الجامعة المصرية
للتعلم الإلكتروني الأهلية
EELU
THE EGYPTIAN E-LEARNING UNIVERSITY

DATA STRUCTURES

By

Dr. Yasser Abdelhamid

OUTLINE

- Analysis of algorithms
- Asymptotic notations
- Counting algorithm steps
- Insertion Sort

ANALYSIS OF ALGORITHMS

الجامعة المصرية
للتعلم الإلكتروني الأهلية

EELU

THE EGYPTIAN E-LEARNING UNIVERSITY

IMPORTANCE OF ALGORITHM ANALYSIS

- **It is extremely important to expect the resources required by an algorithm.**
  - Execution Time
  - Amount of Memory
  - Other resources (disk usage, communication ports, software resources, etc.)

ALGORITHM PERFORMANCE

- **Time Complexity** is a function describing the **amount of time** required to run an algorithm in terms of the size of the input.

- **Space Complexity** is a function describing a **mount of memory** to run an algorithm takes in terms of the size of the input.

COMPLEXITY ANALYSIS ASSUMPTIONS

- Analysis is independent of the configuration of the computer where it will be executed.

- Analysis is independent of the programming language that will be used.

- We are interested in cases where the input data (n) asymptotes to infinity (extremely big number of input elements).

ASYMPTOTIC NOTATIONS

ASYMPTOTIC NOTATIONS

- Three standard notations: O, Ω,θ.
- O, provides upper bound (Worst case) of the algorithm performance.
- Ω provides lower bound (Best case) of the algorithm performance.
- O, Ω,θ identify classes of functions
- Expected time (T) is a function of the number of input data elements (n)

WORST CASE

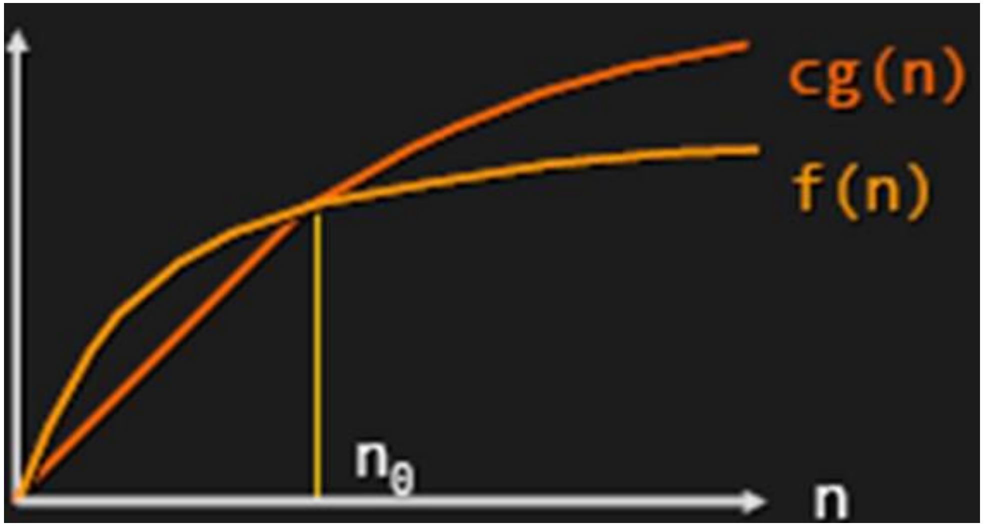
- The case that causes maximum number of steps to be executed.
- T(n) = upper bound on running time of an algorithm for an input of size n.
- Example: Search for number 8

2	3	5	4	1	7	6

O NOTATION ("BIG-O")

- f(n) has the upper order (worst case) O(g(n)) if there is a constant c which is greater than 0, and nother constant n_0 that is greater than or equal to zero and f(n) is always less than or equal to c.g(n) for all values of $n >= n_{0.}$
- f(n)=O(g(n)) if c > 0 and n_0 >= 0 such that

f(n) <= c.g(n) for all values of $n >= n_{0.}$



BEST CASE

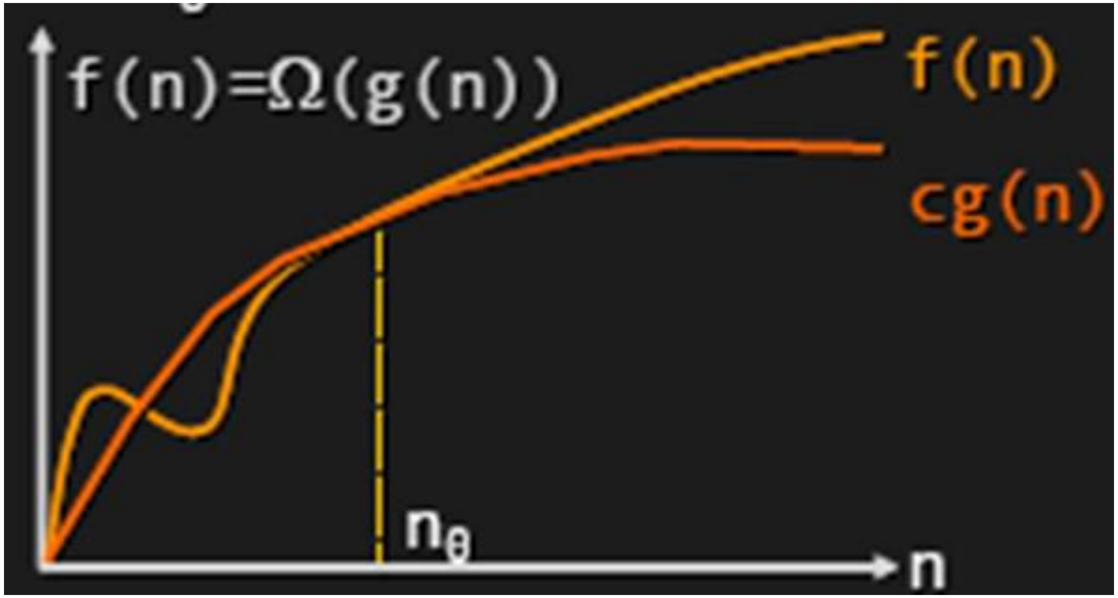
- The case that causes minimum number of steps to be executed.
- Example: Search for number 2

2	3	5	4	1	7	6

Ω NOTATION ("BIG-OMEGA")

- f(n) has the lower order (best case) Ω(g(n)) if there is a constant c which is greater than 0, and nother constant n_0 that is greater than or equal to zero, and f(n) is always greater than or equal to c.g(n) for all values of n greater than than or equal to n_0.
- f(n)= Ω(g(n)) if c > 0 and n_0 >= 0 such that

f(n) >= c.g(n) for all values of n >= n_0.

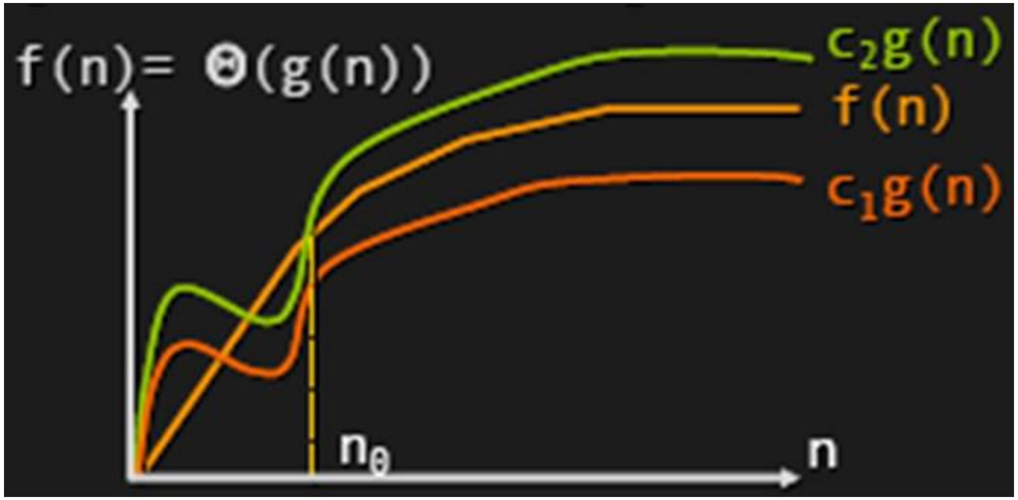


AVERAGE CASE

- We take all possible inputs and calculate computing time for all of the inputs
- T(n) =expected time of algorithm over all inputs of size n

θ NOTATION ("BIG-THETA")

- f(n) has the average order θ(g(n)) if there is a constant c_1 , c_2 which are greater than 0, and nother constant n_0 that is greater than or equal to zero, and f(n) is always greater than or equal to c_1.g(n) and less than or equal to c_2 .g(n) for all values of n greater than than or equal to n_0.
- f(n)= θ(g(n)) if c_1 , c_2 > 0 and n_0 >= 0 such that c_1 .g(n) <= f(n) <= c_2 .g(n) for all values of n >= n_0.



EXAMPLE

- Find T(n) for the following program.

$$
\begin{aligned}
&for(int\ i = 1; i \leq n; i++)\\
&\qquad print\ i;
\end{aligned}
$$

EXAMPLE

- How many times does sum++ run?

```
for(i=4; i<n;i++)
   for(j=0; j<=n; j++)
      sum++;
```

I	j	count
4	0,1,2,…,n	n-0+1 = n-1
5	0,1,2,…,n	n-0+1 = n-1
…	…	…
n-1	0,1,2,…,n	n-0+1 = n-1

- number of times $= (n-1 -4 +1) * (n-1)$
- $= (n-4)(n-1) = n(n-1)-4(n-1)$
- $= n^2 – 5n + 4$
- $\approx O(n^2)$

EXAMPLE

- How many times does sum++ run?

```
for(i=4; i<n;i++)
   for(j=0; j<=i; j++)
       sum++;
```

I	j	count
4	0,1,2,3,4	5
5	0,1,2,3,4,5	6
…	…	…
n-1	0…n-1	n

- **number of times** $= \sum_{i=5}^{n} i = \sum_{i=1}^{n} i - \sum_{i=1}^{4} i$
- $= \frac{n(n+1)}{2} - 10$
- $\approx \frac{1}{2} n^2$
- $\approx O(n^2)$

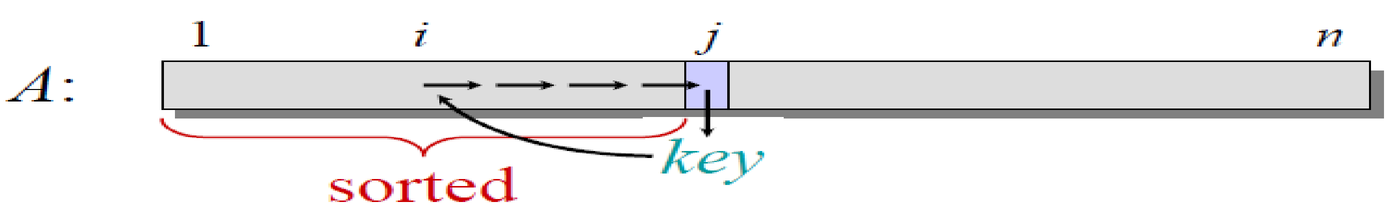
INSERTION SORT

- An efficient algorithm for sorting a list of small number of elements
- Works the way many people sort a hand of playing cards
- Start with an empty left hand and the cards are on the table
- Then remove one card at a time from the table and insert it into the correct position in the left hand.
- To find the correct position for a card, we compare it with each of the cards already in the hand, starting from right.

INSERTION SORT

"pseudocode"

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted sequence A[1 .. j − 1].
    i = j − 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```



EXAMPLE

8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6

2 4 8 9 3 6

2 3 4 8 9 6

2 3 4 6 8 9 *done*

ANALYSIS OF INSERTION SORT ALGORITHM

	INSERTION-SORT(A)	*cost*	*times*
1	**for** $j = 2$ **to** $A.length$	c_1	n
2	$key = A[j]$	c_2	$n-1$
3	// Insert $A[j]$ into the sorted sequence $A[1 \,.\,.\, j-1]$.	0	$n-1$
4	$i = j - 1$	c_4	$n-1$
5	**while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6	$A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7	$i = i - 1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
8	$A[i+1] = key$	c_8	$n-1$

- t_j denotes the number of times the while loop test in line 5 is executed for that value of j.
- Comments are not executable statements, and so they take no time.

ANALYSIS OF INSERTION SORT ALGORITHM

- **Running Time of Insertion Sort:**

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1).$$

- **In INSERTION-SORT, the best case occurs if the array is already sorted.**

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\ &= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8). \end{aligned}$$

THANK YOU

الجامعة المصرية
للتعلم الإلكتروني الأهلية

EELU

THE EGYPTIAN E-LEARNING UNIVERSITY