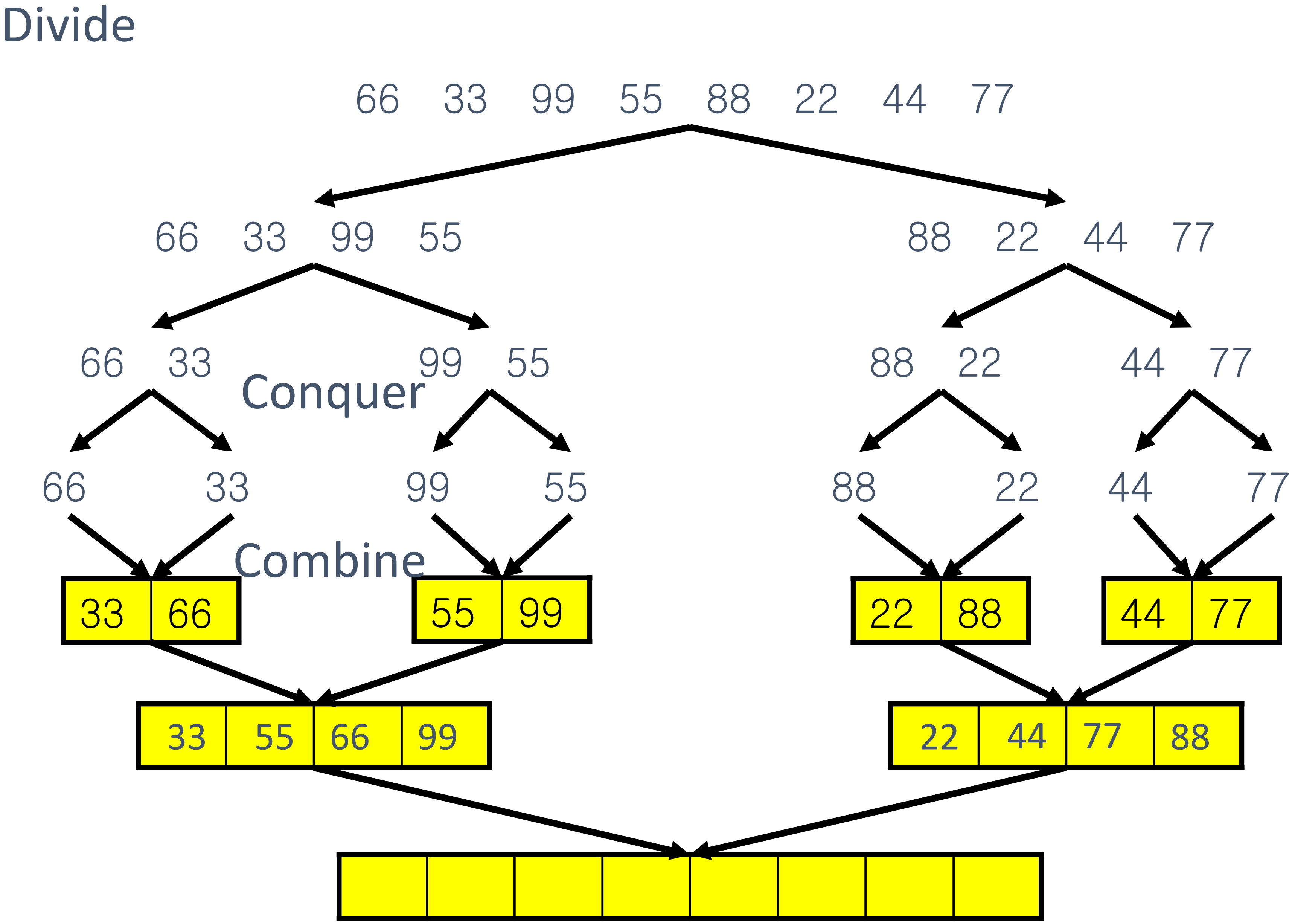
Recursive method

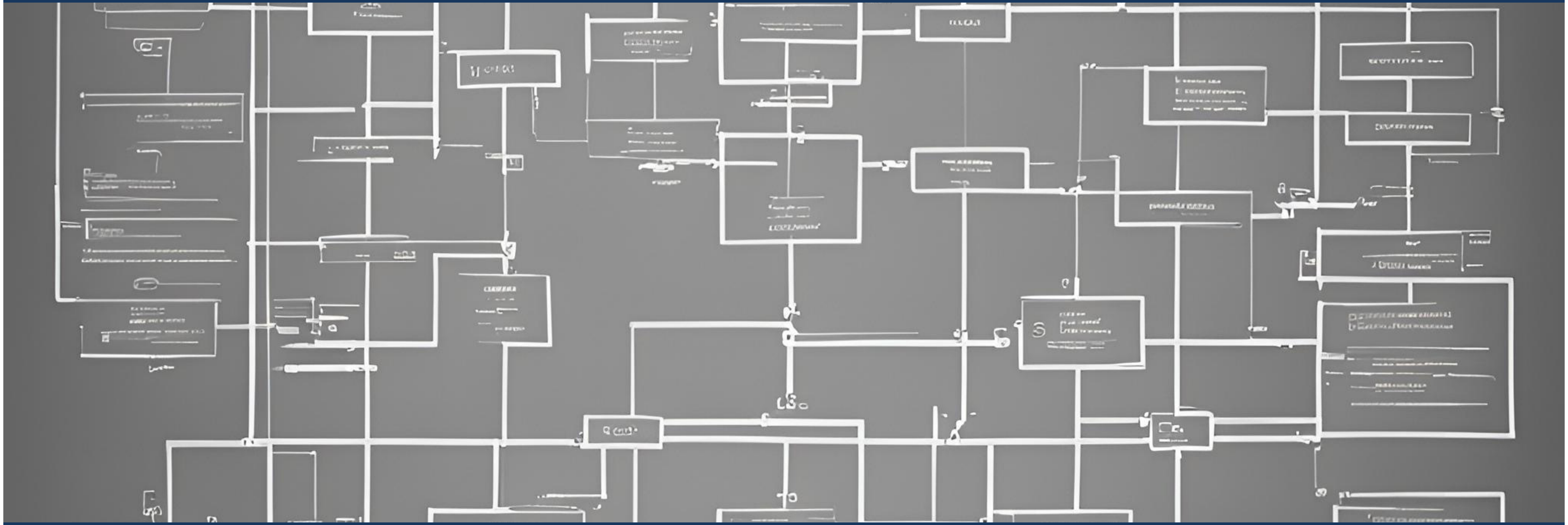
- Some recursive methods use the divide and conquer paradigm

- because it solves a problem by reducing it to smaller sub-problems, hoping that their solutions can be used to solve the larger problem.

- Non-recursive method is usually faster.(for example dynamic programming)

Example: Merge sort





Techniques

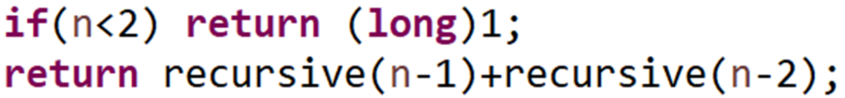
Mathematical induction

- to prove a given statement about any well-ordered set.
- The proof consists of two steps:
  - The base case: prove that the statement holds for the first natural number n. Usually, n = 0 or n = 1, rarely, n = −1
  - The inductive step: prove that, if the statement holds for some natural number n, then the statement holds for n + 1.
- $n! = 1 \times 2 \times 3 \times \cdots \times n$
- $n! = \begin{cases} 1, & \text{if } n = 0, 1 \\ n(n-1)! & \text{if } n > 1 \end{cases}$

Base case

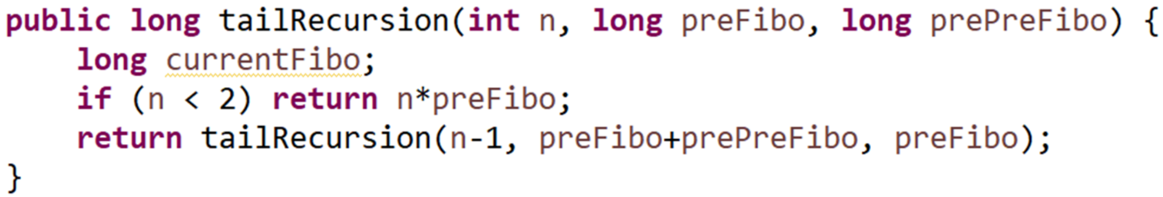
Induction case

Avoiding recursion



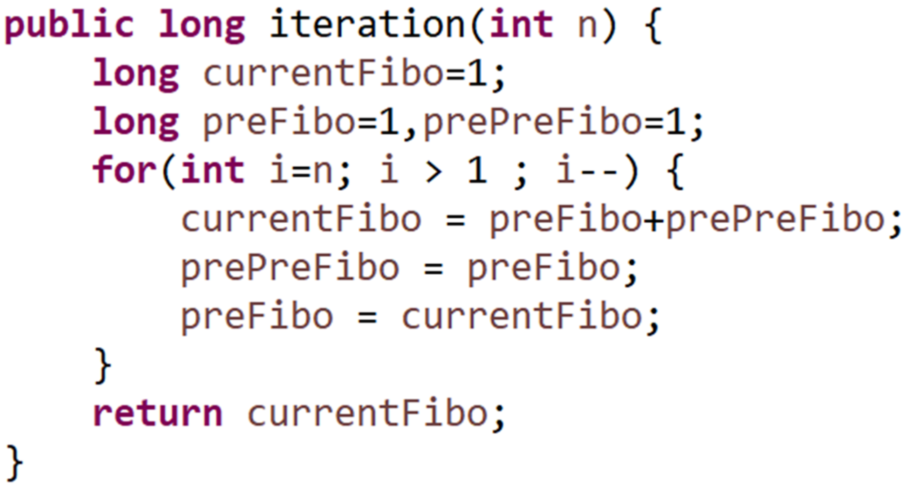
```
if(n<2) return (long)1;
return recursive(n-1)+recursive(n-2);
```

- Tail recursion



```
public long tailRecursion(int n, long preFibo, long prePreFibo) {
    long currentFibo;
    if (n < 2) return n*preFibo;
    return tailRecursion(n-1, preFibo+prePreFibo, preFibo);
}
```

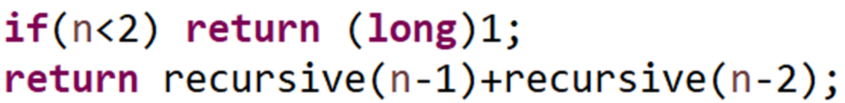
- Iteration



```
public long iteration(int n) {
    long currentFibo=1;
    long preFibo=1,prePreFibo=1;
    for(int i=n; i > 1 ; i--) {
        currentFibo = preFibo+prePreFibo;
        prePreFibo = preFibo;
        preFibo = currentFibo;
    }
    return currentFibo;
}
```

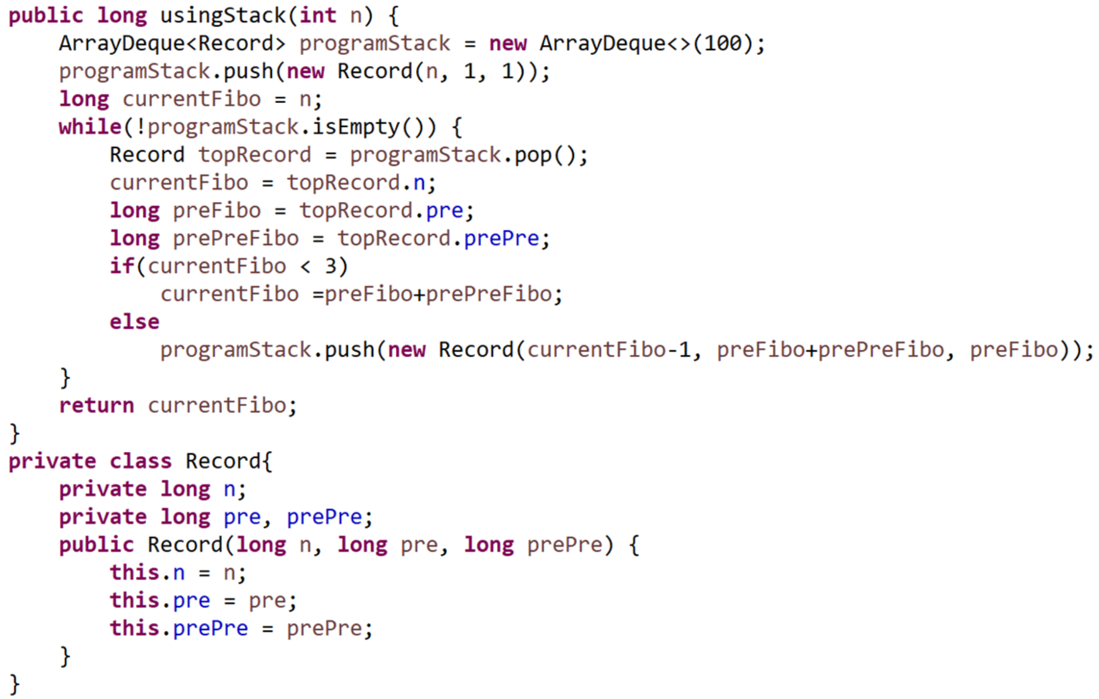
- Using a stack
- Memorize the result

Avoiding recursion



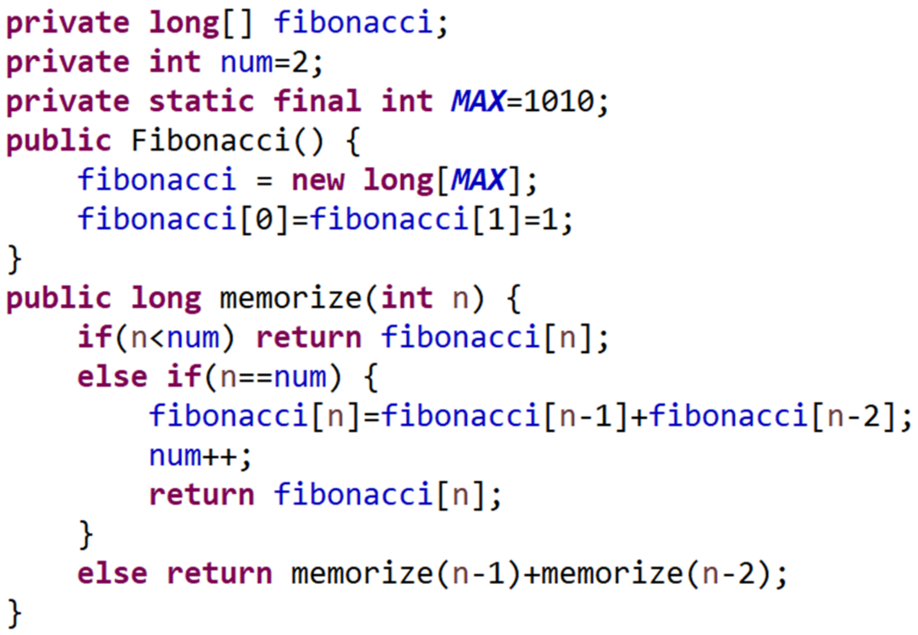
```java
if(n<2) return (long)1;
return recursive(n-1)+recursive(n-2);
```

- Tail recursion
- Iteration
- Using a stack



```java
public long usingStack(int n) {
    ArrayDeque<Record> programStack = new ArrayDeque<>(100);
    programStack.push(new Record(n, 1, 1));
    long currentFibo = n;
    while(!programStack.isEmpty()) {
        Record topRecord = programStack.pop();
        currentFibo = topRecord.n;
        long preFibo = topRecord.pre;
        long prePreFibo = topRecord.prePre;
        if(currentFibo < 3)
            currentFibo =preFibo+prePreFibo;
        else
            programStack.push(new Record(currentFibo-1, preFibo+prePreFibo, preFibo));
    }
    return currentFibo;
}
private class Record{
    private long n;
    private long pre, prePre;
    public Record(long n, long pre, long prePre) {
        this.n = n;
        this.pre = pre;
        this.prePre = prePre;
    }
}
```

- Memorize the result



```java
private long[] fibonacci;
private int num=2;
private static final int MAX=1010;
public Fibonacci() {
    fibonacci = new long[MAX];
    fibonacci[0]=fibonacci[1]=1;
}
public long memorize(int n) {
    if(n<num) return fibonacci[n];
    else if(n==num) {
        fibonacci[n]=fibonacci[n-1]+fibonacci[n-2];
        num++;
        return fibonacci[n];
    }
    else return memorize(n-1)+memorize(n-2);
}
```

Thanks