

CSC 212: Data Structures and Abstractions

Introduction to Analysis of Algorithms

Marco Alvarez

Department of Computer Science and Statistics
University of Rhode Island

Fall 2020



Analysis of Algorithms

Problem, algorithm and program

- **Problem** is a task to be performed
 - ✓ best thought in terms of (well-defined) inputs and outputs
 - ✓ problem definition does not impose constraints on how the problem is solved but often includes resource constraints
- **Algorithm** is a sequence of steps followed to solve a problem
 - ✓ it must be correct and composed of a finite number of concrete steps
 - ✓ there can be no ambiguity
 - ✓ it must terminate
- **Program** is a representation of an algorithm in some programming language

Analysis of algorithms

Algorithm

“Any well-defined computational procedure that takes some value, or set of values, as **input** and produces some value, or set of values, as **output**.”

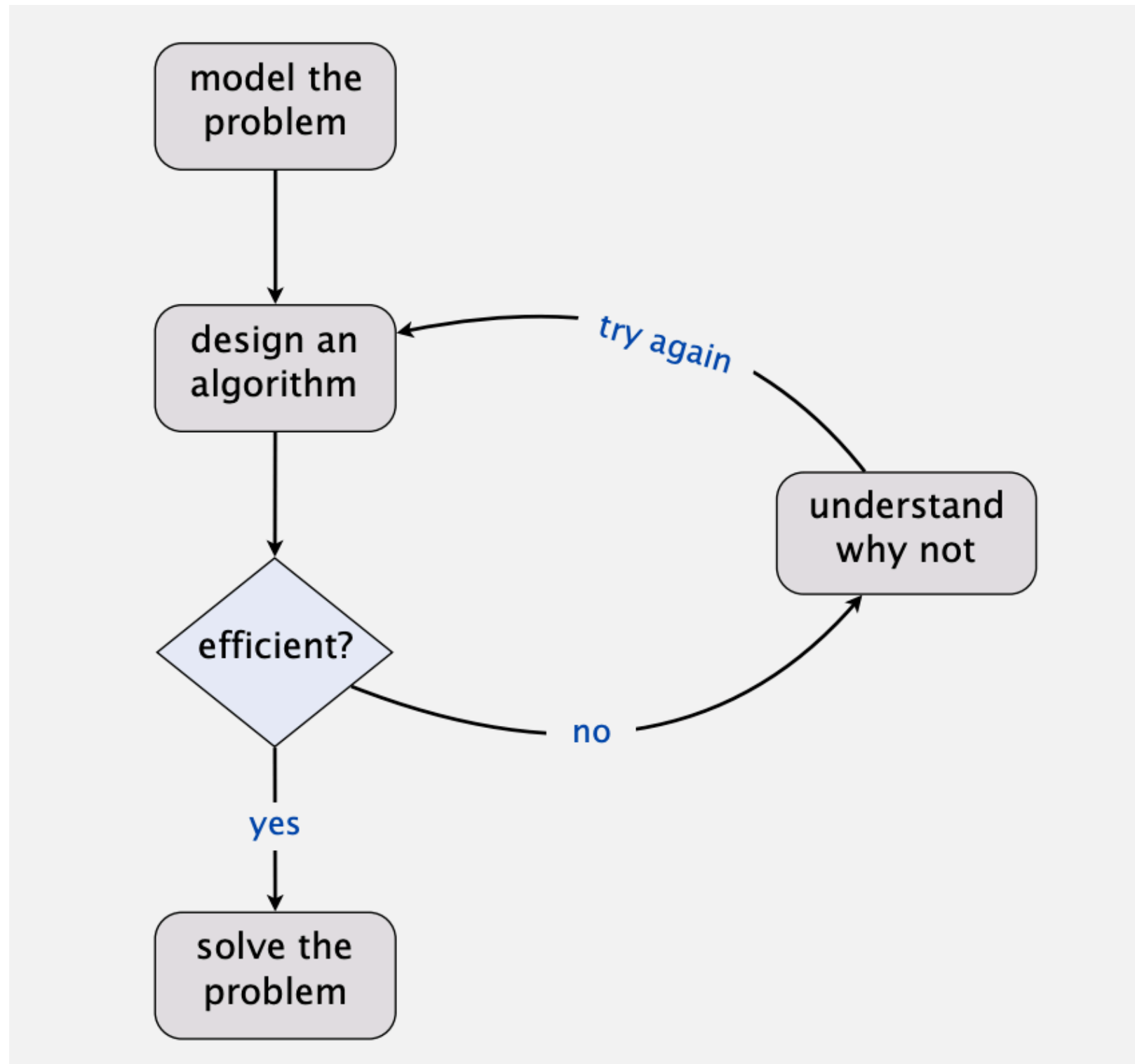
[Cormen et al., Introduction to Algorithms, 3rd. Ed.]

Amount of resources necessary to execute an algorithm?

- ✓ **Time Complexity** (running time)
- ✓ **Space Complexity** (memory)

Resources typically depend on input size

Developing a usable algorithm



Why analysis of algorithms?

- Classify algorithms / problems
- Predict performance / resources
- Provide guarantees
- Understand underlying principles of problems
- and ...



GitHub

Google



Microsoft
Research

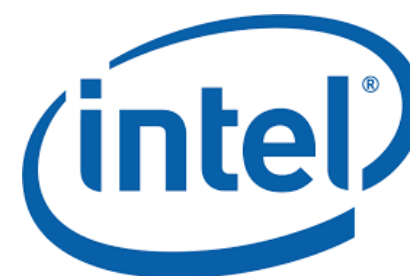
amazon



Palantir



ACTIVISION



IBM
Research



cloudera

LinkedIn



Analyzing computational cost

Empirical Analysis

- ✓ **Run** algorithm
- ✓ Measure actual time

Mathematical Model

- ✓ **Analyze** algorithm
- ✓ Develop Model

Empirical analysis (timing)

- Implement algorithm
- Run on different input sizes
- Record actual running times
- Calculate hypothesis
- Predict and validate

Timing Algorithms

Example 1

e

... mathematical constant that is the base of the natural logarithm. It is approximately equal to 2.71828.

$$\lim_{n \rightarrow \infty} \left(1 + \frac{1}{n} \right)^n$$



Leonhard Euler (1707–1783) was a Swiss mathematician, physicist, astronomer, geographer, logician and engineer who made important and influential discoveries in many branches of mathematics.

$$e = \lim_{n \rightarrow \infty} \left(1 + \frac{1}{n} \right)^n$$

$$e = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \dots$$

Algorithm 1

```
long double euler1(int n) {  
    long double sum = 0;  
    long double fact;  
    for (int i = 0 ; i <= n ; i++) {  
        fact = 1;  
        for (int j = 2 ; j <= i ; j++) {  
            fact *= j;  
        }  
        sum += (1.0 / fact);  
    }  
    return sum;  
}
```

Algorithm 2

```
long double euler2(int n) {  
    long double sum = 0;  
    long double fact = 1;  
    for (int i = 0 ; i <= n ; i++) {  
        sum += (1.0 / fact);  
        fact *= (i+1);  
    }  
    return sum;  
}
```

Which is more efficient?

```
long double euler1(int n) {  
    long double sum = 0;  
    long double fact;  
    for (int i = 0 ; i <= n ; i++) {  
        fact = 1;  
        for (int j = 2 ; j <= i ; j++) {  
            fact *= j;  
        }  
        sum += (1.0 / fact);  
    }  
    return sum;  
}
```

```
long double euler2(int n) {  
    long double sum = 0;  
    long double fact = 1;  
    for (int i = 0 ; i <= n ; i++) {  
        sum += (1.0 / fact);  
        fact *= (i+1);  
    }  
    return sum;  
}
```

Example 2

$$F_0 = 0$$

$$F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2}$$



0 1 1 2 3 5 8 13 21 34 ...

Recursive

```
uint64_t fibR(uint16_t n) {  
    if (n < 2) {  
        return n;  
    } else {  
        return fibR(n-1) + fibR(n-2);  
    }  
}
```

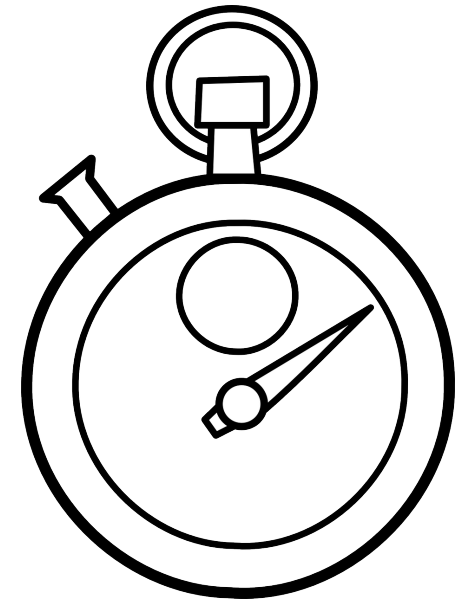
Iterative

```
uint64_t fibI(uint16_t n) {  
    uint64_t sum;  
    uint64_t prev[] = {0, 1};  
  
    if (n < 2) {  
        return n;  
    }  
  
    for (uint16_t i = 2 ; i <= n ; i++ ) {  
        sum = prev[0] + prev[1];  
        prev[0] = prev[1];  
        prev[1] = sum;  
    }  
  
    return sum;  
}
```

Timing ...

```
void time_func(uint16_t n, const char *name) {
    uint64_t val;
    Clock::time_point tic, toc;
    if (! strcmp(name, "Iter")) {
        tic = Clock::now();
        val = fib_iter(n);
        toc = Clock::now();
    }
    if (! strcmp(name, "Rec")) {
        tic = Clock::now();
        val = fib_rec(n);
        toc = Clock::now();
    }
    std::cout << name << " fib(" << n << "):\t" << std::fixed << std::setprecision(4)
    << Seconds(toc-tic).count() << " sec.\tOutput: " << val << std::endl;
}

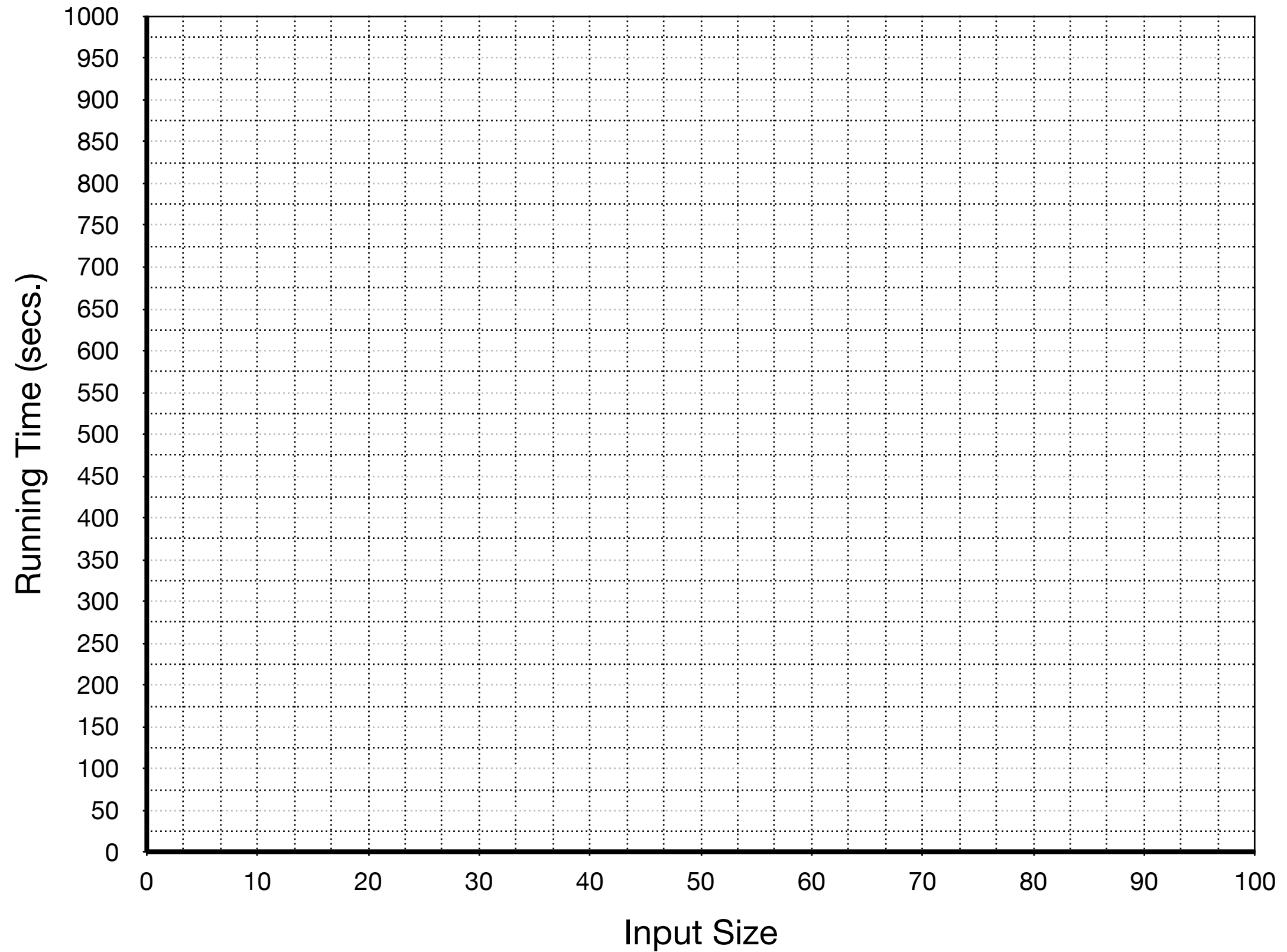
int main(int argc, char **argv) {
    if (argc != 3) {
        std::cout << "Usage: ./fib <n> <alg>\n";
        std::cout << "\t<n>\tn-th term to be calculated\n";
        std::cout << "\t<alg>\talgorithm to be used (Rec or Iter)\n";
        return 0;
    }
    uint16_t n = (uint16_t) atoi(argv[1]);
    time_func(n, argv[2]);
}
```



Iter fib(1):	0.0000 sec.	Output: 1
Iter fib(2):	0.0000 sec.	Output: 1
Iter fib(3):	0.0000 sec.	Output: 2
Iter fib(4):	0.0000 sec.	Output: 3
Iter fib(5):	0.0000 sec.	Output: 5
Iter fib(6):	0.0000 sec.	Output: 8
Iter fib(7):	0.0000 sec.	Output: 13
Iter fib(8):	0.0000 sec.	Output: 21
Iter fib(9):	0.0000 sec.	Output: 34
Iter fib(10):	0.0000 sec.	Output: 55
Iter fib(11):	0.0000 sec.	Output: 89
Iter fib(12):	0.0000 sec.	Output: 144
Iter fib(13):	0.0000 sec.	Output: 233
Iter fib(14):	0.0000 sec.	Output: 377
Iter fib(15):	0.0000 sec.	Output: 610
Iter fib(16):	0.0000 sec.	Output: 987
Iter fib(17):	0.0000 sec.	Output: 1597
Iter fib(18):	0.0000 sec.	Output: 2584
Iter fib(19):	0.0000 sec.	Output: 4181
Iter fib(20):	0.0000 sec.	Output: 6765
Iter fib(21):	0.0000 sec.	Output: 10946
Iter fib(22):	0.0000 sec.	Output: 17711
Iter fib(23):	0.0000 sec.	Output: 28657
Iter fib(24):	0.0000 sec.	Output: 46368
Iter fib(25):	0.0000 sec.	Output: 75025
Iter fib(26):	0.0000 sec.	Output: 121393
Iter fib(27):	0.0000 sec.	Output: 196418
Iter fib(28):	0.0000 sec.	Output: 317811
Iter fib(29):	0.0000 sec.	Output: 514229
Iter fib(30):	0.0000 sec.	Output: 832040
Iter fib(31):	0.0000 sec.	Output: 1346269
Iter fib(32):	0.0000 sec.	Output: 2178309
Iter fib(33):	0.0000 sec.	Output: 3524578
Iter fib(34):	0.0000 sec.	Output: 5702887
Iter fib(35):	0.0000 sec.	Output: 9227465
Iter fib(36):	0.0000 sec.	Output: 14930352
Iter fib(37):	0.0000 sec.	Output: 24157817
Iter fib(38):	0.0000 sec.	Output: 39088169
Iter fib(39):	0.0000 sec.	Output: 63245986
Iter fib(40):	0.0000 sec.	Output: 102334155
Iter fib(41):	0.0000 sec.	Output: 165580141
Iter fib(42):	0.0000 sec.	Output: 267914296
Iter fib(43):	0.0000 sec.	Output: 433494437
Iter fib(44):	0.0000 sec.	Output: 701408733
Iter fib(45):	0.0000 sec.	Output: 1134903170
Iter fib(46):	0.0000 sec.	Output: 1836311903
Iter fib(47):	0.0000 sec.	Output: 2971215073
Iter fib(48):	0.0000 sec.	Output: 4807526976
Iter fib(49):	0.0000 sec.	Output: 7778742049

Rec fib(1):	0.0000 sec.	Output: 1
Rec fib(2):	0.0000 sec.	Output: 1
Rec fib(3):	0.0000 sec.	Output: 2
Rec fib(4):	0.0000 sec.	Output: 3
Rec fib(5):	0.0000 sec.	Output: 5
Rec fib(6):	0.0000 sec.	Output: 8
Rec fib(7):	0.0000 sec.	Output: 13
Rec fib(8):	0.0000 sec.	Output: 21
Rec fib(9):	0.0000 sec.	Output: 34
Rec fib(10):	0.0000 sec.	Output: 55
Rec fib(11):	0.0000 sec.	Output: 89
Rec fib(12):	0.0000 sec.	Output: 144
Rec fib(13):	0.0000 sec.	Output: 233
Rec fib(14):	0.0000 sec.	Output: 377
Rec fib(15):	0.0000 sec.	Output: 610
Rec fib(16):	0.0000 sec.	Output: 987
Rec fib(17):	0.0000 sec.	Output: 1597
Rec fib(18):	0.0000 sec.	Output: 2584
Rec fib(19):	0.0001 sec.	Output: 4181
Rec fib(20):	0.0001 sec.	Output: 6765
Rec fib(21):	0.0001 sec.	Output: 10946
Rec fib(22):	0.0002 sec.	Output: 17711
Rec fib(23):	0.0004 sec.	Output: 28657
Rec fib(24):	0.0006 sec.	Output: 46368
Rec fib(25):	0.0010 sec.	Output: 75025
Rec fib(26):	0.0016 sec.	Output: 121393
Rec fib(27):	0.0026 sec.	Output: 196418
Rec fib(28):	0.0044 sec.	Output: 317811
Rec fib(29):	0.0081 sec.	Output: 514229
Rec fib(30):	0.0113 sec.	Output: 832040
Rec fib(31):	0.0190 sec.	Output: 1346269
Rec fib(32):	0.0309 sec.	Output: 2178309
Rec fib(33):	0.0513 sec.	Output: 3524578
Rec fib(34):	0.0790 sec.	Output: 5702887
Rec fib(35):	0.1345 sec.	Output: 9227465
Rec fib(36):	0.2100 sec.	Output: 14930352
Rec fib(37):	0.3293 sec.	Output: 24157817
Rec fib(38):	0.5225 sec.	Output: 39088169
Rec fib(39):	0.8442 sec.	Output: 63245986
Rec fib(40):	1.3614 sec.	Output: 102334155
Rec fib(41):	2.2176 sec.	Output: 165580141
Rec fib(42):	3.6171 sec.	Output: 267914296
Rec fib(43):	5.9064 sec.	Output: 433494437
Rec fib(44):	9.7282 sec.	Output: 701408733
Rec fib(45):	15.3014 sec.	Output: 1134903170
Rec fib(46):	24.5570 sec.	Output: 1836311903
Rec fib(47):	40.2523 sec.	Output: 2971215073
Rec fib(48):	63.8484 sec.	Output: 4807526976
Rec fib(49):	104.5104 sec.	Output: 7778742049

Hypothesis



Limitations of empirical analysis

- Requires implementing several algorithms for the same problem
 - ✓ may be difficult and time consuming
 - ✓ implementation details also play a role (one particular algorithm may be “better written”)
- Requires extensive testing
 - ✓ time consuming
 - ✓ choice of test cases might favor one of the algorithms
- Variations in HW, SW, and OS affect analysis