Data Structures

Lecture: Asymptotic Notations & Complexity Analysis

By

Om Suthar

Asst. Professor,

Lovely Professional University, Punjab

Contents

- Basic Terminology
- Complexity of Algorithm
- Asymptotic Notations
- Review Questions

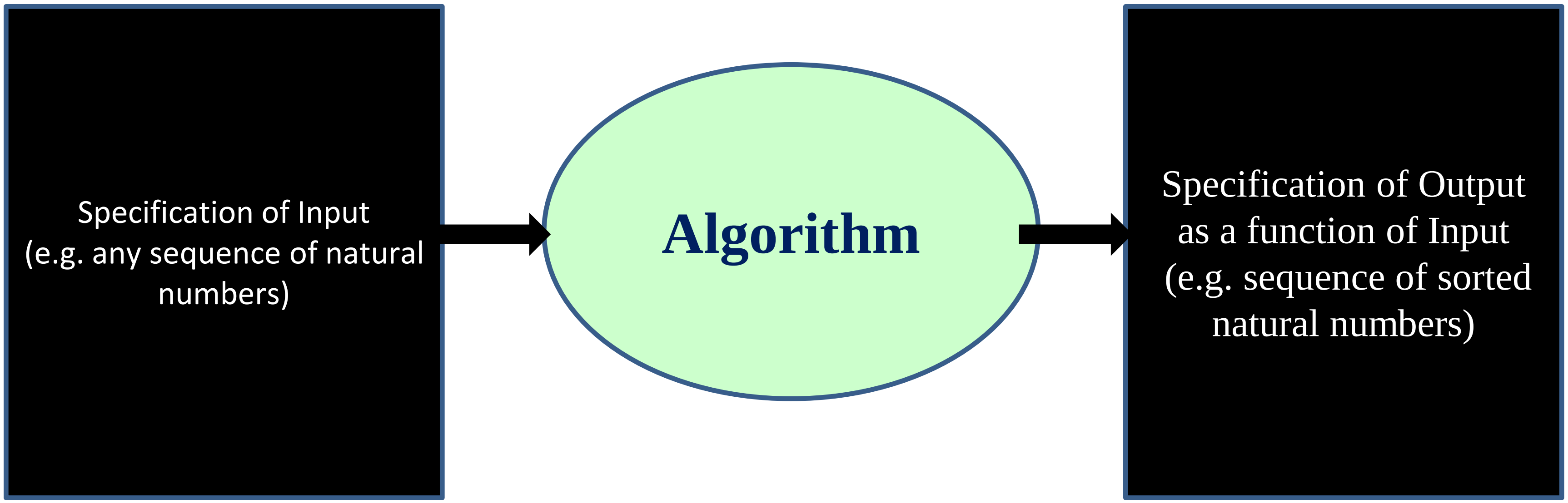
Basic Terminology

- Algorithm: is a finite step by step list of well-defined instructions for solving a particular problem.

- Complexity of Algorithm: is a function which gives running time and/or space requirement in terms of the input size.

- Time and Space are two major measures of efficiency of an algorithm.

Algorithm


Specification of Input
(e.g. any sequence of natural numbers)
Algorithm
Specification of Output as a function of Input
(e.g. sequence of sorted natural numbers)

Characteristics of Good Algorithm

- Efficient
  - Running Time
  - Space used

- Efficiency as a function of input size
  - Size of Input
  - Number of Data elements

Time-Space Tradeoff

- By increasing the amount of space for storing the data, one may be able to reduce the time needed for processing the data, or vice versa.

Complexity of Algorithm

- Time and Space used by the algorithm are two main measures for efficiency of any algorithm *M*.

- Time is measured by counting the number of key operations.

- Space is measured by counting the maximum of memory needed by the algorithm.

- Complexity of Algorithm is a function ***f(n)*** which gives running time and/or space requirement of algorithm M in terms of the size n of the input data.

- Worst Case: The maximum value of ***f(n)*** for any possible input.

- Average Case: The expected or average value of ***f(n)***.

- Best Case: Minimum possible value of ***f(n).***

Analysis of Insertion Sort Algorithm

	cost	times
for j←2 to n do	c1	n
key ←A[j]	c2	$n-1$
i ←j-1	c3	$n-1$
while i>0 and A[i] > key	c4	$\sum_{j=2}^{n} tj$
do A[i+1] ←A[i]	c5	$\sum_{j=2}^{n} (tj-1)$
i--	c6	$\sum_{j=2}^{n} (tj-1)$
A[i+1] ← key	c7	$n-1$

Total Time = $n(c1 + c2 + c3 + c7) + \sum_{j=2}^{n} tj\,(c4 + c5 + c6)$

$- (c2 + c3 + c5 + c6 + c7)$

Analysis of Insertion Sort

Total Time = n(c1 + c2 + c3 + c7) + $\sum_{j=2}^{n} tj\ (c4 + c5 + c6)$

- (c2 + c3 + c5 + c6 + c7)

- Best Case: Elements are already sorted, tj=1

  running time = f(n)

- Worst Case: Elements are sorted in reverse order, tj=j

  running time = f(n^2)

- Average Case: tj= j/2

  running time = f(n^2)

Rate of Growth

- The rate of growth of some standard functions *g(n)* is:

$$\log_2 n < n < n\log_2 n < n^2 < n^3 < 2^n$$

n \ $g(n)$	$\log n$	n	$n \log n$	n^2	n^3	2^n
5	3	5	15	25	125	32
10	4	10	40	100	10^3	10^3
100	7	100	700	10^4	10^6	10^{30}
1000	10	10^3	10^4	10^6	10^9	10^{300}

Asymptotic Notations

- Goal: to simplify analysis of running time .

- Useful to identify how the running time of an algorithm increases with the size of the input in the limit.

- Asymptotic is a line that approaches a curve but never touches.

Asymptotic Notations

Special Classes of Algorithms

- Logarithmic: $O(\log n)$
- Linear: $O(n)$
- Quadratic: $O(n^2)$
- Polynomial: $O(n^k)$, $k >= 1$
- Exponential: $O(a^n)$, $a > 1$

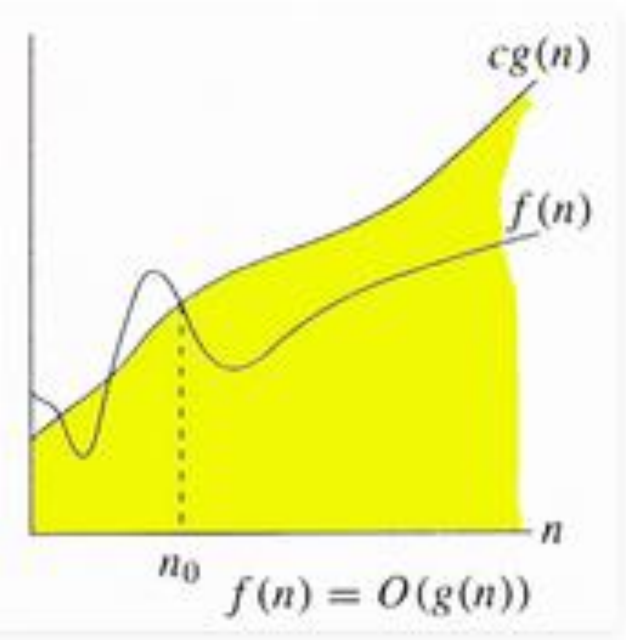
Big-Oh (O) Notation

- Asymptotic upper bound

- $f(n) = O(g(n))$, if there exists constants c and n_0 such that,

- $f(n) <= c\, g(n)$ for $n >= n_0$

- $f(n)$ *and* $g(n)$ are functions over non-negative integers.
- Used for Worst-case analysis.



Big-Oh *(O)* Notation

- Simple Rule:

Drop lower order terms and constant factors.

Example:

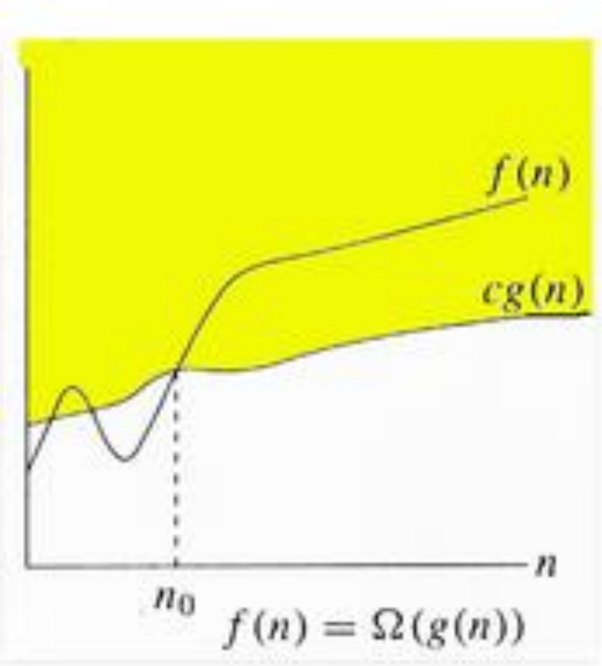
- $50n \log n$ is $O(n \log n)$
- $8n^2 \log n + 5 n^2 + n$ is $O(n^2 \log n)$

Big-Omega *(Ω)* Notation

- Asymptotic lower bound

- $f(n) = \Omega(g(n))$, if there exists constants c and n_0 such that,
  $c\,g(n) <= f(n)$ for n >= n_0

- Used to describe Best-case running time.



Big-Theta (Θ) Notation

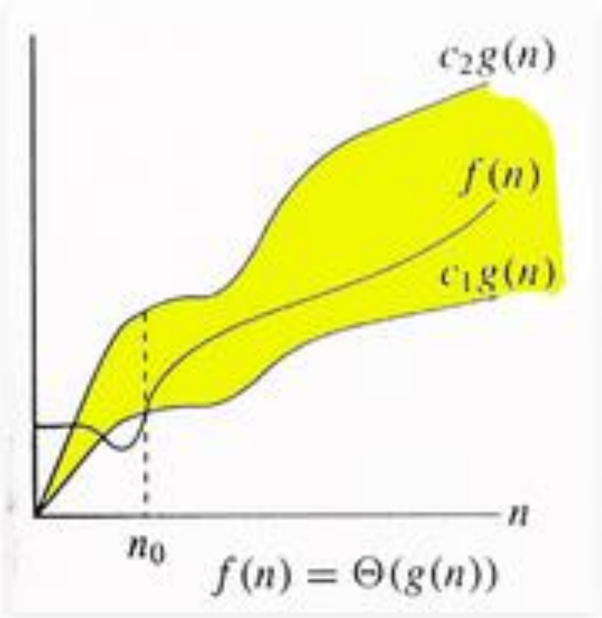
- Asymptotic tight bound

- $f(n) = \Theta(g(n))$, if there exists constants c_1, c_2 and n_0 such that,

- $c_1\, g(n) <= f(n) <= c_2\, g(n)$ for n $>= n_0$

- $f(n) = \Theta(g(n))$, iff f(n) = O(g(n)) and

  f(n) = Ω (g(n))



Little-Oh (o) Notation

- Non-tight analogue of Big-Oh.

- $f(n) = o\,(g(n))$, if for every c, there exists n_0 such that,

$$f(n) < c\, g(n) \text{ for n} >= n_0$$

- Used for comparisons of running times.

Analysis of Algorithms

- // Here c is a constant

```
for (int i = 1; i <= c; i++)
    {
    // some O(1) expressions
    }
```

- **O(1):** Time complexity of a function (or set of statements) is considered as O(1) if it doesn’t contain loop, recursion and call to any other function.

Analysis of Algorithms

- ```
  for (int i = 1; i <= n; i += c)
      { // some O(1) expressions }
  ```

- ```
  for (int i = n; i > 0; i -= c)
      { // some O(1) expressions }
  ```

- **O(n):** Time Complexity of a loop is considered as O(n) if the loop variables is incremented / decremented by a constant amount.

Analysis of Algorithms

- ```
  for (int i = 1; i <=n; i += c)
    {
     for (int j = 1; j <=n; j += c)
          { // some O(1) expressions
          }
     }
  ```
- ```
   for (int i = n; i > 0; i -= c)
    {
     for (int j = i+1; j <=n; j += c)
           { // some O(1) expressions
     }     }
  ```
- **O(n^2)**: Time complexity of nested loops is equal to the number of times the innermost statement is executed.

Analysis of Algorithms

- for (int i = 1; i <=n; i *= c)

  { // some O(1) expressions }

- for (int i = n; i > 0; i /= c)

  { // some O(1) expressions }

- **O(Logn)** Time Complexity of a loop is considered as O(Logn) if the loop variables is divided / multiplied by a constant amount.

Analysis of Algorithms

- for (int i = 2; i <=n; i = pow(i, c))

  { // some O(1) expressions }

- //Here fun is sqrt or cuberoot or any other constant root

for (int i = n; i > 0; i = fun(i))

  { // some O(1) expressions }

- **O(LogLogn)** Time Complexity of a loop is considered as O(LogLogn) if the loop variables is reduced / increased exponentially by a constant amount.

Analysis of Algorithms

- for (int i = 2; i*i <=n; i++))
  { // some O(1) expressions }

- **O(√n)** Time Complexity.

Questions

Review Questions

- When an algorithm is said to be better than the other?
- Can an algorithm have different running times on different machines?
- How the algorithm’ running time is dependent on machines on which it is executed?

Review Questions

Find out the complexity:

```
function ()
   {
N  if (condition)
         {
            for (i=0; i<n; i++) { // simple statements}
         }
   else
      {
         for (j=1; j<n; j++)
            for (k=n; k>0; k--) {// simple statement}
      }
   }
```