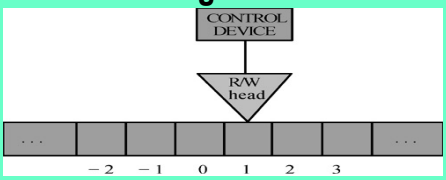
Some Informal Definitions

- **algorithm** - a system of uniquely determined rules that specify successive steps in solving a problem
- **program** - a clearly specified series of computer instructions implementing the algorithm
- **elementary operation** - a computer instruction executed in a single time unit (computing step)
- **running** (computing) **time** of an algorithm - a number of its computing steps (elementary operations)

Turing Machine



- control device: finite states Q
- moving read / write head:
  - read / write a symbol from / to an infinite tape
  - move left / right or stay

Efficiency of Algorithms

How to compare algorithms / programs:

- by **domain of definition** – legal inputs
- by **correctness** – correct output for each legal input (in fact, you need a **formal proof**!)
- by **basic resources** – ***maximum*** or ***average*** requirements: **computing time** and **memory space**

Simple Example 1: $F(n) = 2^n$

- Implicit formula: $F(n) = 2F(n-1)$; $F(0) = 1$, or $F(n) = F(n-1) + F(n-1)$; $n = 1,2,\ldots$
- 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, …
- Explicit formula with $F(1) = 1$: $F(n) = 2^n$
- **Proof with math induction:**
  - **Base condition:** $F(0) = 2^0 = 1$
  - **Induction hypothesis:** $F(n) = 2F(n-1) = 2 \cdot 2^{n-1} = 2^n$

Mathematical Induction

Instead of directly proving a relationship $T(n)$, math induction proves the following conditions:

1. **Base condition** : $T(n_{base})$, e.g., T(0) or T(1)
2. **Inductive hypothesis**: for every $n > n_{base}$ if $T(n-1)$ holds for $n-1$, then $T(n)$ holds for n; or

2'. **Strong induction**: if $T(k)$ holds for $k = n_{base}, \ldots, n-1$, then $T(n)$ holds for n

Example 2: Fibonacci Numbers

- 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, …
- Implicit formula: $F(n) = F(n-1) + F(n-2)$
- Analysis: let's derive the explicit formula for $F(n)$
  - **characteristic equation: assume** $F(n) = c\phi^n$; $1 < \phi < 2 \Rightarrow$ $c\phi^n = c\phi^{n-1} + c\phi^{n-2} \Rightarrow \phi^2 = \phi + 1 \Rightarrow \phi_{1,2} = (1 \pm \sqrt{5})/2$
  - **general solution: linear combination** $F(n) = c_1\phi_1^n + c_2\phi_2^n$
  - $F(1) = F(2) = 1 \Rightarrow$ $F(n) = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n$

Example 2: Euclid's Algorithm

- The greatest common divisor, $k = \mathrm{GCD}(n,m)$ is the greatest positive integer such that it divides both two positive integers m and n
- ***"Brute-force"* solution**: to exhaust all the integers from 1 to the minimum of m and n
- Is it practicable to use such algorithm to find **GCD(3 787 776 332, 3 555 684 776)** or even **GCD(9245,7515)?**

Euclid's GCD Algorithm

- *Euclid's analysis*: if k divides both m and n, then it divides their difference ($n - m$ if $n > m$):

$$\mathrm{GCD}(n, m) = \mathrm{GCD}(n-m, m)$$

- k divides every difference when the subtraction is repeated up to λ times until $n - \lambda m < m$:

$$\mathrm{GCD}(n, m) = \mathrm{GCD}(\ n \bmod m, m\ )$$

where n mod m, or n modulo m is the reminder of the division of n by m (in Java: n%m, e.g. 13%5 = 3)

Euclid's GCD Algorithm

GCD(9245,7515) = 5

9245 mod 7515 = 1730	7515 mod 1730 = 595
1730 mod 595 = 540	595 mod 540 = 55
540 mod 55 = 45	55 mod 45 = 10
45 mod 10 = 5	10 mod 5 = 0 ⇒**GCD=5**

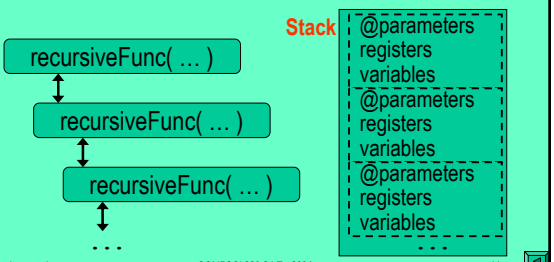
8 steps vs **7515** steps of the brute-force algorithm**!**

How to Run Faster / Save Memory?

- (**RF**) Save results of computations that could be reused later for the same data
- (**RF**) Tabulate functions of one or two integer arguments with relatively small ranges
- (**SM**) **Be careful** with recursive computations (to be sure that the stack is not growing too fast!)
- (**SM**) Free in due time and reuse the allocated memory

Linear Space Increase

```
void recursiveFunc(…) { …; recursiveFunc( … ); …;
```



Exponential Space/Time Increase

```
int fibonacci( int n ) {
  if ( n <= 0 ) return 0; else
  if ( n == 1 ) return 1; else
    return   fibonacci( n – 1)
           + fibonacci( n – 2);
}
```

Processing time / memory space are proportional to fibonacci(n)

fibonacci(1)=1
fibonacci(2)=1
fibonacci(3)=2
fibonacci(3)=2
…
fibonacci(n)≅ $1.618^n/\sqrt{5}$