Lecture 3.1 & 3.2

- Dr. Anurag Sharma

Complexity Analysis of Algorithms – Big O order

Time complexity

- In computer science, the time complexity is the computational complexity that describes the amount of time it takes to run an algorithm.
- Time complexity is commonly estimated by counting the number of elementary operations performed by the algorithm, supposing that each elementary operation takes a fixed amount of time to perform.

Order

- Algorithms with time complexities such as n and $100n$ are called linear-time algorithms because their time complexities are linear in the input size n.
- Algorithms with time complexities such as n^2 and $0.01n^2$ are called quadratic-time algorithms because their time complexities are quadratic in the input size n.

An Intuitive Introduction to Order

- Functions such as $5n^2$ and $5n^2 + 100$ are called **pure quadratic functions** because they contain no linear term.
- Functions such as $0.1n^2 + n + 100$ is called a **complete quadratic function** because it contains a linear term.
- So what is the time complexity of these functions?

Cont.

- Throw away low-order terms when classifying complexity functions. Why?
- For example, $0.1n^3 + 10n^2 + 5n + 25 \cong n^3$, a pure cubic function.
- When an algorithm's time complexity is polynomial of order 2, it is called a quadratic-time algorithm.
- Examples of complexity categories:
  - $log(n), n, n\,log(n), n^2, n^3, 2^n$ etc.

Exercise:

- Draw the following in Matlab and observe the behavior of the functions:

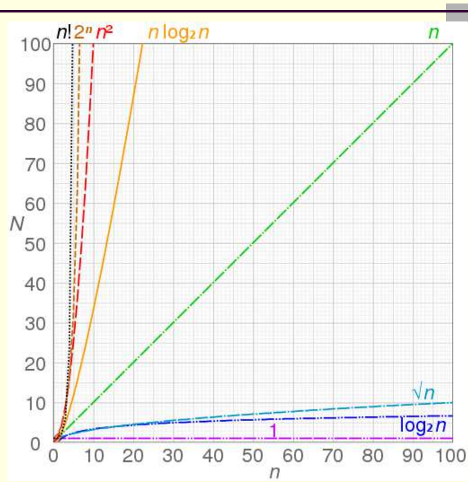
```
y1 = 0.1*n.^3+10*n.^2+ 5*n + 25;
y2 = n.^3;
y3 = n.^2;
y4 = n.^1;
y5 = n.^4;
```

- n = 0:1000

Big O Notation

- Big O notation, is used to describe upper bound of the time, and space usage of an algorithm.
  - In this notation n refers to the size of the input into the algorithm.
  - Order refers to the time or amount of time it takes the algorithm to finish executing n sized input.

Commonly used Big O order

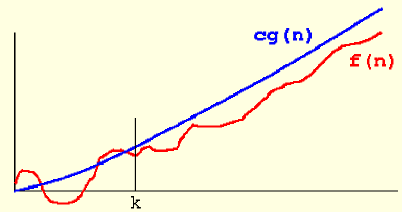


Big O Notation (cont.)

- big-O notation for a problem of size N:
  - a constant-time method is "order 1": O(1)
  - a linear-time method is "order N": O(N)
  - a quadratic-time method is "order N squared": $O(N^2)$
- Note that the big-O expressions do not have constants or low-order terms.
  - This is because, when N gets large enough, constants and low-order terms don't matter

Formal definition of Big O

- For a given complexity function $f(n)$, it big order $O(g(n))$ means there are positive constants c and k, such that:
  - $0 \leq f(n) \leq cg(n)$ for all $n \geq k$



Example

- What is the big O order of the following function?

$$n^2 + 3n + 4$$

$$n^2 + 3n + 4 \leq n^2 + 3n^2 + 4n^2 \text{ for } n > 0$$

$$\Rightarrow n^2 + 3n + 4 \leq 8n\text{^}2$$

Here $C = 8$ and $k = 1$ [using a simple approach)

- Answer: $O(n^2)$
- It is also possible to have $C = 2$ and $k = 10$, however, the final answer will remain same.

Asymptotic behavior

- Big O describes the **asymptotic behavior** of a function because it is concerned only with eventual behavior. We say that big O puts an asymptotic upper bound on a function.
- Similar notation is used to describe the least amount of a resource that an algorithm needs for some class of input. The lower bound of an algorithm is denoted by the symbol **Ω (omega)**. [this will not be covered ☺]

Realization of different orders

● Table 1.4 Execution times for algorithms with the given time complexities

n	$f(n) = \lg n$	$f(n) = n$	$f(n) = n \lg n$	$f(n) = n^2$	$f(n) = n^3$	$f(n) = 2^n$
10	0.003 μs*	0.01 μs	0.033 μs	0.10 μs	1.0 μs	1 μs
20	0.004 μs	0.02 μs	0.086 μs	0.40 μs	8.0 μs	1 ms†
30	0.005 μs	0.03 μs	0.147 μs	0.90 μs	27.0 μs	1 s
40	0.005 μs	0.04 μs	0.213 μs	1.60 μs	64.0 μs	18.3 min
50	0.006 μs	0.05 μs	0.282 μs	2.50 μs	125.0 μs	13 days
10^2	0.007 μs	0.10 μs	0.664 μs	10.00 μs	1.0 ms	4×10^{13} years
10^3	0.010 μs	1.00 μs	9.966 μs	1.00 ms	1.0 s	
10^4	0.013 μs	10.00 μs	130.000 μs	100.00 ms	16.7 min	
10^5	0.017 μs	0.10 ms	1.670 ms	10.00 s	11.6 days	
10^6	0.020 μs	1.00 ms	19.930 ms	16.70 min	31.7 years	
10^7	0.023 μs	0.01 s	2.660 s	1.16 days	31,709 years	
10^8	0.027 μs	0.10 s	2.660 s	115.70 days	3.17×10^7 years	
10^9	0.030 μs	1.00 s	29.900 s	31.70 years		

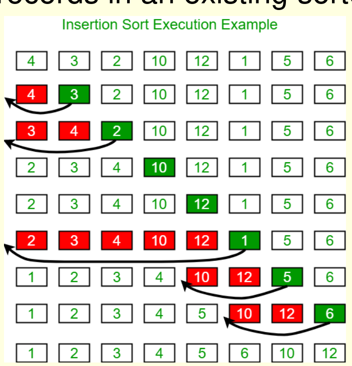
*1 μs = 10^{-6} second.
†1 ms = 10^{-3} second.

Writing efficient algorithm

- Good programming skills
- Use built-in algorithms which are well-tested and have known Big O order
- Good analytical ability and understanding of mathematics
- Knows strengths and weaknesses of data structures

Insertion sort

- An insertion sort algorithm is one that sorts by inserting records in an existing sorted array.



- Ref: https://www.geeksforgeeks.org/insertion-sort/

Code

```java
/*Function to sort array using insertion sort*/
void sort(int arr[])
{
    int n = arr.length;
    for (int i=1; i<n; ++i)
    {
        int key = arr[i];
        int j = i-1;

        /* Move elements of arr[0..i-1], that are
           greater than key, to one position ahead
           of their current position */
        while (j>=0 && arr[j] > key)
        {
            arr[j+1] = arr[j];
            j = j-1;
        }
        arr[j+1] = key;
    }
} //Ref: https://www.geeksforgeeks.org/insertion-sort
```

Cont.

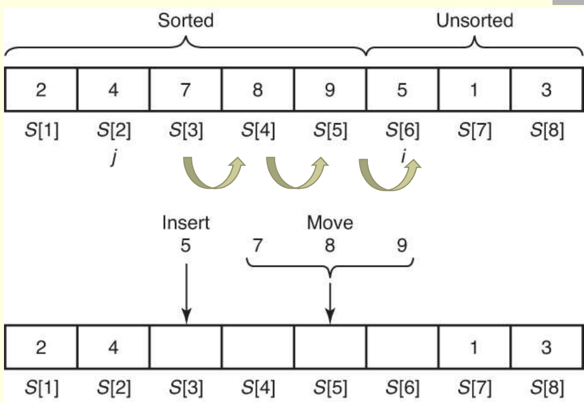


Figure 7.1: An example illustrating what Insertion Sort does when *i*=6 and *j*=2 (top). The array before inserting, and (bottom) the insertion gap.

Big O order?

- Worst case:

$$1 + 2 + \ldots. + n - 1 = \frac{n(n-1)}{2}$$

- The Big O order for the worst case is $O(n^2)$
- What is the best case?
- When array is already sorted then the order is $O(n)$

Choice of data structure

- Big O order for inserting an element?
- Using an array:
  - Worst case: $O(n)$
- Using a linked list:
  - Worst case: $O(1)$

- Big O order for retrieving an element?