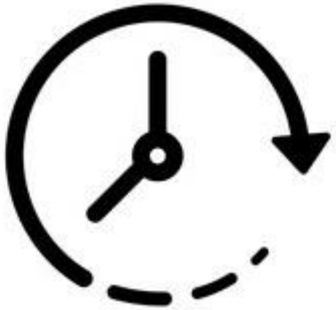
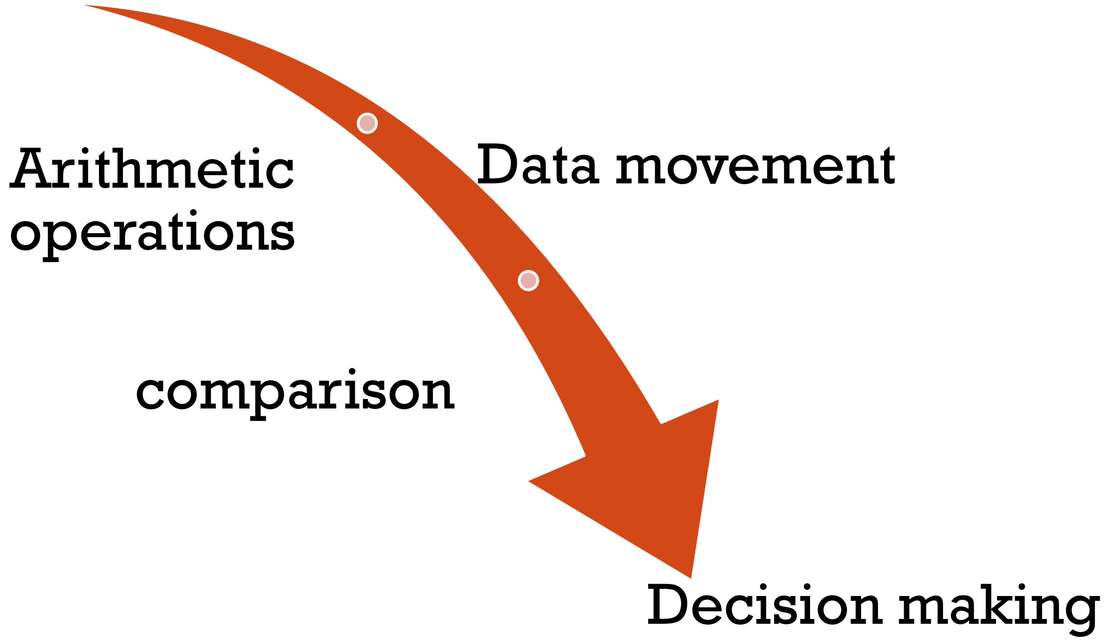
ANALYSIS & DESIGN OF ALGORITHMS

Dr. Sondos Fadl

RUNNING TIME



- The **running time** of an algorithm on a particular input is the **number of primitive operations** or "**steps**" executed before termination.



Approximate not exact !

PRIMITIVE OPERATIONS

- Ex: Find min value of an array?

```
Min(A, 1 , n)
  m=A(1)
  for i=2  to n
     if A(i) <m
          m=A(i)
```

EFFICIENCY VS. SPEED

Array size 10^6

- Ex: sorting numbers:

Ahmed's computer$=10^9$ operations /sec

Ahmed's Algorithm$=2n^2$ operations

$$\text{Time}=\frac{Algorithm\ complexity}{device\ speed}=\frac{2\times(10^6)^2}{10^9}=2000\ sec$$

Mohamed's computer$=10^7$ operations /sec

Mohamed's Algorithm$=50n\log n$ operations

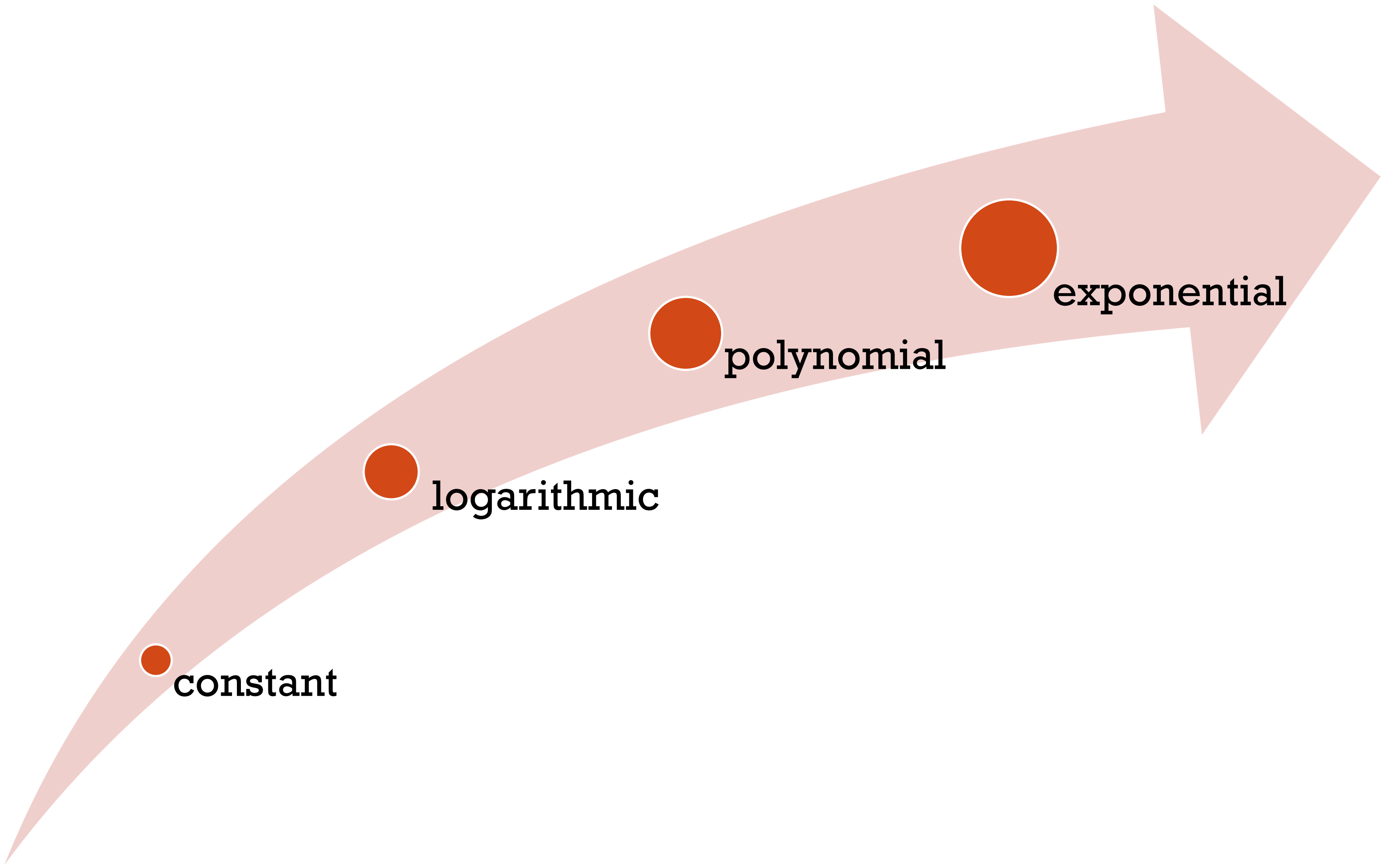
20 times better !!!

$$\text{Time}=\frac{Algorithm\ complexity}{device\ speed}=\frac{50\times10^6\log 10^6}{10^7}=100\ sec$$

TYPICAL RUNNING TIME

- 1 (constant)
- $\log n$ (logarithmic)
- n^k (polynomial)
  - n linear
  - n^2 quadratic
  - n^3 cubic
- 2^n (exponential)

Input size	n	n^2	$\log_{10} n$
10	10	100	1
100	100	10000	2
10^6	10^6	$(10^6)^2$	6


constant
logarithmic
polynomial
exponential

$$n^3 \quad > \quad n^2$$

$$n^2 \quad > \quad \log n$$

$$n^2 \log n \quad > \quad \mathrm{n} \log n^2$$

$$n \log n \quad < \quad n^2$$

LOOP

n

```
For i=1 to n
      print 'Hi'
```

10

```
For i=1 to n
      if i <11
         print 'Hi'
      else
         break
```

LOOP

Nested loop:

$$\sum_{i=1}^{n} n \Leftarrow n^2$$

```
For i=1 to n
    for j=1 to n
        print 'Hi'
```

n

Dependent nested loop:

$$\sum_{i=1}^{n} i \Leftarrow$$

```
For i=1 to n
    for j=1  to I
        print 'Hi'
```

$i = 1,\quad 1$
$i = 2,\quad 2$
.
.
.
$i = n,\quad n$

SUM EQUATIONS

Why study summations?

1. In general, the running time of a ***loop*** can be expressed as the sum of the running time of each iteration.
2. Summations come up in solving recurrences.

SUM EQUATIONS

- $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$
- $\sum_{i=1}^{n} constant\ or\ n = \quad * \quad = n^2$
- $\sum_{i=1}^{\log_2 n} 2^i = 2n - 1$

- $\sum_{i=1}^{\log_2 n} constant =$ *

- $\sum_{i=j}^{n} 1 = n - j + 1$

- $\sum_{i=1}^{n} x^i = \frac{x^{n+1}-1}{x-1}$ x→ constant

- $\sum_{i=1}^{n} i^2 = \frac{2n^3+3n^2+n}{6}$

- $\sum_{i=1}^{n} \frac{1}{i} = \log n + 1$

EXAMPLES

$$\sum_{i=1}^{n} 1$$

For (i=1, i≤ n,i++)

Print i

end

$$\sum_{i=1}^{n} constant\ or\ n = \quad * \quad = n^2$$

$$T(N) = \sum_{i=1}^{n} 1 = 1 * n = n$$

EXAMPLES

$$\sum_{i=1}^{n-1} 1$$

```
For (i=n-1, i≥ 1,i--)
      Print i
end
```

$$T(N) = \sum_{i=1}^{n-1} 1 = \quad 1 \quad * (n - 1) \quad = n - 1$$

EXAMPLES

$$\sum_{i=1}^{\log_2 n} 1 \left\{ \begin{array}{l} \text{For (i=1, i} \leq n\text{,i*=2)} \\ \quad \text{Print i} \end{array} \right.$$

end

$$\sum_{i=1}^{\log_2 n} constant = \quad *$$

$$T(N) = \sum_{i=1}^{\log_2 n} 1 = \quad 1 \quad * \log_2 n = \log_2 n$$

EXAMPLES

$$\sum_{i=1}^{\log_2 n} 1 \quad \{$$

For (i=1, i≤ n,i/=2)

Print i

end

$\sum_{i=1}^{\log_2 n} constant =$ *

$$T(N) = \sum_{i=1}^{\log_2 n} 1 = \quad 1 \quad * \log_2 n = \log_2 n$$

EXAMPLES

$\sum_{i=1}^{n} t$

For (i=1, i$\leq n$,i++)

  For (j=1, j$\leq n$,j++)

    print "hi"

  end

end

$\sum_{i=1}^{n} 1$

$\sum_{i=1}^{n} constant\ or\ n = \quad * \quad = n^2$

$$T(N) = \sum_{i=1}^{n}\sum_{j=1}^{n} 1 = \sum_{i=1}^{n} n = n^2$$

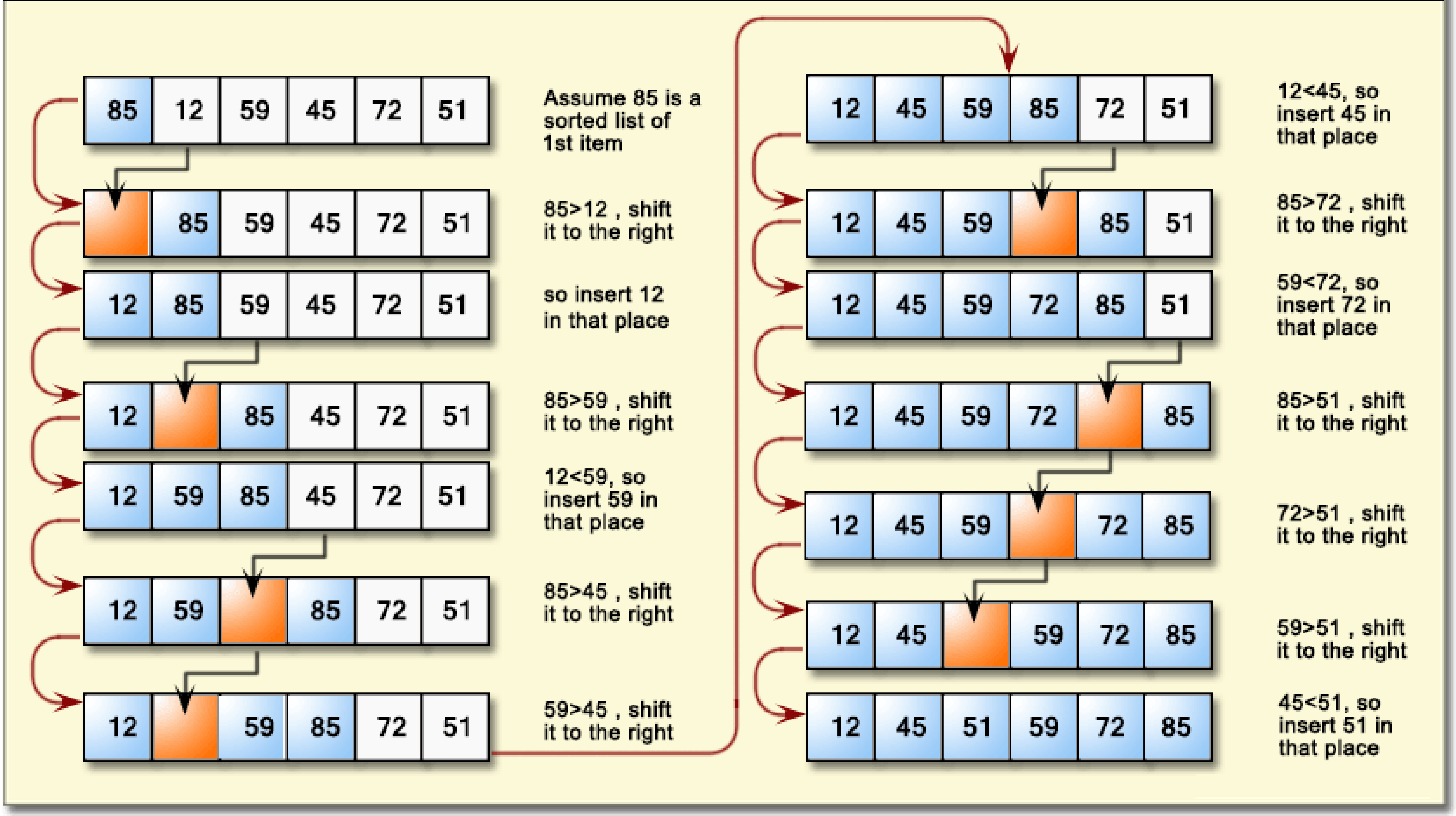
INSERTION SORT

- Our first algorithm, insertion sort, solves the ***sorting problem.***

Input: A sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$.

Output: A permutation (reordering) $\langle a'_1, a'_2, \ldots, a'_n \rangle$ of the input sequence such that $a'_1 \le a'_2 \le \cdots \le a'_n$.

Insertion Sort



INSERTION SORT

```
for j = 2 to A.length do
    key = A[j]
    i = j − 1
    while i > 0andA[i] > key do
        A[i + 1] = A[i]
        i = i − 1
    end while
    A[i + 1] = key
end for
```

$$T(N) = \sum_{j=2}^{n} t$$

The best case occurs

- If the array is already sorted, Thus t_j = 1 for j = 2;3; … ;n, and the best-case running time is

$$T(N) = \sum_{j=2}^{n} 1 = n$$

$$O(n)$$

The worst case occurs

- If the array is in reverse sorted order that is, in decreasing order—the worst case results. We must compare each element A[j] with each element in the entire sorted subarray A[1 .. j - 1], and so t_j = j for j = 2;3; … ;n

$$T(N) = \sum_{j=2}^{n} j = \frac{n(n+1)}{2} = \frac{n^2 + n}{2}$$

$$O(n^2)$$