CS350 ABET Objective 1 & 8

Analyze the running time and space complexity of algorithms.

Compare the rates of growth of functions.

Textbook Section 2.1, ~5%.

We outline a general framework for analyzing the efficiency of algorithms. Time efficiency indicates how fast an algorithm runs.

Most algorithms are designed to work with inputs of arbitrary size. The efficiency of an algorithm is stated as a function relating the input size to the number of steps executed (or storage locations occupied) to completion.

Typical efficiency functions are compared and worst-, best-, and average-cases are defined.

Efficiency of an Algorithm

Concern about two resources:

1. Running time
2. Memory space

time efficiency/complexity
= time required to run
space efficiency/complexity
= space required to run
(space usually less crucial)

Requires resources depend on inputs.
Usually a larger input takes more resources.

E.g.: multiply two numbers.
E.g.: sorting a sequence.

Input's size

What do we measure or count?
It depends on the problem.
Must know/understand algorithm.

E.g.: multiply two numbers.
Number of digits.
Digits are multiplied with each other.
More digits. more multiplications.

E.g.: sorting a sequence.
Length (number of elements) of the sequence.
Elements are moved around.
More elements, more moves.

E.g.: evaluate a polynomial.
Degree of the polynomial.
Multiply and add variables and/or coefficients.
Higher degree, more multiplications and additions.

Unit for Measuring Time

Some obvious choice:

1. Clock actual running time
2. Count all the operations executed
3. Count one particular basic operation

Last option is the standard choice (no time!)

Let:

- n = the input size
- C(n) = the count of basic operation
- cop = the execution time of basic op
- T(n) = running time

$$T(n) \approx c_{op} C(n)$$

Time of Basic Op

Typically c_{op} is hard to guess and ignored.

What can you say if

1. double machine speed?
2. double input size?

1. Half the time.

2. Assume $C(n) = 3.3n^2$ (arbitrary):

$$\frac{T(2\mathrm{n})}{T(n)} \approx \frac{c_{op}C(2\mathrm{n})}{c_{op}C(n)} = \frac{3.3(2\mathrm{n})^2}{3.3n^2} = 4$$

In both cases c_{op} does not matter.

Orders of growth

n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10^1	$3.3 \cdot 10^1$	10^2	10^3	10^3	$3.6 \cdot 10^6$
10^2	6.6	10^2	$6.6 \cdot 10^2$	10^4	10^6	$1.3 \cdot 10^{30}$	$9.3 \cdot 10^{157}$
10^3	10	10^3	$1.0 \cdot 10^4$	10^6	10^9		
10^4	13	10^4	$1.3 \cdot 10^5$	10^8	10^{12}		
10^5	17	10^5	$1.7 \cdot 10^6$	10^{10}	10^{15}		
10^6	20	10^6	$2.0 \cdot 10^7$	10^{12}	10^{18}		

Worst, Best, Average Case

ALGORITHM *SequentialSearch*$(A[0..n-1], K)$

```
//Searches for a given value in a given array by sequential search
//Input: An array A[0..n − 1] and a search key K
//Output: The index of the first element in A that matches K
//           or −1 if there are no matching elements
i ← 0
while i < n and A[i] ≠ K do
    i ← i + 1
if i < n return i
else return −1
```

Basic operation: key comparison

Worst:	n	Key is not in array.
Best:	1	Key is in A[0].

Average: assume key found with probabilty p, and every index is equally likely:

$$C_{aver}(n) = \frac{p(n+1)}{2} + n(1-p)$$

References

Textbook Section 2.1

Web:

http://en.wikipedia.org/wiki/Analysis_of_algorithms