Design and Analysis of Algorithms

Dina El-Manakhly, Ph. D.

dina_almnakhly@science.suez.edu.eg

Lecture2 (16/10/2022)

Strategy

- A **Strategy** is an approach or a design for solving a computational problem.

- **Example**
  - Greedy method
  - Dynamic programming
  - Backtracking
  - Branch and bound
  - Brute Force
  - Divide and conquer

Brute Force

- A brute-force algorithm solves a problem in the most simple, direct way.
- Brute Force search is the naive approach (intuitive).
- A brute force algorithm solves a problem through exhaustion: it goes through all possible choices until a solution is found.
- Example:

  If there is a lock of 4-digit PIN. The digits to be chosen from 0-9 then the brute force will be trying all possible combinations one by one like 0001, 0002, 0003, 0004, and so on until we get the right PIN. In the worst case, it will take 10,000 tries to find the right combination.
- Brute force algorithms are simple but very slow.

Advantages and Disadvantages of Brute Force

- **Advantages**
  - Widely Applicable
  - Easy (Do not think much to solve it)
  - Good for small problems

- **Disadvantages**
  - Often inefficient for large input sizes Because the complexity is high.

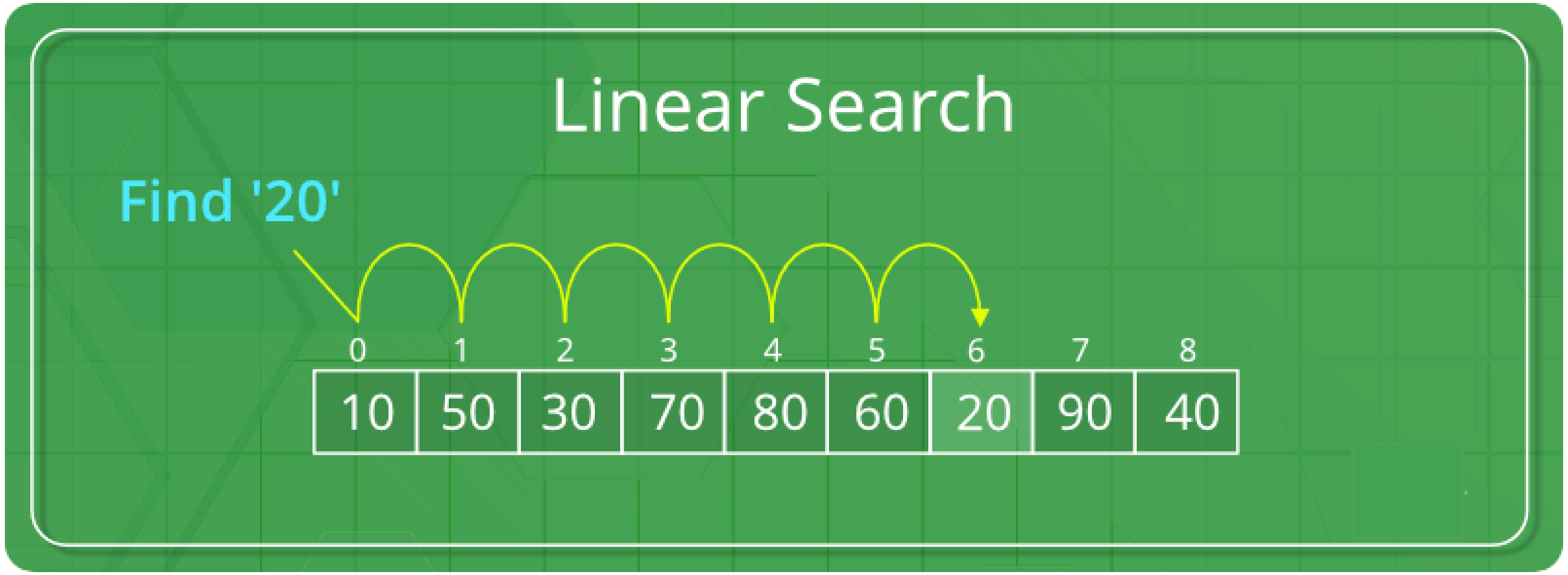
Complexity of an algorithm is a measure of the amount of time and/or space required by an algorithm for an input of a given size (n).

Some standard algorithms that follow Brute Force algorithm

- Linear Search.
- Selection Sort.
- Merging Problem.

Linear Search (Sequential Search)

- **Linear Search** *is defined as a sequential search algorithm that starts at one end and goes through each element of a list until the desired element is found, otherwise the search continues till the end of the data set. It is the easiest searching algorithm*



Linear Search (Sequential Search)

- **Problem Definition:** *Given an array $A=(a_1, a_2,\dots, a_n)$ of n elements and an element k. Find the smallest index, pos, of an occurrence of the element k in A if it exist. Otherwise, pos is equal to zero.*

Examples

Example 1: Given A=(2,4,9,6,3,10,7,1) and k=7 then search(A,k)=7

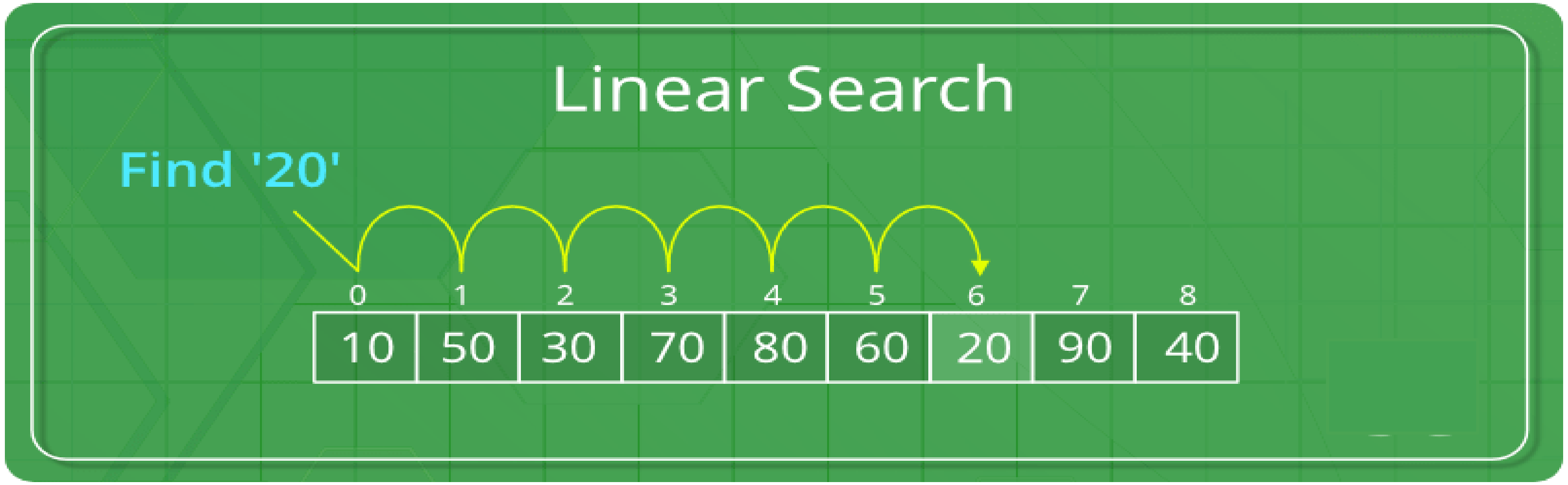
Example 2: Given A=(2,4,9,6,3,10,7,1) and k=17 then search(A,k)=0

Example 3: Given A=(2,4,9,6,9,10,9,1) and k=9 then search(A,k)=3

Linear Search (Sequential Search)

- The main idea of sequential search algorithm is scanning the array from the start element to final element in the array.
- In each iteration **i**, **1≤i ≤n**, we do the following test:

  if the element at index **i** is equal to the element **k** then return **i**. Otherwise increase **i** by one. Until the element **k** is found or return **0**.



Pseudo Code

Algorithm: Sequential_Search

Input: An array A=(a_1, $a_{2,\dots,}$ a_n) of n elements and the element k.

Output: return **i** if the element **k** equals the element $\mathbf{a_i}$. Otherwise, return **0**.

```
start
pos=0; i=1; found = false
while i≤n  and not found  do
        if  ai= k then
                pos= i
                found=true
        else
                i=i+1
 end while
 return pos
```

Tracing

1	2	3	4	5	6	7	8	9	10
12	4	60	19	3	71	34	66	1	34

34

Algorithm: Sequential_Search

Input: An array $A=(a_1, a_{2,\ldots,} a_n)$ of n elements and the element k.

Output: return **i** if the element **k** equals the element $\mathbf{a_i}$. Otherwise, return **0**.

start

pos=0; i=1; found = false

while i≤n and not found do

    if a_i= k then

        pos= i

        found=true

    else

        i=i+1

end while

return pos

Tracing

1	2	3	4	5	6	7	8	9	10
12	4	60	19	3	71	34	66	1	34

34

i=1

Pos=0

Found= false

1	2	3	4	5	6	7	8	9	10
12	4	60	19	3	71	34	66	1	34

Algorithm: Sequential_Search

Input: An array A=(a_1, $a_{2,\dots,}$ a_n) of n elements and the element k.

Output: return **i** if the element **k** equals the element **a_i**. Otherwise, return **0**.

start

pos=0; i=1; found = false

while i≤n and not found do

    if a_i= k then

        pos= i

        found=true

    else

        i=i+1

end while

return pos

Tracing

1	2	3	4	5	6	7	8	9	10
12	4	60	19	3	71	34	66	1	34

34

i=2

Pos=0

Found= false

1	2	3	4	5	6	7	8	9	10
12	4	60	19	3	71	34	66	1	34

Algorithm: Sequential_Search

Input: An array $A=(a_1, a_{2,\ldots,} a_n)$ of n elements and the element k.

Output: return **i** if the element **k** equals the element **a_i**. Otherwise, return **0**.

```
start
pos=0; i=1; found = false
while i≤n  and not found  do
        if  a_i= k then
                pos= i
                found=true
        else
                i=i+1
end while
return pos
```

Tracing

1	2	3	4	5	6	7	8	9	10
12	4	60	19	3	71	34	66	1	34

34

i=3

Pos=0

Found= false

1	2	3	4	5	6	7	8	9	10
12	4	60	19	3	71	34	66	1	34

Algorithm: Sequential_Search

Input: An array $A=(a_1, a_{2,\dots,} a_n)$ of n elements and the element k.

Output: return **i** if the element **k** equals the element $\mathbf{a_i}$. Otherwise, return **0**.

start

pos=0; i=1; found = false

while i≤n and not found do

    if a_i= k then

        pos= i

        found=true

    else

        i=i+1

end while

return pos

Tracing

1	2	3	4	5	6	7	8	9	10
12	4	60	19	3	71	34	66	1	34

34

i=4

Pos=0

Found= false

1	2	3	4	5	6	7	8	9	10
12	4	60	19	3	71	34	66	1	34

Algorithm: Sequential_Search

Input: An array A=(a_1, $a_{2,\dots,}$ a_n) of n elements and the element k.

Output: return **i** if the element **k** equals the element **a_i**. Otherwise, return **0**.

```
start
pos=0; i=1; found = false
while i≤n  and not found  do
        if  ai= k then
                pos= i
                found=true
        else
                i=i+1
 end while
 return pos
```

Tracing

1	2	3	4	5	6	7	8	9	10
12	4	60	19	3	71	34	66	1	34

34

i=5

Pos=0

Found= false

1	2	3	4	5	6	7	8	9	10
12	4	60	19	3	71	34	66	1	34

Algorithm: Sequential_Search

Input: An array A=(a_1, $a_{2,\ldots,}$ a_n) of n elements and the element k.

Output: return **i** if the element **k** equals the element **a_i**. Otherwise, return **0**.

start

pos=0; i=1; found = false

while i≤n and not found do

    if a_i= k then

        pos= i

        found=true

    else

        i=i+1

end while

return pos

Tracing

1	2	3	4	5	6	7	8	9	10
12	4	60	19	3	71	34	66	1	34

34

i=6

Pos=0

Found= false

1	2	3	4	5	6	7	8	9	10
12	4	60	19	3	71	34	66	1	34

Algorithm: Sequential_Search

Input: An array $A=(a_1, a_{2,\ldots,} a_n)$ of n elements and the element k.

Output: return **i** if the element **k** equals the element **a_i**. Otherwise, return **0**.

```
start
pos=0; i=1; found = false
while i≤n  and not found  do
        if  a_i= k then
                pos= i
                found=true
        else
                i=i+1
 end while
 return pos
```

Tracing

1	2	3	4	5	6	7	8	9	10
12	4	60	19	3	71	34	66	1	34

34

i=7

Pos=7

Found= true

1	2	3	4	5	6	7	8	9	10
12	4	60	19	3	71	34	66	1	34

Algorithm: Sequential_Search

Input: An array A=(a_1, $a_{2,\ldots,}$ a_n) of n elements and the element k.

Output: return **i** if the element **k** equals the element **a_i**. Otherwise, return **0**.

```
start
pos=0; i=1; found = false
while i≤n  and not found  do
    if  ai= k then
        pos= i
        found=true
    else
        i=i+1
end while
return pos
```

Selection Sort

- **Problem Definition:** Given an array A=(a_1, a_2,…, a_n) of n elements. Sorting the array is rearrangement the elements of the array such that $a_i \leq a_{i+1}$, $1 \leq i \leq n$-1.

Examples

Example 1: Given A=(2,4,9,6,3,10,7,1)

Sort(A)=(1,2,3,4,6,7,9,10)

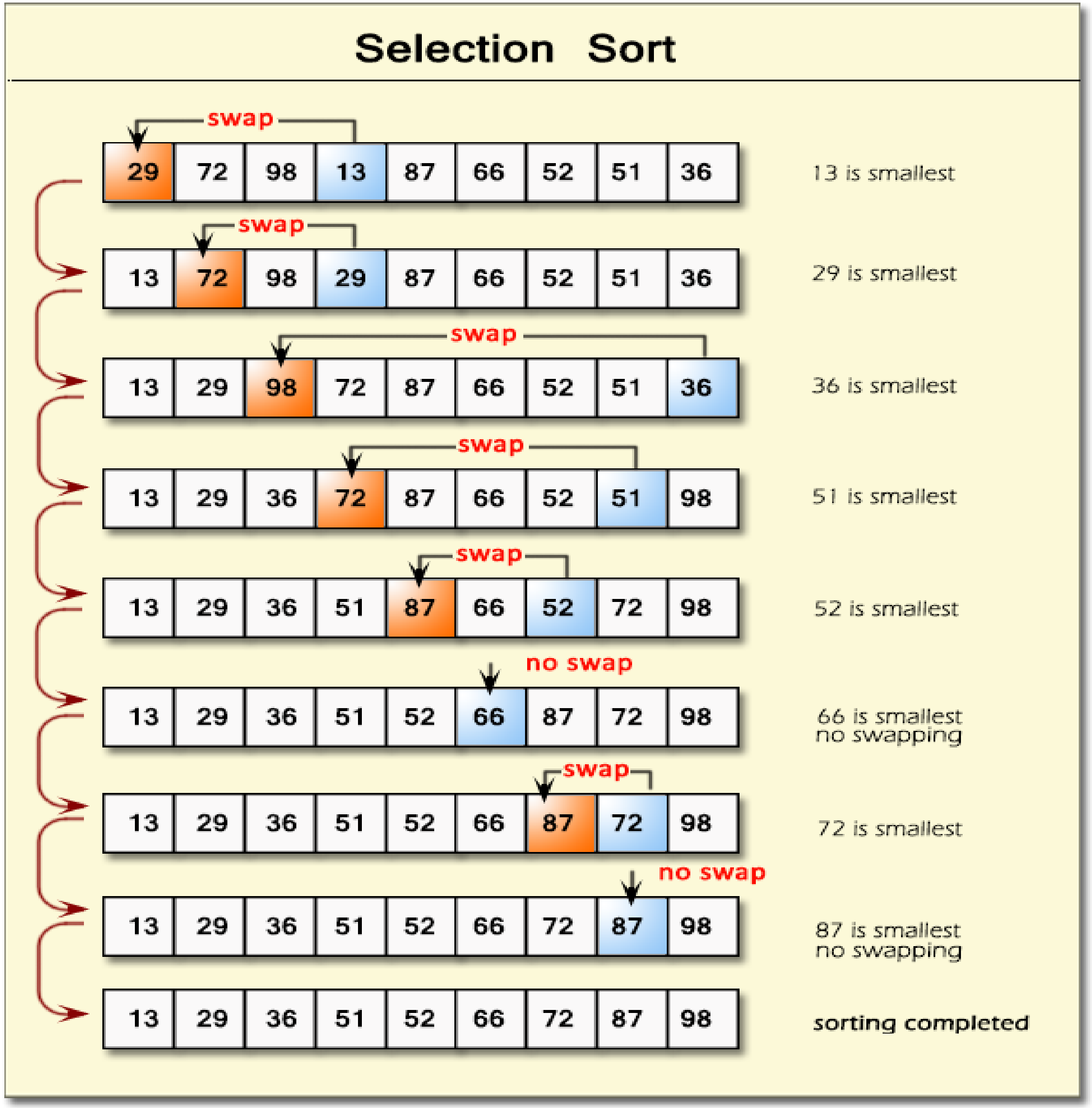
Example 2: Given A=(2,4,9,6,9,10,9,1)

Sort(A)=(1,2,4,6,9,9,9,10)

Selection Sort

The main idea of selection sort algorithm as follows:

- First, we find the minimum element of the array A and store it in a_1.
- Next, we find the minimum of the remaining n-1 elements and store it in a_2.
- We continue this way until the second largest element is stored in a_{n-1} and the largest element of A is stored in a_n.



Selection Sort Pseudo Code

```
for i = 1 to n − 1 do
    min = i
    for j = i + 1 to n do
        // Find the index of the i^th smallest element
        if A[j] < A[min] then
            min = j
        end if
    end for
    Swap A[min] and A[i]
end for
```

Merge two sorted arrays Problem

- **Problem Definition:** Given two sorted arrays $A=(a_1, a_2,\ldots, a_n)$ and $B=(b_1, b_2,\ldots, b_m)$ of ***n*** and ***m*** elements respectively. Merging the two sorted arrays is an array $C=(c_1, c_2,\ldots, c_{n+m})$ of ***n+m*** elements such that:

(i) $c_i \in C$ belongs to A or B, $\forall\ 1 \le i \le n+m$.

(ii) a_i and b_j appear exactly once in C, $\forall\ 1 \le i \le n$ *and* $1 \le j \le m$.

Examples

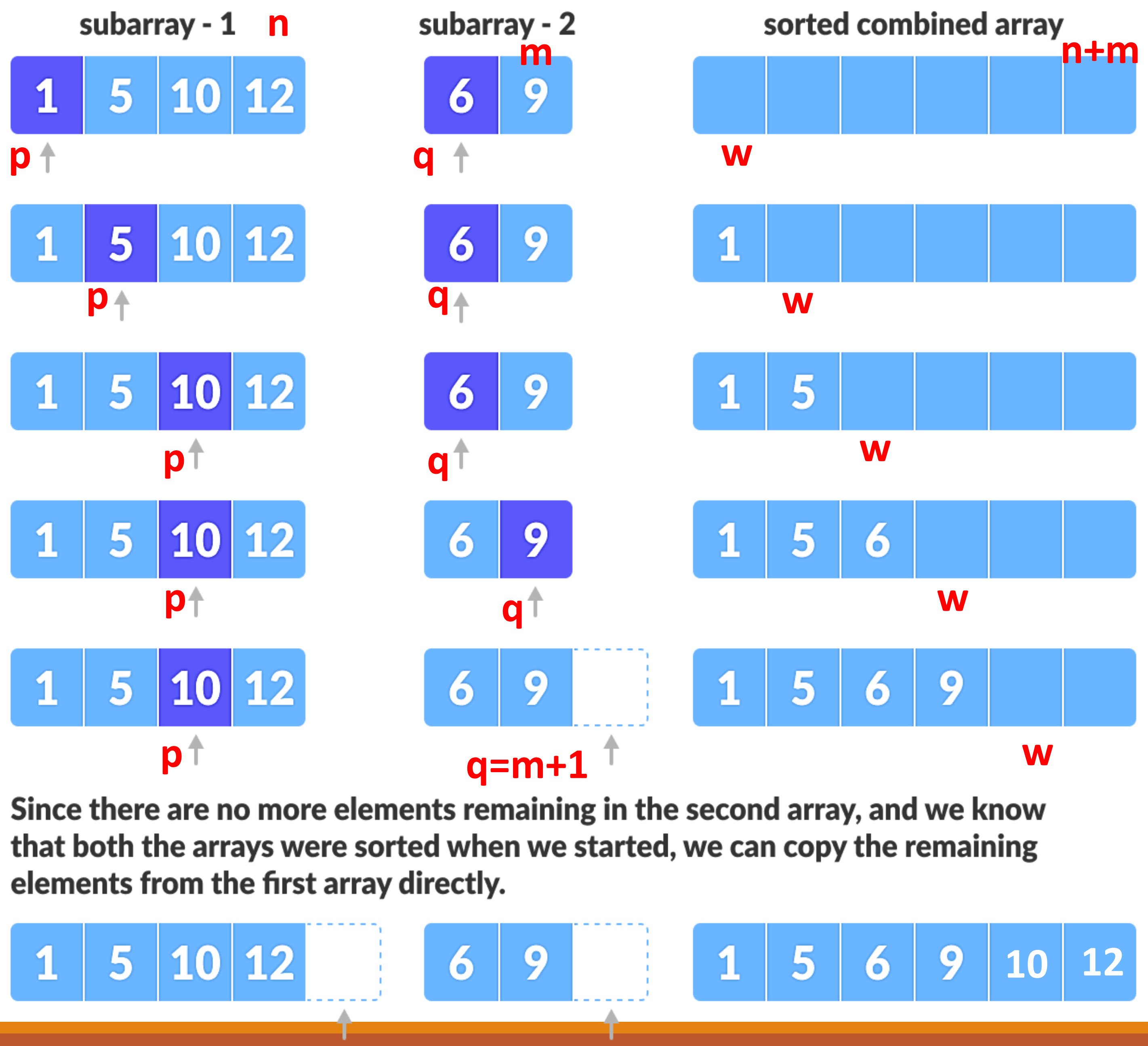
Example 1: Given A=(1,3,4,5,10) and *B*=(2,3,3,7,8)

Merge(A,B)=(1,2,3,3,3,4,5,7,8,10)

Merge two sorted arrays Problem

Main Idea: The main idea of merging algorithm as follows:

- We maintain two pointers **p** and **q** that initially point to a_1 and b_1 respectively.
- In each step, we compare the elements a_p and b_q . If a_p is less than or equal b_q then append a_p to the array C at position w. Then increment p and w by 1. Otherwise, append b_q to the array C at position w. Then increment q and w by 1.
- This process ends when p=n+1 or q=m+1. In case of p=n+1, we append the remaining elements B(q..m) to C(w..n+m). In the second case (q=m+1), we append A(p..n) to array C(w..n+m).



Pseudo Code

Algorithm: Merging

Input: Two sorted arrays $A=(a_1, a_2,\ldots, a_n)$ and $B=(b_1, b_2,\ldots, b_m)$ of n and m elements respectively.

Output: Sorted array $C=(c_1, c_2,\ldots, c_{n+m})$ s.t. (i) $c_i \in C$ belongs to A or B, $\forall\ 1 \le i \le n+m$. (ii) a_i and b_j appear exactly once in C, $\forall\ 1 \le i \le n$ and $1 \le j \le m$.

Begin

1. $p=q=w=1$

2. While $p \le n$ and $q \le m$ do

    if $a_p \le b_q$ Then

        $c_w = a_p$, p=p+1, w=w+1

    else $c_w = b_q$, q=q+1, w=w+1

3. If p > n then $C(c_w, c_{w+1},\ldots, c_{n+m})=B(b_q, b_{q+1},\ldots, b_m)$

    if q> m then $C(c_w, c_{w+1},\ldots, c_{n+m})=A(a_p, a_{p+1},\ldots, a_n)$

End.

Assignment 1

- Design an algorithm using brute force approach to compute 2^{n}.