CSE408
Measuring of input size & running time

Lecture #3

Analysis of algorithms

- Issues:
  - correctness
  - time efficiency
  - space efficiency
  - optimality

- Approaches:
  - theoretical analysis
  - empirical analysis

Theoretical analysis of time efficiency

Time efficiency is analyzed by determining the number of repetitions of the *basic operation* as a function of *input size*

- *Basic operation*: the operation that contributes most towards the running time of the algorithm

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

Empirical analysis of time efficiency

- Select a specific (typical) sample of inputs

- Use physical unit of time (e.g., milliseconds)

  or

  Count actual number of basic operation's executions

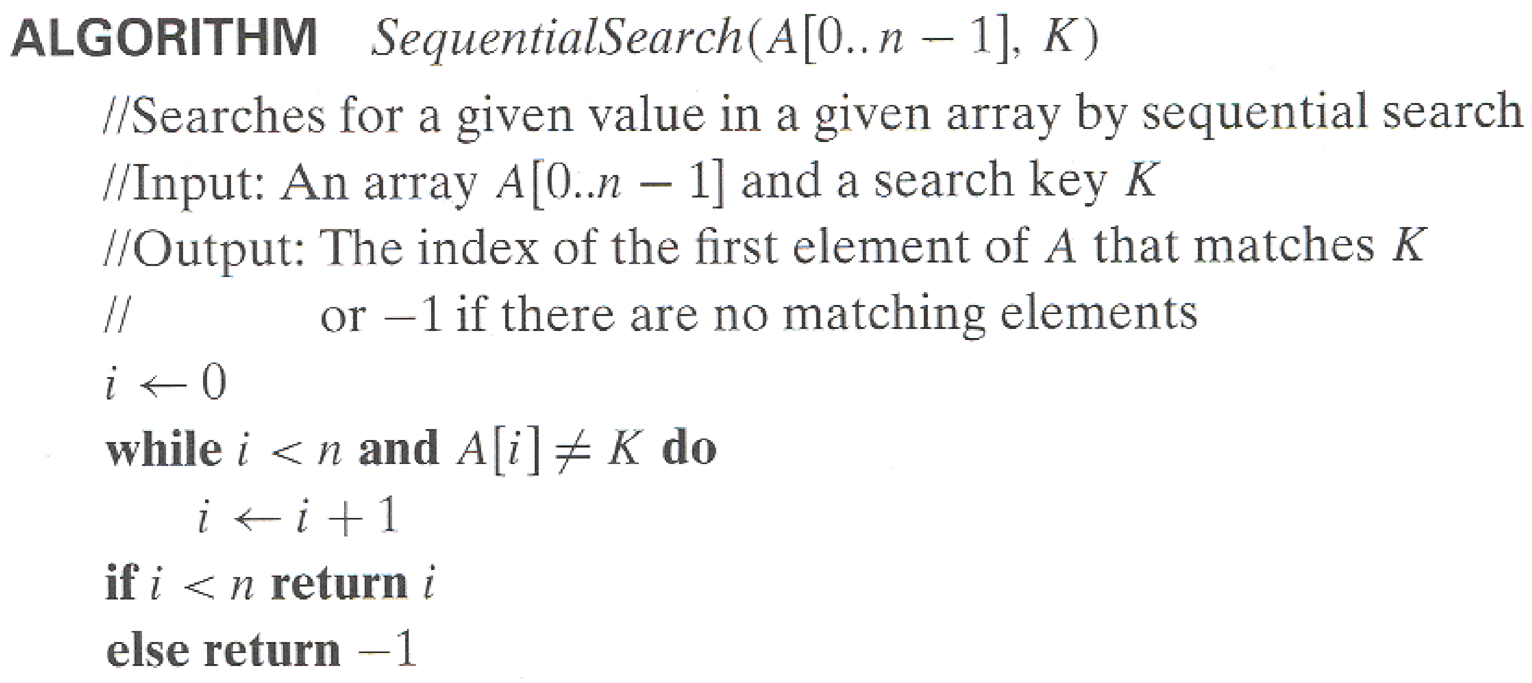
- Analyze the empirical data

Best-case, average-case, worst-case

For some algorithms efficiency depends on form of input:

- Worst case: $C_{\text{worst}}(n)$ – maximum over inputs of size n
- Best case: $C_{\text{best}}(n)$ – minimum over inputs of size n
- Average case: $C_{\text{avg}}(n)$ – “average” over inputs of size n
  - Number of times the basic operation will be executed on typical input
  - NOT the average of worst and best case
  - Expected number of basic operations considered as a random variable under some assumption about the probability distribution of all possible inputs

Example: Sequential search



```
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 of 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
```

- Worst case

- Best case

- Average case

Example

Let's consider again sequential search. The standard assumptions are that (a) the probability of a successful search is equal to *p* *($0 \leq p \leq 1$) and (b) the* probability of the first match occurring in the *ith position of the list is the same* for every *i*.

we can find the average number of key comparisons *Cavg(n) as follows. In the case of a successful search, the* probability of the first match occurring in the *ith position of the list is p/n for* every *i, and the number of comparisons made by the algorithm in such a situation* is obviously *i. In the case of an unsuccessful search, the number of comparisons* will be *n with the probability of such a search being (1− p). Therefore,*

Types of formulas for basic operation's count

- Exact formula

  e.g., $C(n) = n(n-1)/2$

- Formula indicating order of growth with specific multiplicative constant

  e.g., $C(n) \approx 0.5\, n^2$

- Formula indicating order of growth with unknown multiplicative constant

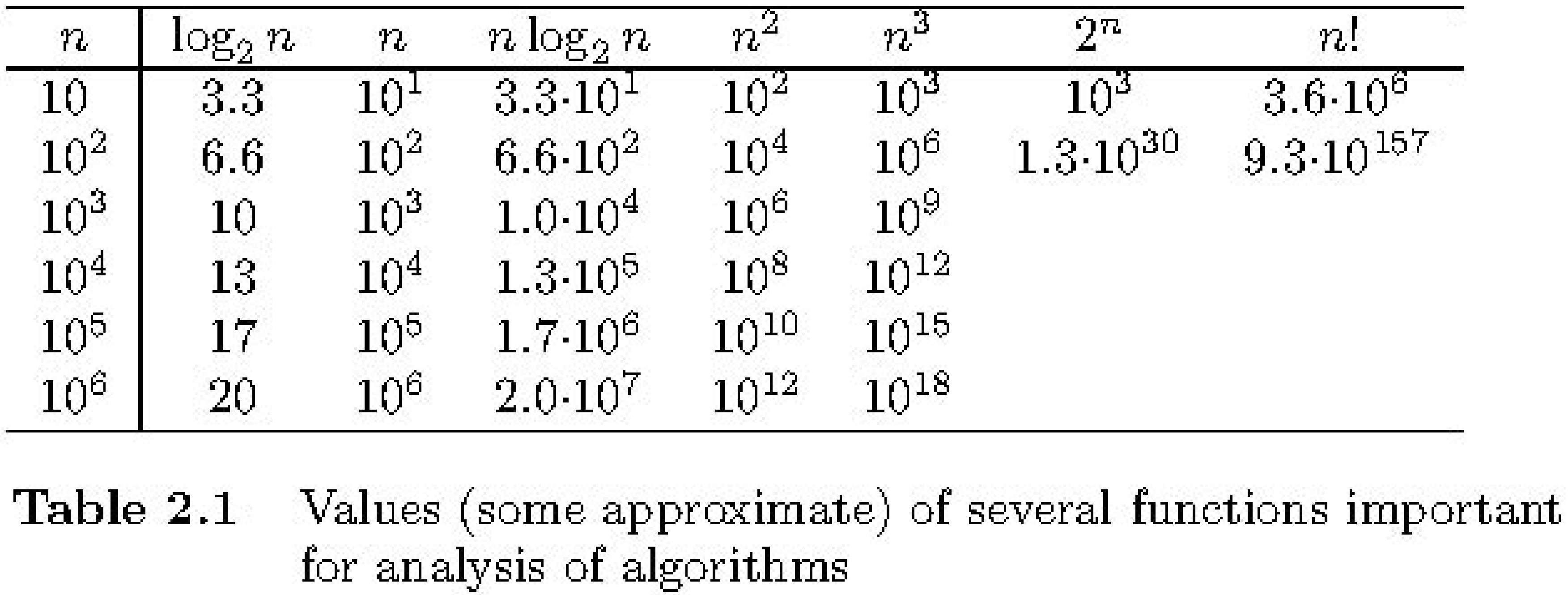
  e.g., $C(n) \approx cn^2$

Order of growth

- Most important: Order of growth within a constant multiple as $n \longrightarrow \infty$

- Example:
  - How much faster will algorithm run on computer that is twice as fast?
  - How much longer does it take to solve problem of double input size?

Values of some important functions as *n*



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}		

Table 2.1 Values (some approximate) of several functions important for analysis of algorithms

Here is an important application. Let c_{op} be the execution time of an algorithm's basic operation on a particular computer, and let $C(n)$ be the number of times this operation needs to be executed for this algorithm. Then we can estimate

the running time $T(n)$ of a program implementing this algorithm on that computer by the formula

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

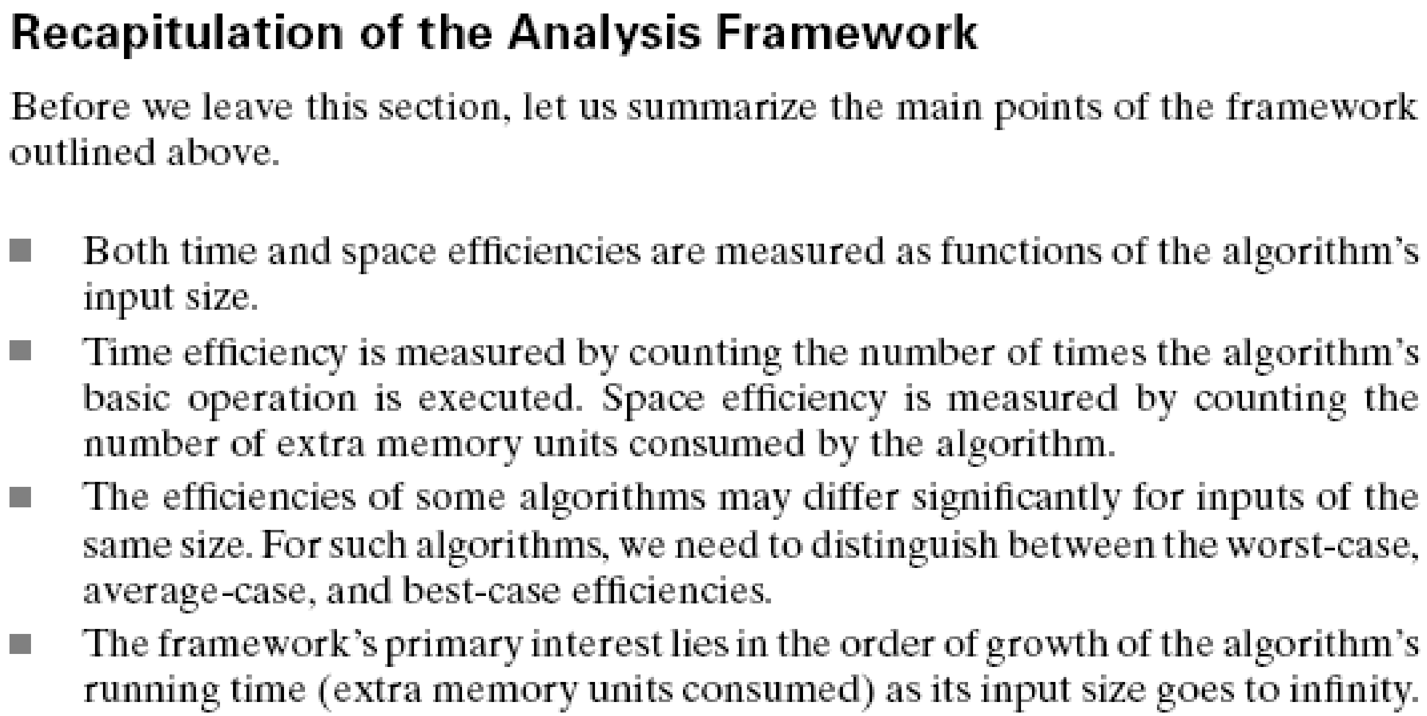
$C(n) = \frac{1}{2}n(n-1)$, how much longer will the algorithm run if we double its input size? The answer is about four times longer. Indeed, for all but very small values of n,

$$C(n) = \frac{1}{2}n(n-1) = \frac{1}{2}n^2 - \frac{1}{2}n \approx \frac{1}{2}n^2$$

and therefore

$$\frac{T(2n)}{T(n)} \approx \frac{c_{op}C(2n)}{c_{op}C(n)} \approx \frac{\frac{1}{2}(2n)^2}{\frac{1}{2}n^2} = 4.$$

Conclusion



Recapitulation of the Analysis Framework

Before we leave this section, let us summarize the main points of the framework outlined above.

- Both time and space efficiencies are measured as functions of the algorithm's input size.
- Time efficiency is measured by counting the number of times the algorithm's basic operation is executed. Space efficiency is measured by counting the number of extra memory units consumed by the algorithm.
- The efficiencies of some algorithms may differ significantly for inputs of the same size. For such algorithms, we need to distinguish between the worst-case, average-case, and best-case efficiencies.
- The framework's primary interest lies in the order of growth of the algorithm's running time (extra memory units consumed) as its input size goes to infinity.

- The efficiency analysis framework concentrates on the order of growth of an algorithm's basic operation count as the principal indicator of the algorithm's
- To compare and rank such orders of growth, computer scientists use three notations:*(big oh), (big omega), and (big theta)*efficiency

The efficiency analysis framework concentrates on the order of growth of an algorithm’s basic operation count as the principal indicator of the algorithm’s

To compare and rank such orders of growth, computer scientists use three notations:*(big oh), (big omega), and (big theta)*efficiency

Thank You !!!