Algorithm Analysis - Introduction

Outline

Introduction

RAM Model

Running time

Analysis - examples

One Problem - Multiple solutions

- Sorting Problem - Algorithms
  - Bubble Sort
  - Selection Sort
  - Insertion Sort
  - Merge, Quick, Heap ...
- Which one to choose?
- Basis for your choice?

Choosing a solution

- Easy to understand and code
- Resource usage - Processor time, Memory Space

Analyzing the algorithms

- Analyzing the algorithms means predicting the resources the algorithm uses
- What are the resources? - **Computational time and Memory for storage**
- **Why do we analyze algorithms?**
- Analyzing several algorithms for a particular problem results in the most efficient algorithm in terms of computational time/memory

Calculate Running Time

- Write program, Measure the actual execution time
  - Dependent on implementation - language, machine, compiler,...
  - Nature of input test cases
- Calculate before coding (*Apriori* Analysis)
  - Implementation independent
  - Theoretical-based on a machine model

Analyzing the algorithms

- A bench mark / model of the implementation technology and the resources of that technology and their costs
- We assume a generic one processor **Random Access Machine** (RAM) model of computation
  - We use RAM as an implementation technology
  - Our algorithms will be implemented as computer programs
  - Instructions are executed one after another, with no concurrent operations

RAM model

As it will be tedious to define each and every operation of RAM and its costs, we assume a realistic RAM

- RAM contains instructions commonly found in real computers such as:
  - **Arithmetic:** eg: add, subtract, multiply, divide, remainder, floor, ceil
  - **Data movement:** eg: load, store, copy
  - **Control:** conditional & unconditional branch, subroutine call and return

Random-Access Machine (RAM) model

- Single Processor Machine model
- Instructions for arithmetic, data movement, transfer of control - each taking a constant amount of time
- Instructions executed one after the other, no concurrent operations

Running time of Instructions:

...

$a = 0$

.....

$b = a$

....

$z\ = x + y$

....

return x

- Each statement is translated to a set of primitive operations or steps
- Running time depends on the number of primitive operations

Primitive Operations

Pseudocode statement: $z = x + y$

Translated code (for a hypothetical machine)

```
load r1, x  // loads contents of memory location x to register r1
load r2, y  // loads contents of memory location y to register r2
add  r1, r2  // adds contents of r1 and r2, stores result in r1
store z, r1  // // moves data in r1 to memory location z
```

- z = x + y required 4 machine instructions
- Number of steps different for different types of statements

Running Time of an Algorithm

- Running time on a particular input is the number of primitive operations or steps executed
- A constant amount of time for each line in the pseudocode
- The i^{th} line takes time c_i where c_i is a constant

Example

sample(a, b)	Cost	times
1. c = a − b	c1	1
2. temp = a	c2	1
3. a = b	c3	1
4. b = temp	c4	1
5. return c	c5	1

Running Time = c1 + c2 + c3 + c4 + c5

Example

Array-Sum(A)

	Cost	times
1. sum = 0	c1	1
2. **for** i = 1 **to** A. length	c2	n + 1
3. sum = sum + A[i]	c3	n
4. return sum	c4	1

Running Time = ???

Example

Array-Sum(A)	Cost	times
1. sum = 0	c1	1
2. **for** i = 1 **to** A. length	c2	n + 1
3. sum = sum + A[i]	c3	n
4. return sum	c4	1

Running Time = *c1 + c2(n + 1) + c3n + c4 // A. length is n*

= (*c2* + *c3*)*n* + (*c1*+*c2*+*c4*)

= *an* + *b*, linear

Calculate Running Times of the following algorithms....

Linear-Search(A, key)

```
for  i  = 1 to A. length
        if A[i] == key
                return i
return −1
```

Calculate Running Times of the following algorithms....

Count-Occurrence(A, key)

1. count = 0
2. **for** i = 1 **to** A. length
3. if A[i] == key
4. count = count + 1
5. return count

Calculate Running Times of the following algorithms....

```
Count-Occurrence(A, key )
count = 0
for i = 1 to A. length
    if A[i] == key
        count = count + 1
return count
```

$$Running\ time = c_1 + c_2(n + 1) + c_3 n + c_4 n + c_5$$

- Running time on an input of *n* items
- $T(n)$, running time as a function of input size, n
- $T(n) = an + b$, a linear function of n

Search-Multiple(A, B)

```
count = 0
for  i  = 1 to B. length
        key  = B[i ]
        for  j  = 1 to A. length
                if A[i] == key
                        count  = count + 1
return count
```