



# Introduction to Algorithm Analysis

---

Algorithm : Design & Analysis

[1]

**As soon as an Analytical Engine exists, it will necessarily guide the future course of the science. Whenever any result is sought by its aid, the question will then arise – By what course of calculation can these results be arrived at by the machine in the shortest time?**

**- Charles Babbage, 1864**

# Introduction to Algorithm Analysis

---

- Goal of the Course
  - Algorithm, the concept
  - Algorithm Analysis: the criteria
  - Average and Worst-Case Analysis
  - Lower Bounds and the Complexity of Problems
-

# Goal of the Course

---

- Learning to **solve real problems** that arise frequently in computer application
  - Learning the basic principles and techniques used for answering the question: “**How good**, or, how bad is the algorithm”
  - Getting to know a group of “very difficult problems” categorized as “**NP-Complete**”
-

# Design and Analysis, in general

---

## ■ Design

- Understanding the goal
- Select the tools
- What components are needed
- How the components should be put together
- Composing functions to form a process

## ■ Analysis

- How does it work?
  - Breaking a system down to known components
  - How the components relate to each other
  - Breaking a process down to known functions
-

# Problem Solving

---

- In general
    - Understanding the problem
    - Selecting the strategy
    - Giving the steps
    - Proving the correctness
    - Trying to improve
  - Using computer
    - Describing the problem:
    - Selecting the strategy:
    - Algorithm:
      - Input/Output/Step:
    - Analysis:
      - Correct or wrong
      - “good” or “bad”
    - Implementation:
    - Verification:
-

# Probably the Oldest Algorithm

## ■ Euclid algorithm

■ input: nonnegative integer  $m, n$

■ output:  $\text{gcd}(m, n)$

← Specification

■ procedure

■ E1.  $n$  divides  $m$ , the remainder  $\rightarrow r$

■ E2. if  $r = 0$  then return  $n$

■ E3.  $n \rightarrow m; r \rightarrow n; \text{goto E1}$

The Problem:

Computing the ***greatest common divisor*** of two nonnegative integers

# Euclid Algorithm: Recursive Version

## ■ Euclid algorithm

- input: nonnegative integer  $m, n$
- output:  $\text{gcd}(m, n)$

Specification

- procedure

```
Euclid(int  $m, n$ )
```

```
  if  $n=0$ 
```

```
    then return  $m$ 
```

```
    else return Euclid( $n, m \bmod n$ )
```

Recursion

Algorithm  
Pseudocode

# Sequential Search, another Example

- Procedure:

```
Int seqSearch(int[] E, int n, int K)
    int ans, index;
    ans=-1;
    for (index=0; index<n; index++)
        if (K==E[index])
            ans=index;
            break;
Return ans;
```

## The Problem:

Searching a list for a specific key.

## Input:

an unordered array E with n entries, a key K to be matched

## Output:

the location of K in E (or *fail*)



# Algorithmically Solvable Problem

---

- Informally speaking
    - A problem for which a computer program can be written that will produce the correct answer for any valid input if we let it run long enough and allow it as much storage space as it needs.
  - Unsolvabe(or un-decidable) problem
    - Problems for which no algorithms exist
    - the Halting Problem for Turing Machine
-

# Computational Complexity

---

- Formal theory of the complexity of computable functions
  - The complexity of specific problems and specific algorithms
-

# Criteria for Algorithm Analysis

---

- Correctness
  - Amount of work done
  - Amount of space used
  - Simplicity, clarity
  - Optimality
-

# Correctness

---

- Describing the “correctness”: the specification of a specified problem:
    - Preconditions vs. post-conditions
  - Establishing the method:
    - Preconditions+Algorithm  $\Rightarrow$  post-conditions
  - Proving the correctness of the implementation of the algorithm
-

# Correctness of Euclid Algorithm

GCD recursion theorem:

For any nonnegative integer  $a$  and positive integer  $b$ :  $\gcd(a,b) = \gcd(b, (a \bmod b))$

Proof:  $\gcd(a,b) \mid \gcd(b, (a \bmod b))$ , and  $\gcd(b, (a \bmod b)) \mid \gcd(a,b)$

## ■ Euclid algorithm

■ input: nonnegative integers  $m, n$

■ output:  $\gcd(m,n)$

■ procedure

**Euclid**(int  $m,n$ )

**if**  $n=0$

**then return**  $m$

**else return** **Euclid**( $n, \underline{m \bmod n}$ )

**if  $d$  is a common divisor of  $m$  and  $n$ , it must be a common divisor of  $n$  and  $(m \bmod n)$**

**1  $(m \bmod n)$  is always less than  $n$ , so, the algorithm must terminate**

2

1

# How to Measure?

---

- Not too general
    - Giving some indication to make useful comparison for algorithms
  - Not too precise
    - Machine independent
    - Language independent
    - Programming style independent
    - Implementation independent
-

# Focusing the View

---

- Counting the number of the passes through a loop while ignoring the size of the loop
  - The operation of interest
    - Search or sorting an array      comparison
    - Multiply 2 matrices      multiplication
    - Find the gcd      bits of the inputs
    - Traverse a tree      processing an edge
    - Non-iterative procedure      procedure invocation
-

# Presenting the Analysis Results

---

- Amount of work done usually depends on the size of the inputs
  - Amount of work done usually doesn't depend on the size solely
-



# Worst-case Complexity

---

- Worst-case complexity, of a specified algorithm  $A$  for a specified problem  $P$  of size  $n$ :
    - Giving the maximum number of operations performed by  $A$  on any input of size  $n$
    - Being a function of  $n$
    - Denoted as  $W(n)$
  - $W(n) = \max \{ t(I) \mid I \in D_n \}$ ,  $D_n$  is the set of input
-

# Worst-Case Complexity of Euclid's

```
Euclid(int  $m, n$ )  
  if  $n=0$   
    then return  $m$   
    else return Euclid( $n, m \bmod n$ )
```

measured by the number  
of recursive calls

- For any integer  $k \geq 1$ , if  $m > n \geq 1$  and  $n < F_{k+1}$ , then the call  $\text{Euclid}(m, n)$  makes **fewer** than  $k$  recursive calls. (to be proved)
  - Since  $F_k$  is approximately  $\phi^k / \sqrt{5}$ , the number of recursive calls in  $\text{Euclid}$  is  $O(\lg n)$ .

For your reference:

$$\phi = (1 + \sqrt{5})/2 \approx 1.6180\dots$$

# Euclid Algorithm and Fibonacci

---

- If  $m > n \geq 1$  and the invocation  $\text{Euclid}(m, n)$  performs  $k \geq 1$  recursive calls, then  $m \geq F_{k+2}$  and  $n \geq F_{k+1}$ .
    - Proof by induction
    - Basis:  $k=1$ , then  $n \geq 1 = F_2$ . Since  $m > n$ ,  $m \geq 2 = F_3$ .
    - For larger  $k$ ,  $\text{Euclid}(m, n)$  calls  $\text{Euclid}(n, m \bmod n)$  which makes  $k-1$  recursive calls. So, by inductive hypothesis,  $n \geq F_{k+1}$ ,  $(m \bmod n) \geq F_k$ .  
Note that  $m \geq n + (m - \lfloor m/n \rfloor n) = n + (m \bmod n) \geq F_{k+1} + F_k = F_{k+2}$
-

# The Bound is Tight

---

- The upper bound for  $\text{Euclid}(m,n)$  is tight, by which we mean that: “if  $b < F_{k+1}$ , the call  $\text{Euclid}(a,b)$  makes fewer than  $k$  recursive calls” is best possible result.
  - There do exist some inputs for which the algorithm makes the same number of recursive calls as the upper bound.
    - $\text{Euclid}(F_{k+1}, F_k)$  recurs exactly  $k-1$  times.
-

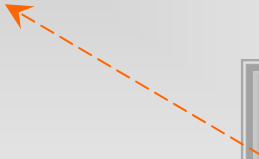
# Average Complexity

---

- Weighted average  $A(n)$

$$A(n) = \sum_{I \in D_n} \Pr(I) t(I)$$

- How to get  $\Pr(I)$ 
  - Experiences
  - Simplifying assumption
  - On a particular application



$\Pr(I)$  is the probability of occurrence of input  $I$

---

# Average Behavior Analysis of Sequential Search

---

- Case 1: assuming that  $K$  is in  $E$ 
    - Assuming no same entries in  $E$
    - Look *all* inputs with  $K$  in the  $i$ th location as *one* input (so, inputs totaling  $n$ )
    - Each input occurs with equal probability (i.e.  $1/n$ )
  - $A_{\text{succ}}(n) = \sum_{i=0..n-1} \Pr(I_i | \text{succ}) t(I_i)$ 
$$= \sum_{i=0..n-1} (1/n)(i+1)$$
$$= (n+1)/2$$
-

# Average Behavior Analysis of Sequential Search

---

- Case 2: K may be not in E
    - Assume that  $q$  is the probability for K in E
    - $A(n) = \text{Pr}(\text{succ})A_{\text{succ}}(n) + \text{Pr}(\text{fail}) A_{\text{fail}}(n)$   
 $= q((n+1)/2) + (1-q)n$
  - *Issue for discussion:*  
*Reasonable Assumptions*
-

# Optimality

---

- “The best possible”

How much work is necessary and sufficient to solve the problem.

- Definition of the optimal algorithm

- For problem  $P$ , the algorithm  $A$  does at most  $W_A(n)$  steps in the worst case (upper bound)
  - For some function  $F$ , it is provable that for any algorithm in the class under consideration, there is some input of size  $n$  for which the algorithm must perform at least  $F(n)$  steps (lower bound)
  - If  $W_A = F$ , then  $A$  is optimal.
-



# Complexity of the Problem

---

- $F$  is a lower bound for a class of algorithm means that: *For any algorithm in the class, and any input of size  $n$ , there is some input of size  $n$  for which the algorithm must perform at least  $F(n)$  basic operations.*
-

# Establishing a lower bound

- Procedure:

```
int findMax(E,n)
max=E(0)
for (index=1;index<n;index++)
    if (max<E(index))
        max=E(index);
return max
```

- Lower bound

For any algorithm A that can compare and copy numbers exclusively, if A does fewer than  $n-1$  comparisons in any case, we can always provide a right input so that A will output a wrong result.

The problem:

**Input:** number array E  
with n entries indexed as  
 $0, \dots, n-1$

**Output:** Return max, the  
largest entry in E

# Bounds: Upper and Lower

---

- **For a specific algorithm** (to solve a given problem), the “**upper bound**” is a cost value **no less than** the maximum cost for the algorithm to deal with the worst input.
  - **For a given problem**, the “**lower bound**” is a cost value **no larger than** any algorithm (known or unknown) can achieve for solving the problem.
  - A computer scientist want to make the two bounds meet.
-

# Home Assignment

---

- pp.61 –
    - 1.5
    - 1.12
    - 1.16 – 1.19
  - Additional
    - Other than speed, what other measures of efficiency might one use in a real-world setting?
    - Come up with a real-world problem in which only the best solution will do. Then come up with one in which a solution that is “approximately” the best is good enough.
-

# Algorithm vs. Computer Science

---

- Sometimes people ask: “What really is computer science? Why don’t we have telephone science? Telephone, it might be argued, are as important to modern life as computer are, perhaps even more so. A slightly more focused question is whether computer science is not covered by such classical disciplines as mathematics, physics, electrical engineering, linguistics, logic and philosophy.

We would do best not to pretend that we can answer these questions here and now. The hope, however, is that the course will implicitly convey something of the uniqueness and universality of the study of algorithm, and hence something of the importance of computer science as an autonomous field of study.

---

- adapted from Harel: “*Algorithmics, the Spirit of Computing*”

# References

---

## ■ Classics

- Donald E.Knuth. *The Art of Computer Programming*
  - Vol.1 Fundamental Algorithms
  - Vol.2 Semi-numerical Algorithms
  - Vol.3 Sorting and Searching

## ■ Popular textbooks

- Thomas H.Cormen, etc. *Introduction to Algorithms*
- Robert Sedgewick. *Algorithms* (with different versions using different programming languages)

## ■ Advanced mathematical techniques

- Graham, Knuth, etc. *Concrete Mathematics: A Foundation for Computer Science*
-

# You Have Choices

---

- Design Techniques Oriented Textbooks
    - Anany Levitin. *Introduction to the Design and Analysis of Algorithms*
    - M.H.Alsuwaiyel. *Algorithms Design Techniques and Analysis*
    - G.Brassard & P.Bratley: *Fundamentals of Algorithmics*
  - Evergreen Textbook
    - Aho, Hopcroft and Ullman. *The Design and Analysis of Computer Algorithm*
  - Something New
    - J.Kleinberg & E.Tardos: *Algorithm Design*
-