CE2001/ CZ2001: Algorithms

Analysis of Algorithms

Dr. Loke Yuan Ren

Learning Objectives

At the end of this lecture, students should be able to:

- Explain what is algorithm analysis
- Measure the resources used
- Analyse the time and space complexity
- Analyse basic program construct
- Perform best-case, worst-case, and average-case analysis

Algorithm Analysis

Algorithm Analysis

- We analyse algorithms to quantify their resource consumption.
- **Asymptotic Algorithm Analysis:** study of the resources used by an algorithm when the problem becomes larger and larger without bound.

> **ASYMPTOTICS:** Study of functions of a parameter N, as N becomes larger and larger without bound.

- We need comparison criteria or measures.
  - **Efficiency:** time and space.

Measurement of Used Resources

- **Empirical comparison** **(Run programmes)**
  - Difficult to do "fairly”.
  - Execution time is not the right measure of time efficiency.
  - Time consuming.

Time and Space Complexities

- Analyze efficiency of an algorithm in two aspects
  - Time
  - Space

- Time complexity: the amount of time used by an algorithm
- Space complexity: the amount of memory units used by an algorithm

Time and Space Complexity

- **Amount of time used** **(Time complexity)**
  - We want to count the number of primitive operations.
  - First, determine the major parameters that affect the problem (problem size).
  - Then derive an equation that relates the parameters to the number of primitive operations that the algorithm does.
  - Related to algorithmic aspects.

Time and Space Complexity

- **Amount of space used** **(Space complexity)**
  - Count the number of storage units required.
  - First, determine the major parameters that affect the problem (problem size).
  - Then derive an equation that relates the parameters to the number of storage units that the algorithm uses
  - Related to data structures.

Time and Space Complexity

- **Problem size**
  - Problem size depends on the problem being studied. e.g. the no. of items to be sorted.
  - Usually the number of primitive operations done by an algorithm is a function of the problem size.

Time and Space Complexity

- **Primitive operations**

  A basic step that can be performed in constant time.

  Examples:
  - Declaration (e.g., int x)
  - Assignment (e.g., x=1)
  - Arithmetic operations (+, -, *, /,%)
  - Comparisons (==, !=, <, >, <=, >=)

These primitive operations take constant time to perform

Basically they are not related to the problem size

changing the input(s) does not affect the computation time

Time Complexity or Time Efficiency

1. Count the number of primitive operations in the algorithm

   i. Repetition Structure: for-loop, while-loop
   ii. Selection Structure: if/else statement, switch-case statement
   iii. Recursive functions

2. Express it in term of problem size

Repetition Structure: ‘while’ Loop and Nested ‘for’ Loops

Repetition Structure: for-loop, while-loop

```
j ← 1                      ········▶ c0
factorial ← 1              ········▶ c1
while j ≤ n do
   factorial ← factorial * j ······▶ c2
   j ← j + 1               ········▶ c3
```

n iterations ➡ $n(c_2+c_3)$

$f(n) = c_0+c_1+n(c_2+c_3)$

The function increases linearly with n (problem size)

i. Repetition Structure: for-loop, while-loop

```
for j ← 1, m do
    for k ← 1, n do
        sum ← sum + M[ j ][ k ] ····▶ c1
```

n iterations $n(c_1)$

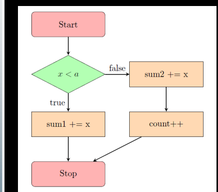
m iterations $m(n(c_1))$

The function increases quadratically with n if m==n

*Some constant time operations are ignored here.

Selection Structure:
if/else statement
switch-case statement

Selection Structure: if/else statement



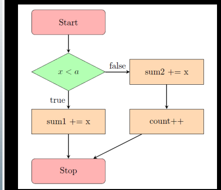

```
if(x<a)
    sum1 += x;
else {
    sum2 += x;
    count ++;
    }
```

When x < a, only one primitive operation is executed
When x ≥ a, two primitive operations are executed

How do we analyze the time complexity?

1. Best-case analysis
2. Worst-case analysis
3. Average-case analysis

Selection Structure: if/else statement



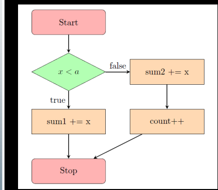

```
if(x<a)
      sum1 += x;
else {
     sum2 += x;
     count ++;
     }
```

When x < a, only one primitive operation is executed
When x ≥ a, two primitive operations are executed

How do we analyze the time complexity?

1. Best-case analysis c_1
2. Worst-case analysis
3. Average-case analysis

Selection Structure: if/else statement



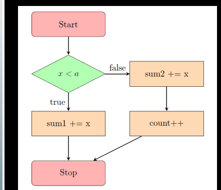

```
if(x<a)
      sum1 += x;
else {
     sum2 += x;
     count ++;
     }
```

When x < a, only one primitive operation is executed
When x ≥ a, two primitive operations are executed

How do we analyze the time complexity?

1. Best-case analysis
2. Worst-case analysis c_2
3. Average-case analysis

Selection Structure: if/else statement




```
if(x<a)
      sum1 += x;
else {
     sum2 += x;
     count ++;
     }
```

When x < a, only one primitive operation is executed
When x ≥ a, two primitive operations are executed

How do we analyze the time complexity?

1. Best-case analysis c_1
2. Worst-case analysis c_2
3. Average-case analysis

$$p(x < a)\, c_1 + p(x \geq a) c_2$$
$$= p(x < a)\, c_1 + (1 - p(x < a)) c_2$$

NANYANG TECHNOLOGICAL UNIVERSITY

Selection Structure: switch-case statement

Selection Structure: switch-case statement

```
switch(choice){
    case 1: compute the summation; break;      ····▸ 5n
    case 2: search BST; break;                 ····▸ 6log2 n
    case 3: print BST; break;                  ····▸ 3n
    case 4: search for the minimum; break;     ····▸ 4 log2 n
}
```

Time Complexity

1. Best-case analysis ····▸ $C + 4\log_2 n$
2. Worst-case analysis ····▸ $C + 5n$
3. Average-case analysis ····▸ $C + \sum_{i=1}^{m} p(i)T_i$

Different Cases of Analysis

Different Cases of Analysis

- **Best-case analysis** gives us the minimum no. of primitive operations performed by the algorithm on any input of size n ➡ best-case time complexity

NOT when n=1

- **Worst-case analysis** gives us the maximum no. of primitive operations performed by the algorithm on any input of size n ➡ worst-case time complexity

NOT when n=infinity

- **Average analysis** gives us the average no. of primitive operations performed by the algorithm on all inputs of size n ➡ average time complexity

Different Cases of Analysis

```
pt=head;                        ····▸ c1
while (pt.key != a){
    pt = pt.next;
    if(pt == NULL) break;        } c2  (n-1) iterations
}
```

1. Best-case analysis: c_1 when **a** is the first item in the list
2. Worst-case analysis: $c_2 \cdot (n-1) + c_1$ when **a** is the last item in the list
3. Average-case analysis
   - Assumed that every item in the list has an equal probability as a search key

$$\frac{1}{n}\sum_{i=1}^{n}(c_1 + c_2(i-1)) = \frac{1}{n}[nc_1 + c_2\sum_{i=1}^{n}(i-1)$$

$$= c_1 + \frac{c_2}{n}\cdot\frac{n}{2}(0 + (n-1))$$

$$= c_1 + \frac{c_2(n-1)}{2}$$

Time Complexity Function

Time Complexity Function

- **Time complexity function** of an algorithm is an expression of the number of operations performed by the algorithm.
- Express time complexity (of an algorithm) in terms of problem size (as a function of n).
- For asymptotic analysis, the exact value of the constant terms and multipliers in time complexity function are not important.

Summary

Summary

- What is algorithm analysis?
- How to measure the resources used?
- Concepts of Time and Space Complexity
- Analysing basic program construct
- Best-case, worst-case and average-case analysis