Algorithm Discovery

Topics:

Attributes of Algorithms
Measuring Efficiency

Prof. Dr. Slim Abdennadher

29.11.2007

Attributes of Algorithms

- **Correctness**
  - Give a correct solution to the problem!
- **Ease of understanding**
  - Clarity and ease of handling
- **Program maintenance**
  - Fix errors
  - Extend the program to meet new requirements
- **Efficiency**
  - **Time:** How long does it take to solve the problem?
  - **Space:** How much memory is needed?

A Choice of Algorithms

- Possible to come up with several different algorithms to solve the same problem.
- **Which one is the best?**
  - Most efficient: Time vs. Space
  - Easiest to maintain?
- **How do we measure time efficiency?**
  - Running time?
  - Number of executed operations?

Measuring Efficiency

- Need a **metric** to measure efficiency of algorithms
  - **Running Time:** How long does it take to solve the problem?
    - Depends on machine speed
  - **Number of Operations:** How many operations does the algorithm execute?
    - Better metric but a lot of work to count all operations
  - **Number of Fundamental Operations:** How many "fundamental operations" does the algorithm execute?
- Depends on size and type of input, interested in knowing:
  - **Best-case**, **Worst-case**, **Average-case** behavior
- **Need to analyze the algorithm!**

Example: Average of n numbers (I)

Problem: Find the average of n numbers

```
1. get n,A1,A2,....,An
2. set sum to 0
3. set i to 1
4. while (i <= n)  {
5.    set sum to (sum + Ai)
6.    set i to (i + 1)
   }
7. set average to sum/n
8. print average
```

- **How many steps does the algorithm execute?**
  - Steps 2, 3, 7, and 8 are executed once.
  - Steps 4, 5, and 6 depend on the input size `n`.
- **Total Number of Executed operations:**

$$1 + 1 + (n + 1) + n + n + 1 + 1 = 3n + 5$$

Example: Average of n numbers (II)

Problem: Find the average of n numbers

```
1. get n,A1,A2,....,An
2. set sum to 0 ---------------- 1 operation   --> executed once
3. set i to 1   ---------------- 1 operation   --> executed once
4. while (i <= n)  { ----------- 1 operation   --> (n+1) repetitions
5.    set sum to (sum + Ai) ---- 1 operation   --> n repetitions
6.    set i to (i + 1) --------- 1 operation   --> n repetitions
   }
7. set average to sum/n -------- 1 operation   --> executed once
8. print average --------------- 1 operation   --> executed once
```

Sequential Search – Analysis

```
1. get Name, N1, ...,Nn, T1, ...,Tn
2. set i to 1 and set Found to NO
3.   while (i <= n and FOUND = NO)  {
4.       if Name = Ni then
5.           print Ti
6.           set Found to YES
7.       else set i to i+1           }
8. if Found = NO then
9.  print "Sorry, name not in directory"
```

- **How many steps does the algorithm execute?**
  - Steps 2, 5, 6, 8 and 9 are executed at most once.
  - Steps 3, 4, and 7 depend on input size.
- **Worst case:** Steps 4 and 7 are executed at most `n`-times and step 3 `n+1` times.
- **Best case:** Steps 4 and 7 are executed only once.
- **Average case:** Steps 4 and 7 are executed approximately $(n/2)$-times.

Sequential Search – Worst Case Behavior

Worst-Case: The name is not in the list:

```
1. get Name, N1, ...,Nn, T1, ...,Tn
2. set i to 1 and set Found to NO ------ 2 operations   --> executed once
3.   while (i <= n and FOUND = NO)  { -- 2 operations   --> (n+1) repetitions
4.      if Name = Ni then -------------- 1 operation    --> n repetitions
5.         print Ti
6.         set Found to YES
7.      else set i to i+1 } ------------ 1 operation    --> n repetitions
8. if Found = NO then ------------------ 1 operation    --> executed once
9.  print "Sorry, name not in directory" 1 operation    --> executed once
```

- **Steps 2, 8 and 9 are executed once.**
- **Steps 5 and 6 are not executed.**
- **Step 3 is executed `n+1`-times.**
- **Steps 4 and 7 are executed `n`-times.**
- **Total Number of Executed operations:**

$$2+2\times(n+1)+n+n+2 = 4n+6$$

Order of Magnitude: Big-O Notation

- We are:
  - Not interested in knowing the exact number of operations the algorithm performs.
  - **Mainly interested in knowing how the number of operations grows with increased input size!**
- Why?
  - Given large enough input, the algorithm with faster growth will execute more operations.
- **Order of Magnitude, O(. . .), measures how the number of operations grows with input size n.**

Order of Magnitude

- Not interested in the exact number of operations, for example, algorithms where total operations are:
  - n
  - 6n
  - 6n + 278
  - 5000n + 2000
- are all of order **O(n)**
  - For the previous algorithms, the total number of operations grows approx. proportionally with input size (given large enough `n`).

Linear Algorithms – O(n)

- If the number of operations grows in proportion, or linearly, with input size, it is a **linear** algorithm, **O(n)**.
- **Example:** Sequential search is linear, denoted O(n).

Constant Algorithms – O(1)

- If the number of operations remains the same, e.g. problem size doubles but number of operations remains the same, it is a **constant** algorithm, **O(1)**.
- **Example:** Calculate the sum of all the integers from 1 to n with the Gauss algorithm

```
get n
set result to ((n+1)*n)/2
print result
```

Summary

- We are concerned with the efficiency of algorithms
  - Time- and Space-efficiency
  - Need to analyze the algorithms
- Order of magnitude measures the efficiency
  - E.g. O(1), O(n), O(n^2), ...
  - Measures how fast the work grows as we increase the input size n.
  - Desirable to have slow growth rate.