## Sorting Algorithms I

## Dr. Mohamed Loey

Lecturer, Faculty of Computers and Information Benha University Egypt

## Table of Contents

## Sorting Algorithms

## Bubble Sort

## Selection Sort

## Insertion Sort

## Sorting Algorithms

$\square$ Sorting Algorithm is an algorithm made up of a series of instructions that takes an array as input, and outputs a sorted array.

There are many sorting algorithms, such as:

- Selection Sort, Bubble Sort, Insertion Sort, Merge Sort, Heap Sort, QuickSort, Radix Sort, Counting Sort, Bucket Sort, ShellSort, Comb Sort, Pigeonhole Sort, Cycle Sort


## Bubble Sort

## Bubble Sort

-Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

## Bubble Sort

## -Algorithm:

- Step1: Compare each pair of adjacent elements in the list
- Step2: Swap two element if necessary
- Step3: Repeat this process for all the elements until the entire array is sorted


## Bubble Sort

## Example 1 Assume the following Array:




1

都
$\qquad$

## Bubble Sort

## $\square$ First Iteration:

## a Compare



| $\uparrow$ | $\uparrow$ |
| :---: | :---: |
| $j$ | $j+1$ |

## 

1
+
$j+1$

\section*{\author{

}}
$\qquad$


## I



都


#### Abstract




## $\square$ First Iteration:

## $\square$ Swap

## Bubble Sort

| 1 | 5 | 4 | 2 |  |
| :---: | :---: | :---: | :---: | :---: |
| $\uparrow$ | $\uparrow$ |  |  |  |
| $j$ | $j+1$ |  |  |  |

j
J (都

$\qquad$

## .

## Bubble Sort

## $\square$ First lteration:

$\square$ Compare


nalysis and Design of Algorithms $\qquad$
-


#### Abstract









| 1 | 5 | 4 | 2 |
| :---: | :---: | :---: | :---: |
|  | $\uparrow$ | $\uparrow$ |  |
|  | $j$ | $j+1$ |  |

## 

```
        O
```

$\square$



## $\square$ First lteration:

a Swap

## Bubble Sort



| 1 | $\mathbf{4}$ | $\mathbf{5}$ | $\mathbf{2}$ |
| :---: | :---: | :---: | :---: |
|  | $\uparrow$ <br> $j$ | $\uparrow$ <br> $j+1$ |  |



$$
1
$$

I.

-

## Bubble Sort

## $\square$ First lteration:

$\square$ Compare


## $\square$ First Iteration:

a swap

## Bubble Sort

| 1 | $\mathbf{4}$ | $\mathbf{2}$ | $\mathbf{5}$ |
| :--- | :--- | :--- | :---: |

- j+1


## Bubble Sort <br> 都




## Bubble Sort

$\square$ Second Iteration:
$\square$ Compare

| 1 | 4 | 2 | 5 |
| :---: | :---: | :---: | :---: |
| $\uparrow$ | $\uparrow$ |  |  |
| $j$ | $j+1$ |  |  |

J+1

## Bubble Sort

## $\square$ Second Iteration:

## $\square$ Compare



## Bubble Sort

## $\square$ Second Iteration:

## $\square$ Swap

 .


## 



另
$+$
$\qquad$
$\qquad$
$\qquad$

## Bubble Sort <br> St


12

## Bubble Sort

## Third Iteration:

## $\square$ Compare

## 


$\qquad$

## Bubble Sort



## Bubble Sort

## $\square$ Array is now sorted



2 3

4

## Bubble Sort

- Example 2:

| 5 | 4 | 2 | 1 | 3 |
| :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- | | 2 | 4 | 1 | 3 | 5 |
| :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- |
| 4 | 5 | 2 | 1 | 3 | | 2 | 1 | 4 | 3 | 5 |
| :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- |
| 4 | 2 | 5 | 1 | 3 | | 2 | 1 | 3 | 4 | 5 |
| :--- | :--- | :--- | :--- | :--- |
| 4 | 2 | 1 | 5 | 3 | | 2 | 1 | 3 | 4 | 5 |
| :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- | :--- |
| 4 | 2 | 1 | 3 | 5 |
| 4 | 1 2 3 4 5 <br> 4 2 1 3 51 2 3 4 5 |  |  |  |

## Bubble Sort

What is the output of bubble sort after the 1st iteration given the following sequence of numbers: 1329418453763
a) 2491318374563
b) 2941318374563
c) 1324918453763
d) 2491318453763

## Bubble Sort

What is the output of bubble sort after the 1 st iteration given the following sequence of numbers: 1329418453763
a) 2491318374563
b) 2941318374563
c) 1324918453763
d) 2491318453763

## Bubble Sort

## $\square$ Python Code

def BubbleSort(arr):
return arr

```
for i in range(len(arr)-1):
for \(j\) in range(len(arr)-i-1): if \(\operatorname{arr}[j]>\operatorname{arr}[j+1]\) : \(\operatorname{arr}[j], \operatorname{arr}[j+1]=\operatorname{arr}[j+1], \operatorname{arr}[j]\)
for ilin range(len(arr)-1):
    for j in range(len(arr)-i-1).
        if arr[j]> arr[j+1].
        arr[j], arr[j+1] = arr[j+1], arr[j]
\[
1-2
\]
```

    return arr
    
## Bubble Sort

$$
\begin{aligned}
& \text { arr }=[5,1,4,2] \\
& \text { Sortedarr=BubbleSort(arr) } \\
& \text { print(Sortedarr) }
\end{aligned}
$$

## Bubble Sort <br> 

\author{

- Example of worst case
} <br> \section*{\section*{Time Complexity: $\mathrm{O}\left(\mathrm{n}^{2}\right)$ as there are two nested loops}} <br> \section*{\section*{Time Complexity: $\mathrm{O}\left(\mathrm{n}^{2}\right)$ as there are two nested loops}}

| 5 | 4 | 3 | 2 | 1 |
| :--- | :--- | :--- | :--- | :--- |

[^0] 2
exity: $O\left(n^{2}\right)$ as there are two nested loops

 .

-
 $\pi$

正

## Selection Sort

## Selection Sort

The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning.

## Selection Sort

## -Algorithm:

- Step1: Find the minimum value in the list
- Step2: Swap it with the value in the current position
- Step3: Repeat this process for all the elements until the entire array is sorted


## Selection Sort

| 8 | 12 | 5 | 9 | 2 |
| :--- | :--- | :--- | :--- | :--- |

## Example 1 Assume the following Array:



## Selection Sort

- Compare



## Selection Sort



## - Compare

$\qquad$

## Selection Sort

$$
\min
$$

| 8 | 12 | 5 | 9 | 2 |
| :---: | :---: | :---: | :---: | :---: |
| + |  | $\uparrow$ |  |  |
|  |  | j |  |  |
|  |  | $\uparrow$ $\min$ |  |  |

## $\square$ Move

$\qquad$

## Selection Sort

$\square$ Compare


## Selection Sort

- Compare



## Selection Sort

## $\square$ Move

| 8 | 12 | 5 | 9 | 2 |
| :---: | :---: | :---: | :---: | :---: |
| $i$ |  |  | $\uparrow$ |  |
| $i$ |  |  |  |  |

## Selection Sort

## $\square$ Smallest


$\qquad$

## Selection Sort

## Swap



## Selection Sort

$\square$ Sorted

## UUn Sorted





| 2 | 12 | 5 | 9 | 8 |
| :---: | :---: | :---: | :---: | :---: |
| $\uparrow$ <br> Sorted | $\uparrow$ <br> Un Sorted |  |  |  |

$\qquad$

## Selection Sort

## a Compare

| 2 | 12 | 5 | 9 | 8 |
| :---: | :---: | :---: | :---: | :---: |
| $\uparrow$ | $\uparrow$ | $\uparrow$ |  |  |
| Sorted | i | j |  |  |
|  | $\underset{\min }{\uparrow}$ |  |  |  |

$$
\min
$$

## Selection Sort

## $\square$ Move

 $Z$| 2 | 12 | 5 | 9 | 8 |
| :---: | :---: | :---: | :---: | :---: |
| $\uparrow$ | $\uparrow$ | $\uparrow$ |  |  |
| Sorted | i | j |  |  |
|  |  | $\uparrow$ $\min$ |  |  |



## Selection Sort

## a Compare

| 2 | 12 | 5 | 9 | 8 |
| :---: | :---: | :---: | :---: | :---: |
| $\uparrow$ <br> Sorted | $\uparrow$ | $\uparrow$ | $\uparrow$ |  |



## Selection Sort

## a Compare


(

## Selection Sort

## $\square$ Smallest

| 2 | 12 | 5 | 9 | 8 |
| :---: | :---: | :---: | :---: | :---: |
| $\uparrow$ <br> Sorted | $\uparrow$ | $\uparrow$ |  |  |

## Selection Sort

## $\square$ Swap

| 2 | 12 | 5 | 9 | 8 |
| :---: | :---: | :---: | :---: | :---: |
| $\uparrow$ <br> Sorted | $\uparrow$ | $\uparrow$ |  |  |

$$
\min
$$ .

## Selection Sort

$\square$ Sorted

## - Un Sorted



## Selection Sort

$\square$ Compare

$\begin{array}{ccc}\uparrow & \uparrow & \uparrow \\ \text { Sorted } & i & j \\ & \uparrow & \end{array}$

## Selection Sort

## - Move




## Selection Sort <br> Selecion Sort

- Compare


## 



## Selection Sort

| 2 | 5 | 12 | 9 | 8 |
| :---: | :---: | :---: | :---: | :---: |
| $\begin{gathered} \uparrow \\ \text { Sorted } \end{gathered}$ |  | $\uparrow$ |  | $\uparrow$ |
|  |  | j |
|  |  |  |  | $\uparrow$ |

## - Move

 2 2

## Selection Sort

## $\square$ Smallest

| 2 | 5 | 12 | 9 | 8 |
| :---: | :---: | :---: | :---: | :---: |
|  |  | $\begin{aligned} & \uparrow \\ & i \end{aligned}$ |  | $\underset{\text { min }}{\uparrow}$ |

 -

## Selection Sort

## Swap



## Selection Sort

$\square$ Sorted

## - Un Sorted

 (
min

## Selection Sort <br> Selecion Sort

| 2 | 5 | 8 | 9 | 12 |
| :---: | :---: | :---: | :---: | :---: |
| $\uparrow$ Sorted |  |  | $\uparrow$ | $\uparrow$ |
|  |  |  | i | j |
|  |  |  | $\underset{\text { min }}{\uparrow}$ |  |

## .

 , - -


## $\square$ Compare

2
$\stackrel{\uparrow}{\text { Sorted }}$

## Selection Sort

$\square$ Sorted

## - Un Sorted

| 2 | 5 | 8 | 9 | 12 |
| :---: | :---: | :---: | :---: | :---: |

$\stackrel{\uparrow}{\text { 个 }}$
Un Sorted
Sorted

Un

## Selection Sort

$\square$ Sorted

## U Un Sorted

| 2 | 5 | 8 | 9 | 12 |
| :---: | :---: | :---: | :---: | :---: |
| $\uparrow$ |  |  |  | $\uparrow$ |
| Sorted |  |  |  | i |
|  |  |  |  | $\underset{\min }{\uparrow}$ |

$\begin{array}{cc}\uparrow & \uparrow \\ \text { Sorted } & i \\ & \uparrow \\ & \text { min }\end{array}$

## Selection Sort

## $\square$ Array is now sorted



## Selection Sort

$\square$ Example 2:

| 12 | 10 | 16 | 11 | 9 | 7 |
| :--- | :--- | :--- | :--- | :--- | :--- |


| 7 | 10 | 16 | 11 | 9 | 12 |
| :---: | :---: | :---: | :---: | :---: | :---: |
| 7 | 9 | 16 | 11 | 10 | 12 |
| 7 | 9 | 10 | 11 | 16 | 12 |


| 7 | 9 | 10 | 11 | 16 | 12 |
| :--- | :--- | :--- | :--- | :--- | :--- |


| 7 | 9 | 10 | 11 | 12 | 16 |
| :--- | :--- | :--- | :--- | :--- | :--- |

## Selection Sort

What is the output of selection sort after the 2nd iteration given the following sequence of numbers: 1329418453763
a) 2491318374563
b) 2941318374563
c) 1324918453763
d) 2491318453763

## Selection Sort

What is the output of selection sort after the 2nd iteration given the following sequence of numbers: 1329418453763
a) 2491318374563
b) 2941318374563
c) 1324918453763
d) 2491318453763

## Selection Sort

## $\square$ Python Code

def SelectionSort(A):
for i in range(len(A)): minind = i
for $j$ in range(i+1, len(A)): if $A[m i n i n d]>A[j]:$
minind $=j$
$A[i], A[m i n i n d]=A[m i n i n d], A[i]$
return $A$

## Selection Sort

# $\operatorname{arr}=[8,12,5,9,2]$ Sortedarr=SelectionSort(arr) print(Sortedarr) 

## Selection Sort

Example of worst case

## Time Complexity: $O\left(n^{2}\right)$ as there are two nested loops

| 2 | 3 | 4 | 5 | 1 |
| :--- | :--- | :--- | :--- | :--- |

> | 2 | 3 | 4 | 5 | 1 |
| :--- | :--- | :--- | :--- | :--- | 2

## Insertion Sort

## Dlnsertion sort is a simple sorting

 algorithm that works the way we sort playing cards in our hands.
## Insertion Sort

## -Algorithm:

- Step1: Compare each pair of adjacent elements in the list
- Step2: Insert element into the sorted list, until it occupies correct position.
- Step3: Swap two element if necessary
- Step4: Repeat this process for all the elements until the entire array is sorted


## Insertion Sort

## Assume the following Array:

(

## 



| 5 | 1 | 4 | 2 |
| :--- | :--- | :--- | :--- |

[^1]$\qquad$

-
-
(
-
$\square$ Compare


## Insertion Sort

 - Store= 1

| $\uparrow$ | $\uparrow$ |
| :---: | :---: |
| $j$ | $i$ |
|  | $\uparrow$ |
|  | $j+1$ |

j+1


#### Abstract




## +

 $\begin{array}{lll}5 & 1 \\ & \\ j & \end{array}$正

| 5 | 1 | 4 | 2 |
| :--- | :--- | :--- | :--- |



$\square$














## Insertion Sort

## $\square$ Move



| $\uparrow$ | $\uparrow$ |
| :---: | :---: |
| $j$ | $\mathbf{i}$ |
|  | $\uparrow$ |
|  | $j+1$ |

j+1

Insertion
a Move

## Insertion Sort

| 1 | 5 |
| :---: | :---: |
| $\underset{i}{1}$ | $\uparrow$ |
| $j+1$ | $i$ |

$\begin{array}{cc}\uparrow \\ j+1 & \uparrow \\ i\end{array}$
$\begin{array}{cc}\uparrow & \uparrow \\ j+1 & i\end{array}$
$\square$ Store $=\square$
$\square$
2

 $[1$




$\qquad$

$$
1
$$

$\qquad$


$\qquad$

ji
j+1


## Insertion Sort

## $\square$ Compare

教




路
$\qquad$



.

 $\square$


#### Abstract




$\square$ Move
$\square$ Store $=4$

## Insertion Sort



| 1 |  | 5 | $\mathbf{2}$ |
| :---: | :---: | :---: | :---: |
|  | $\uparrow$ | $\uparrow$ |  |
|  | $j$ | $i$ |  |
|  |  | $\uparrow$ |  |
|  |  |  |  |
|  |  |  |  |

$\square$ Compare

## - Store=


$\square$

## Insertion Sort

 (j+1

$$
{ }^{\mathrm{j}+1}
$$


$\qquad$


$$
0
$$

##  





## Insertion Sort

## - Move

$\square$ Store $=\square$
j+1
?
$\square$


$\qquad$.




$=$
 .
(1)
$\qquad$
$\square$ Compare

## Insertion Sort

| 5 | 2 |
| :---: | :---: |
| $\uparrow$ | $\uparrow$ |
| $j$ | $i$ |
|  | $\uparrow$ |
|  | $j+1$ |

 | 1 | $\mathbf{4}$ | 5 | $\mathbf{2}$ |
| :---: | :---: | :---: | :---: |
|  |  | $\uparrow$ | $\uparrow$ |
|  |  | $j$ | $i$ |
|  |  |  | $\uparrow$ |
|  |  |  |  |

        j+1
        j+1
        j+1
        j+1
        j+1
    j+1
    j+1
    j+1
    j+1
    j+1
                        1
                        1
                        1
                        1
                        1
    

nalysis and Design of Algorithms $\qquad$
$-$
$\qquad$




- 

(

Dr M



2 $\square$





(NSPR
(NSPR









,

$\qquad$


$\qquad$

$\qquad$

$\qquad$


$\square$
$\qquad$
$\qquad$
(NSPR





- Compare



## Insertion Sort

Store= 2


 --( -
 ( ( C $\square$ C
 .

$\qquad$
$\qquad$


$\qquad$

Insertion Sort
$\square$ Move
$\square$ Store $=$
nserton Sort
$\square$ Store=
Ane
Analysis and Design of Algor


| 1 |  | 4 | 5 |
| :---: | :---: | :---: | :---: |
|  |  | $\uparrow$ | $\uparrow$ |
| $j$ | $j+1$ | $\uparrow$ |  |




#### Abstract

.


 $\square$ Move
$\square$ Store $=$

Store=

$$
\text { Pres }
$$

Analysis and Design of Algorithms


$+$
$\qquad$
Store=

$$
\text { Pres }
$$

Analysis and Design of Algorithms
$\qquad$
$\qquad$
$\qquad$
$\qquad$
Store=

$$
\text { Pres }
$$

Analysis and Design of Algorithms
$\qquad$
Store=

$$
\text { Pres }
$$

Analysis and Design of Algorithms
Store=

$$
\text { Pres }
$$

Analysis and Design of Algorithms










$\qquad$
$\qquad$



$\qquad$

$\qquad$


| 2 |  |  |
| :---: | :---: | :---: |
| 1 |  | 4 |




#### Abstract

$\qquad$


Store=

$$
\text { Pres }
$$

Analysis and Design of Algorithms

| 2 |  |  |
| :---: | :---: | :---: |
| 1 |  | 4 |



insertion Sort
$\square$ Move
$\square$ Store=
$\qquad$
$\qquad$

$\qquad$


$\square$


#### Abstract

$\qquad$


$\qquad$
$\qquad$
$\qquad$ 都

$\qquad$
$\qquad$
$\qquad$
$\qquad$
es
es











$\qquad$

## $$
2-0-120
$$ <br>  <br> - <br>  <br>  <br> | 1 |  | 4 | 5 |
| :--- | :--- | :--- | :--- |



$\qquad$

$\qquad$

$\qquad$ ,
2
都

## Insertion Sort

$\square$ Store $=\square$

| $\mathbf{1}$ | $\mathbf{2}$ | $\mathbf{4}$ | $\mathbf{5}$ |
| :---: | :---: | :---: | :---: |
| $\uparrow$ | $\uparrow$ |  | $\uparrow$ |
| $j$ | $j+1$ |  | $i$ |

## $\square$ Compare

$\square$



#### Abstract






$\underbrace{2}$



nalysis and Design of Algorithms

$\qquad$

## $\square$ Array is now sorted


$\qquad$
都




WSOPAO
.

## Selection Sort

- Example 2:

| 5 | 1 | 8 | 3 | 9 | 2 |
| :--- | :--- | :--- | :--- | :--- | :--- |



## Insertion Sort

What is the output of insertion sort after the 1 st iteration given the following sequence of numbers: 73519846

> a) 37519846
> b) 13759846
> c) 34156879
> d) 13456789

## Insertion Sort

What is the output of insertion sort after the 1 st iteration given the following sequence of numbers: 73519846
a) 37519846
b) 13759846
c) 34156879
d) 13456789

## Insertion Sort

What is the output of insertion sort after the $2^{\text {nd }}$ iteration given the following sequence of numbers: 73519846

> a) 35719846
> b) 13759846
> c) 34156879
> d) 13456789

## Insertion Sort

What is the output of insertion sort after the $2^{\text {nd }}$ iteration given the following sequence of numbers: 73519846
a) 35719846
b) 13759846
c) 34156879
d) 13456789

## $\square$ Python Code

```
def InsertionSort(arr):
for i in range(1, len(arr)):
    store = arr[i]
    j = i-1
    while j >=0 and store < arr[j] :
        arr[j+1] = arr[j]
        j -= 1
        arr[j+1] = store
return arr
```


## Insertion Sort

$$
\begin{aligned}
& \operatorname{arr}=[12,6,5,14,3] \\
& \text { Sortedarr=InsertionSort(arr) } \\
& \text { print(Sortedarr) }
\end{aligned}
$$

| 5 | 4 | 3 | 2 | 1 |
| :--- | :--- | :--- | :--- | :--- |


 (2.end

 | 5 | 4 | 3 | 2 | 1 |
| :--- | :--- | :--- | :--- | :--- |












## Contact Me



## THANKS FOR YOUR TIME




[^0]:    analysis and Design of Algorithms

[^1]:    alysis and Design of Algorithms

