

Question: Given 8 identical coins, there is a fake coin, which may be either heavier or lighter. Assume that there is a balancing-scale available, to identify that fake coin,

- how many rounds of comparisons are needed?
- how many rounds of comparisons are required, if the number of coins is not 8 but n instead?

0.1 What is an Algorithm?

An **algorithm** is any well-defined computational procedure that takes some value or a set of values as **input**, and produces some value or a set of values as **output**.

Example one: Sort n integers in increasing order.

Input: a sequence of n numbers, a_1, a_2, \dots, a_n

Output: a permutation (rearrangement) of the numbers as a'_1, a'_2, \dots, a'_n , such that $a'_1 \leq a'_2 \leq \dots \leq a'_n$.

Example two: Given an integer N , determine whether it is a prime number.

Input: N

Output: YES if N is a prime number, otherwise, NO.

- Algorithms
- Algorithm Analysis
- Algorithm Design

0.1 The Definition of Algorithms (cont.)

For a given problem, there are many algorithms for it, e.g., there are a number of algorithms for the sorting problem.

1. insertion sort
2. bubble sort
3. mergesort
4. quicksort
5. shellsort
6. heapsort
7. bucket sort

0.2 Algorithm Analysis

Analysis of an algorithm is to predict the **resources** that the algorithm requires. Resources are measured in relation to a **model of computation**, which is a mathematical abstraction of a computer on which the algorithm is implemented.

In the following several popular computational models are listed.

- ▶ RAM: time and space (traditional serial computers)
- ▶ PRAM: parallel time, number of processors, and read-and-write restrictions (SIMD type of parallel computers)
- ▶ Message Passing Model: communication cost (number of messages), and computational cost (usually the cost for local computation is ignored) (Distributed computing, peer-to-peer networking, MIMD type of machine)
- ▶ Turing Machine: time and space (abstract theoretical machine)

0.2 Algorithm Analysis (continued)

Important concepts used in the analysis of algorithms:

- ▶ Input size: If the input to an algorithm is "5413133", is the input size 1, 7, 23 or 5413133?
- ▶ Running time (worst-case and average-case): The running time of an algorithm on a particular input is the number of primitive operations or steps executed. However there is more than one idea about what a primitive operation is.
- ▶ Order of growth: To simplify the analysis of algorithms, we are interested in the growth rate of the running time, i.e., we only consider the leading terms of a time formula. e.g., the leading term is n^2 in the expression $n^2 + 100n + 50000$.

0.2 Algorithm Analysis (continued)

Most analysis of algorithms of the type we will encounter in this course is done using the following assumptions.

- ▶ RAM (random access machine) model with unlimited memory.
- ▶ Operations such as arithmetic with small integers, boolean operations, fetching and storing small quantities, take unit time.
- ▶ Worst case performance is emphasised, but average and amortised performance is also important.

Many other conventions exist (example: bit complexity) but we will not encounter them very much.

0.2 Algorithm Analysis (continued)

Insertion Sorting

```
Algorithm Insertion(A)
0    $n \leftarrow \text{length}[A]$ 
1   for  $j \leftarrow 2$  to  $n$  do
2        $key \leftarrow A[j]$ 
3   /* Insert  $A[j]$  into sorted sequence  $A[1 \dots j - 1]$  */
4        $i \leftarrow j - 1$ 
5       while  $i > 0$  and  $A[i] > key$  do
6            $A[i + 1] \leftarrow A[i]$ 
7            $i \leftarrow i - 1$ 
8        $A[i + 1] \leftarrow key$ 
```

E.g., analyze insertion sort algorithm: time and space.

See page 24.

0.3 Algorithms Design

For a given problem, there are many ways to design algorithms for it, (e.g. Insertion sort is an incremental approach). The following is a list of several popular design approaches.

- Divide-and-Conquer (D&C)
- Greedy Approach
- Dynamic Programming
- Linear Programming
- Branch-and-Bound
- α - β Pruning

Divide-and-Conquer (D&C) Paradigm

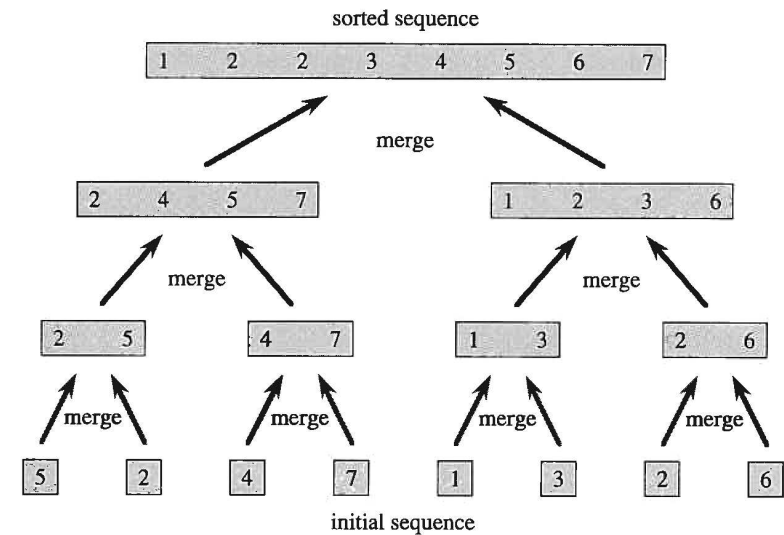
Given a problem \mathcal{A} , if the D&C strategy is applicable, the problem is then solved using the strategy. The detailed steps are as follows.

1. **Divide** \mathcal{A} into a number of subproblems with the same size roughly.
2. **Conquer** the subproblems by solving them recursively. For small enough subproblems, solve them directly without dividing them further.
3. **Combine** the solutions of the subproblems into the solution for the original problem.

Divide-and-Conquer (D&C) Paradigm (cont.)

An Example of D&C Strategy: Consider the merge-sort algorithm.

- **Divide:** divide the n -element sequence into two subsequences of $n/2$ elements each (or as close as possible if n is odd)
- **Conquer:** sort the two subsequences recursively using the merge sort
- **Combine:** merge the two sorted subsequences into a sorted sequence



Cormen, p33