CONTENTS

Problem Solving

- Algorithm Design
- Algorithm Analysis
  - Time Complexity
  - Space Complexity
- Experimental Approach
- Theoretical Approach
- Algorithm Example

Problem Solving: Main Steps

1. Problem definition
2. Algorithm design / Algorithm Specification
3. Algorithm analysis / Performance
4. Implementation
5. Testing
6. Maintenance

1. Problem Definition

- What is the task to be accomplished?
  - Calculate the average of the grades for a given student
  - Understand the talks given out by politicians and translate them in Chinese
- What are the time / space / speed / performance requirements ?

2. Algorithm Design / Specifications

- Algorithm: Finite set of instructions that, if followed, accomplishes a particular task.
- Criteria to follow:
  - Input: Zero or more quantities (externally produced)
  - Output: One or more quantities
  - Definiteness: Clarity, precision of each instruction
  - Finiteness: The algorithm has to stop after a finite (may be very large) number of steps
  - Effectiveness: Each instruction has to be basic enough and feasible

Algorithm Design Goals

- The two basic design goals that one should strive for in a program are:

1. Try to save Time
2. Try to save Space

- A program that runs faster is a better program,

so saving time is an obvious goal. Like wise,

- a program that saves space over a competing program is considered desirable.

4,5,6: Implementation, Testing, Maintenance

- Implementation
  - Decide on the programming language to use
    - C, C++, Lisp, Java, Perl, Prolog, assembly, etc. , etc.
  - Write clean, well documented code

- Test, test, test….

- Integrate feedback from users, fix bugs, ensure compatibility across different versions → Maintenance

3. Algorithm Analysis/Performance

1. Time complexity
   - How much time does it take to run the algorithm
2. Space complexity
   - How much space is required

1. Time Complexity

- Often more important than space complexity
  - space available (for computer programs!) tends to be larger and larger
  - time is still a problem for all of us

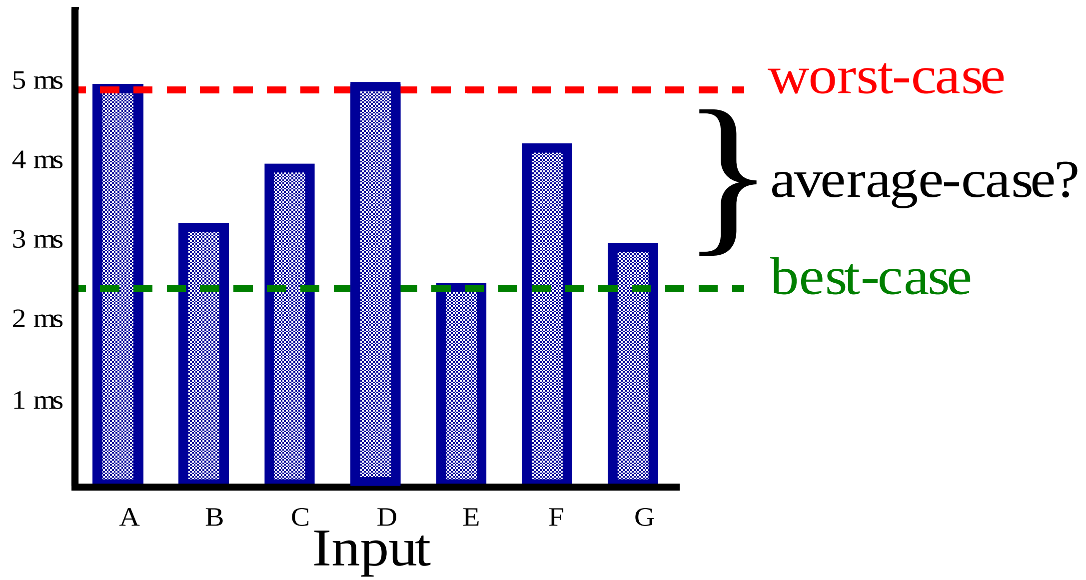
- 3-4GHz processors on the market
  - still …
  - researchers estimate that the computation of various transformations for 1 single DNA chain for one single protein on 1 TerraHZ computer would take about 1 year to run to completion
- Algorithms running time is an important issue

Running Time

- Problem: prefix averages
  - Given an array X
  - Compute the array A such that A[i] is the average of elements X[0] … X[i], for i=0..n-1
- Sol 1
  - At each step i, compute the element X[i] by traversing the array A and determining the *sum* of its elements, respectively the average
- Sol 2
  - At each step i update a *sum* of the elements in the array A
  - Compute the element X[i] as *sum/I*

Big question: Which solution to choose??

Running time



Suppose the program includes an *if-then* statement that may execute or not: → variable running time

Typically Algorithms are measured by their ***worst case***

Space Complexity:

- Space complexity = The amount of memory required by an algorithm to run to completion
- The space need by a program has the following components:
- **Instruction space:** Instruction space is the space needed to store the compiled version of the program instructions.
- **Data space:** Data space is the space needed to store all constant and variable values. Data space has two components:

  -Space needed by constants and simple variables in program.

2. Space Complexity

-Space needed by dynamically allocated objects such as arrays and class instances.

- **Environment stack space:** The environment stack is used to save information needed to resume execution of partially completed functions.
- Some algorithms may be more efficient if data completely loaded into memory
  - Need to look also at system limitations
  - E.g. Classify 2GB of text in various categories [politics, tourism, sport, natural disasters, etc.] – can I afford to load the entire collection?

Space Complexity (cont'd)

1. Fixed part: The size required to store certain data/variables, that is independent of the size of the problem:

   - e.g. name of the data collection

   - same size for classifying 2GB or 1MB of texts

2. Variable part: Space needed by variables, whose size is dependent on the size of the problem:

   - e.g. actual text

   - load 2GB of text VS. load 1MB of text

Space Complexity (cont'd)

- S(P) = c + S(instance characteristics)
  - c = constant
- Example:

```
void float sum (float* a, int n)
{
    float s = 0;
     for(int i = 0; i<n; i++) {
        s+ = a[i];
     }
     return s;
}
```

Space?

one word for n, one for a [passed by reference!], one for i → constant space!

Experimental Approach

- Write a program that implements the algorithm
- Run the program with data sets of varying size.
- Determine the actual running time

- Problems?

Experimental Approach

- It is necessary to implement and test the algorithm in order to determine its running time.
- Experiments can be done only on a limited set of inputs, and may not be indicative of the running time for other inputs.
- The same hardware and software should be used in order to compare two algorithms. – condition very hard to achieve!

Use a Theoretical/Analytical Approach

- Based on high-level description of the algorithms, rather than language dependent implementations
- Makes possible an evaluation of the algorithms that is independent of the hardware and software environments

  → **Generality**

Algorithms (example)

Describe an algorithm for finding the maximum value in a finite sequence of integers.

Solution:

- Set the temporary maximum equal to the first integer in the sequence.
- Compare the next integer in the sequence to the temporary maximum, and if it is larger than the temporary maximum, set the temporary maximum equal to this integer.
- Repeat the previous step if there are more integers in the sequence
- Stop when there are no integers left in the sequence.
- The temporary maximum at this point is the largest integer in the sequence.

Algorithms (example)

Describe an algorithm for finding the maximum value in a finite sequence of integers.

Solution:

```
Procedure max(a1, a2, a3, …, an: integers)
max = a1
for i=2 to n
if max < ai
then max = ai
output max
```

Algorithm Example

- Example: Describe an algorithm for finding the maximum value in a finite sequence of integers.

  (or)

  find the maximum element of an array.

1. **Algorithm** arrayMax(A, *n*):
   Input: An array A storing n integers.
   Output: The maximum element in A.
2. *currentMax* $\leftarrow$ A[0]
3. **for** ***i*** $\leftarrow$ **1 to *n* -1 do**
4. **if** *currentMax* < A[i]
5. **then** *currentMax* $\leftarrow$ A[i]
6. **return** *currentMax*

Low Level Algorithm Analysis

- Based on primitive operations (low-level computations independent from the programming language)
- E.g.:
  - Make an addition = 1 operation
  - Calling a method or returning from a method = 1 operation
  - Index in an array = 1 operation
  - Comparison = 1 operation etc.
- Method: Inspect the pseudo-code and count the number of primitive operations executed by the algorithm