# Problem Solving and Programming CSE1001

Prof. Tulasi Prasad Sariki

October 8, 2019

## Sorting

**PROBLEM**

Results of VIT entrance exam has been released. Given the details of the students such as name, address and marks scored in entrance, write a program to sort the student details so that it will be convenient to call for counselling.

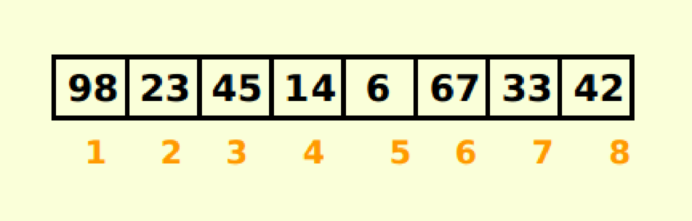
- **Sorting:**
  - It rearranges the elements into either ascending order or descending order.
  - **Ascending Order ??**
  - **Descending Order ??**
  - **Example:**
    - we have elements like **36, 24, 10, 6 and 12**
    - The elements after sorting (ascending order) is **6, 10, 12, 24 and 36**

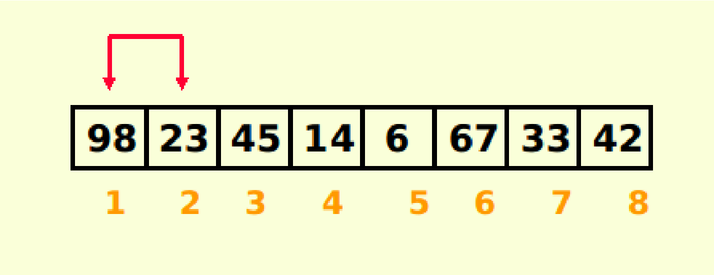
## Sorting

- There are several sorting algorithms available like **bubble sort**, **selection sort**, **insertion sort**, **quick sort**, **merge sort**, **radix sort** etc.
- Sorting operation is performed in many applications to provide the output in desired order.
- For example listing all the product in the increasing order of their names or decreasing order of supplier names.
- Searching will be easier in a sorted collection of elements
- List containing exam scores sorted from lowest to highest or vice versa.
- We will learn **Bubble sorting**, **Selection Sorting** and **Insertion Sorting** in this session.

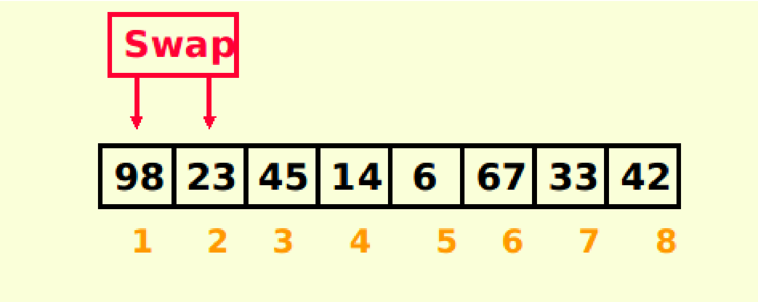
- It is a popular and simple algorithm for sorting data.
- This algorithm is not so efficient.
- **Iverson** was the first to use name "bubble sort" in 1962, even though used earlier.
- Unfortunately it is commonly used where the number of elements is too large.
- **Procedure:**
  - Starts at one end of the list and make repeated scans through the list comparing successive pairs of elements.
  - If the first element is larger than the second, called an "inversion", then the values are swapped.
  - Each scan will push the maximum element to the top.
  - This is the "bubbling" effect → name → bubble sort.
  - This process is continued until the list is sorted.
  - More swaps → More time for sorting.

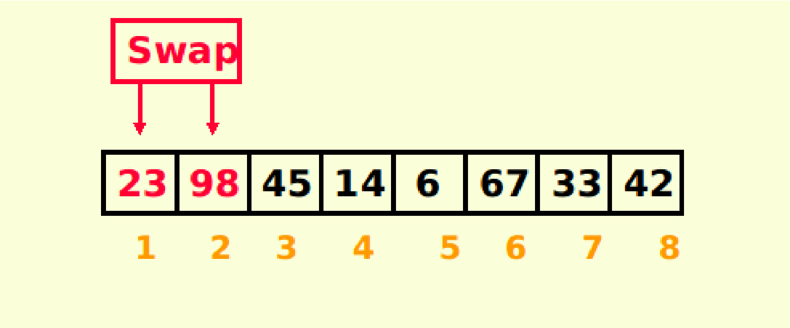
# Bubble Sort Example:



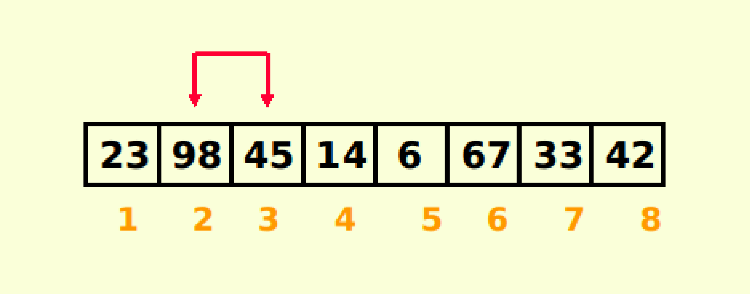

98
23
45
14
6
67
33
42
1
2
3
4
5
6
7
8

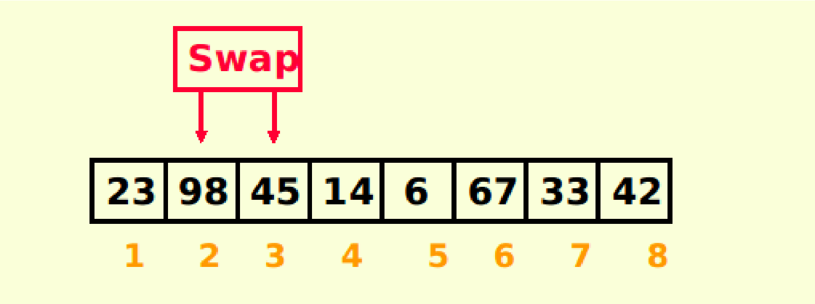
# Bubble Sort Example:

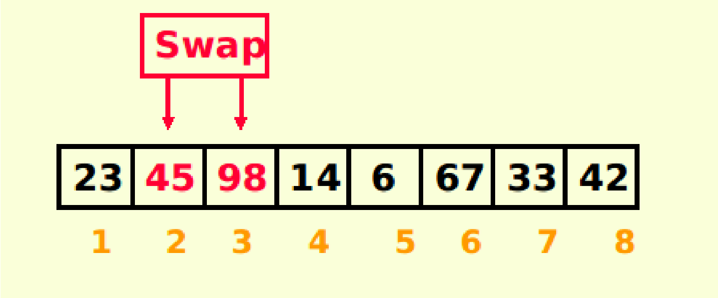



Swap
23
98
45
14
6
67
33
42
1
2
3
4
5
6
7
8

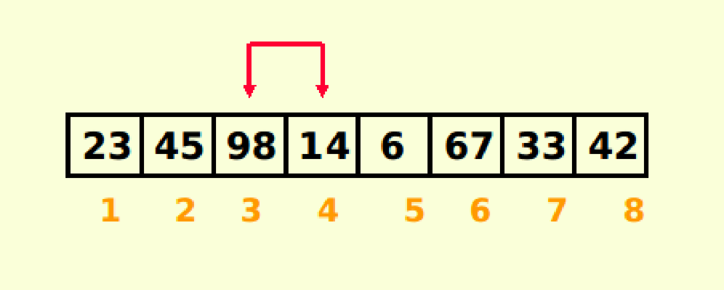
# Bubble Sort Example:



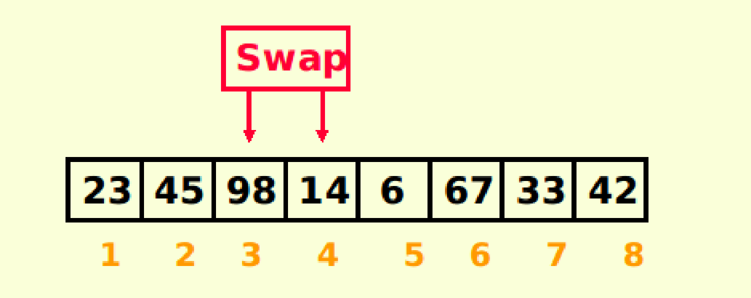

Swap
23
98
45
14
6
67
33
42
1
2
3
4
5
6
7
8

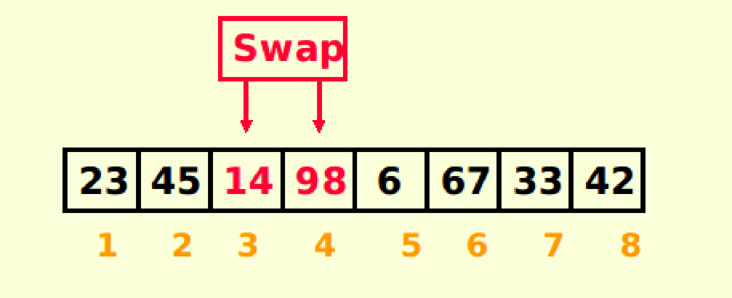

Swap
23
45
98
14
6
67
33
42
1
2
3
4
5
6
7
8

# Bubble Sort Example:

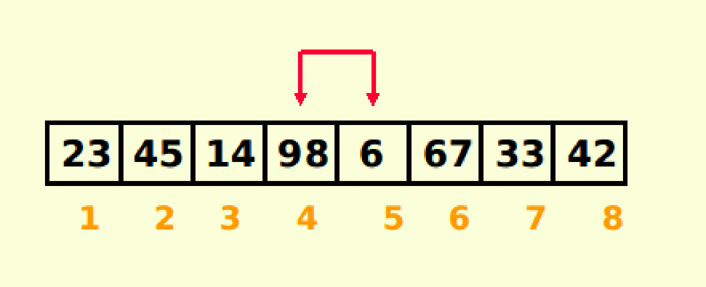


# Bubble Sort Example:

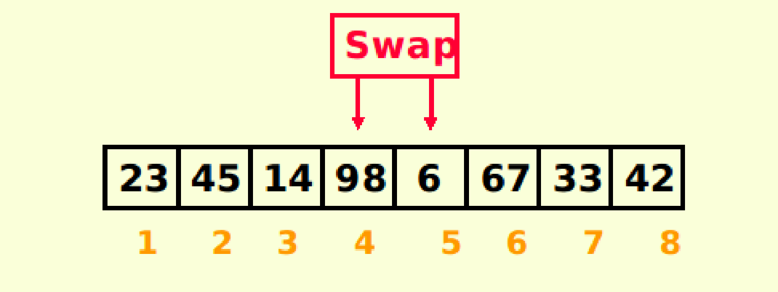



Swap
23
45
14
98
6
67
33
42
1
2
3
4
5
6
7
8

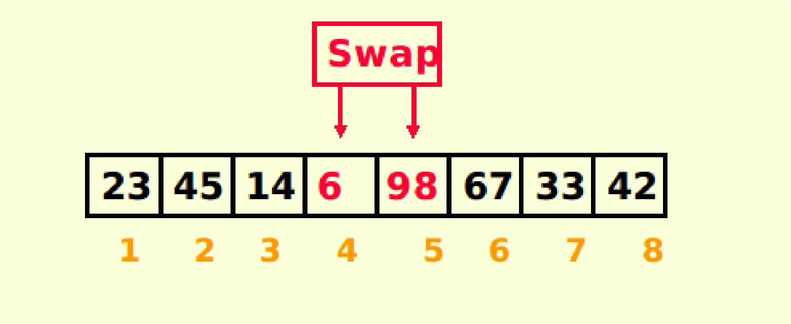
# Bubble Sort Example:



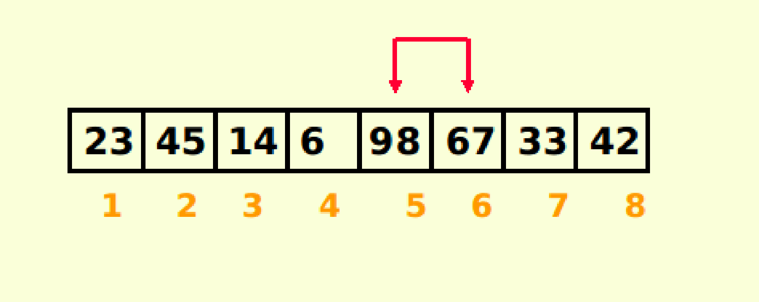
# Bubble Sort Example:



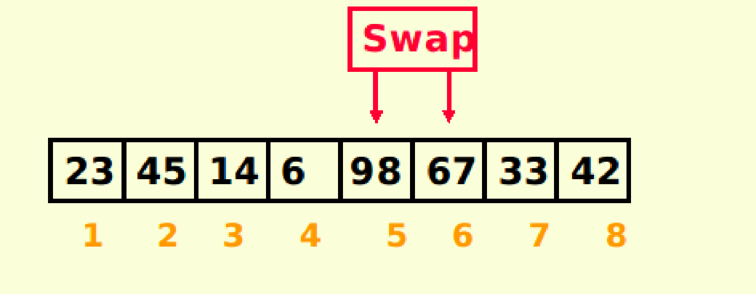
# Bubble Sort Example:



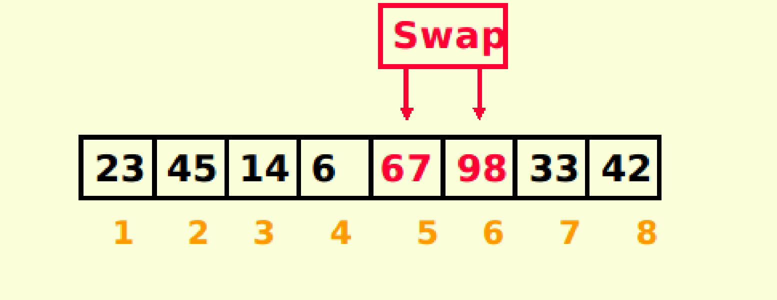
# Bubble Sort Example:



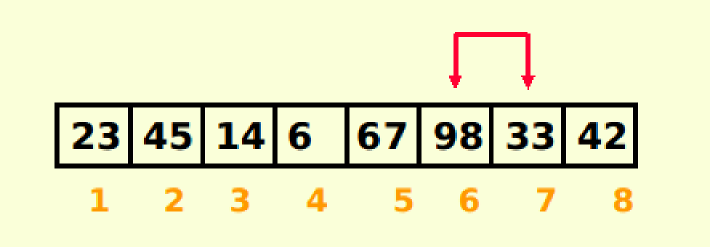
# Bubble Sort Example:



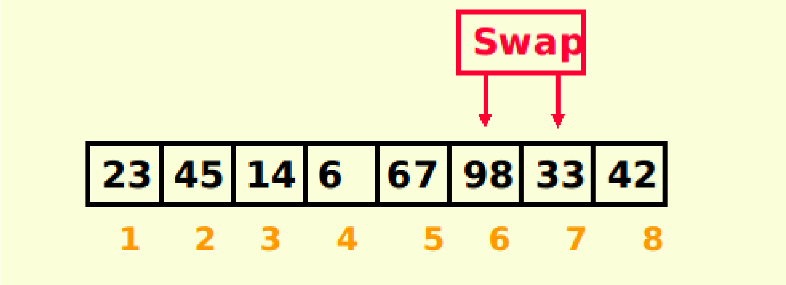
# Bubble Sort Example:



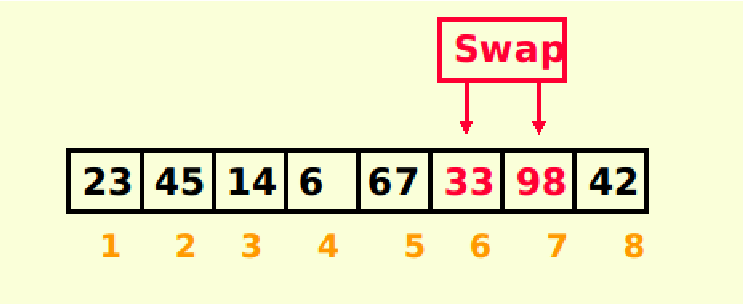
# Bubble Sort Example:



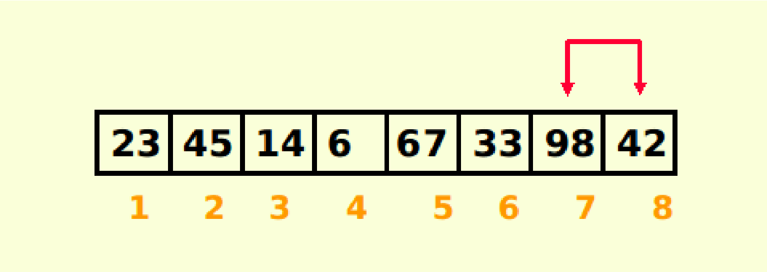
# Bubble Sort Example:

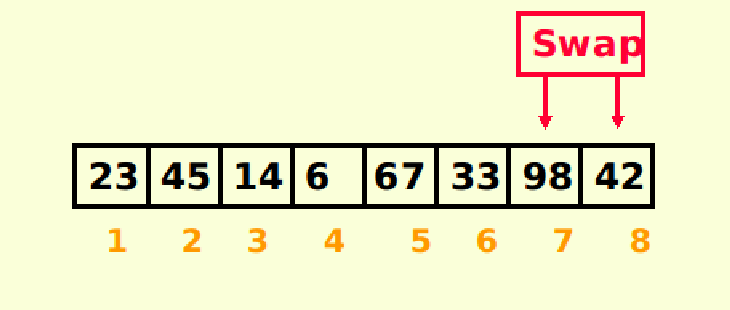


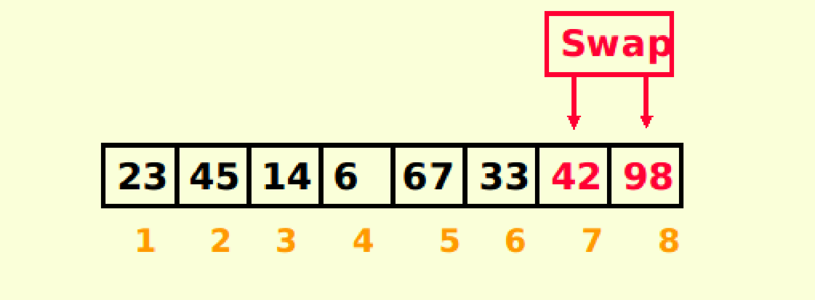
# Bubble Sort Example:



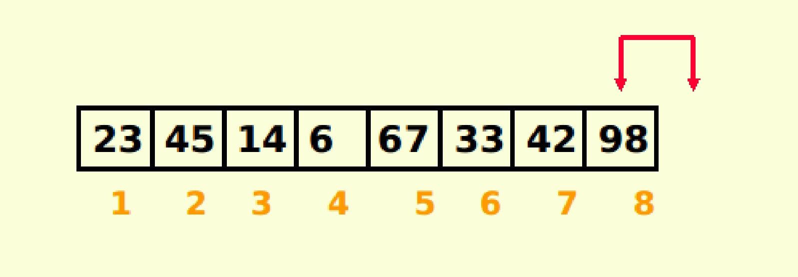
# Bubble Sort Example:



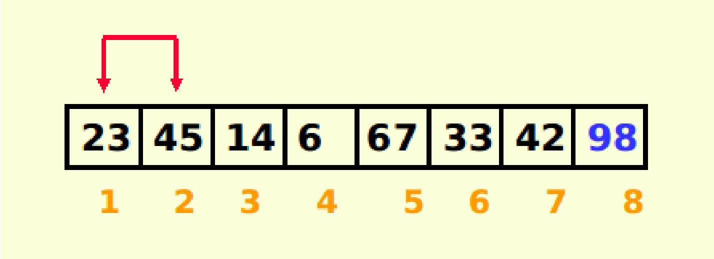

Swap
23
45
14
6
67
33
98
42
1
2
3
4
5
6
7
8


Swap
23
45
14
6
67
33
42
98
1
2
3
4
5
6
7
8

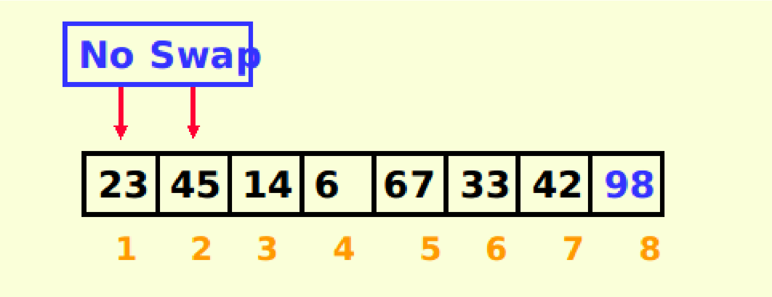
# Bubble Sort Example:



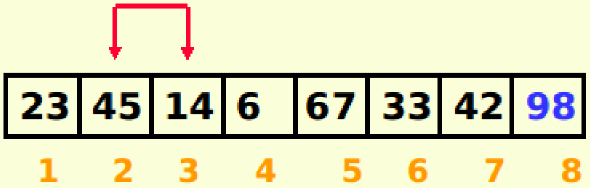
# Bubble Sort Example:

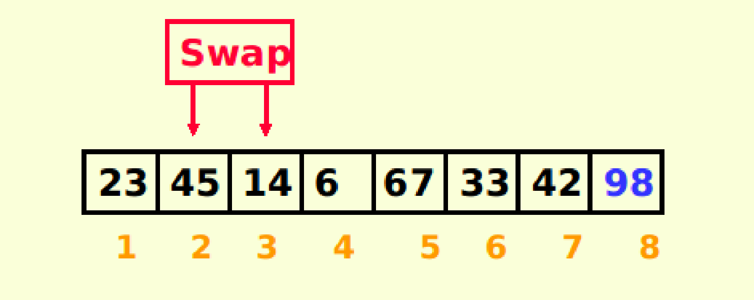


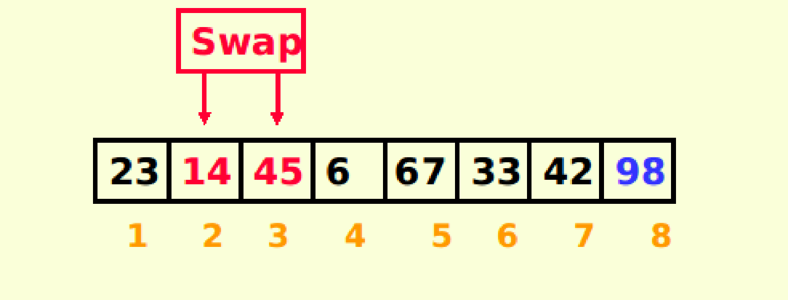
# Bubble Sort Example:



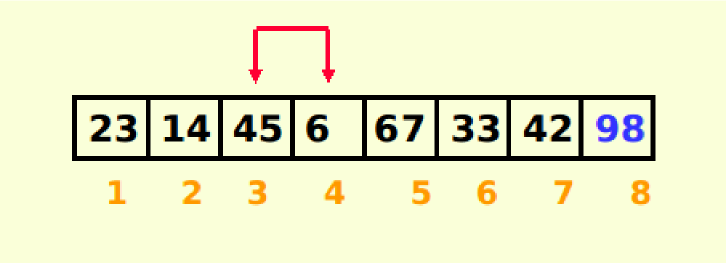
# Bubble Sort Example:



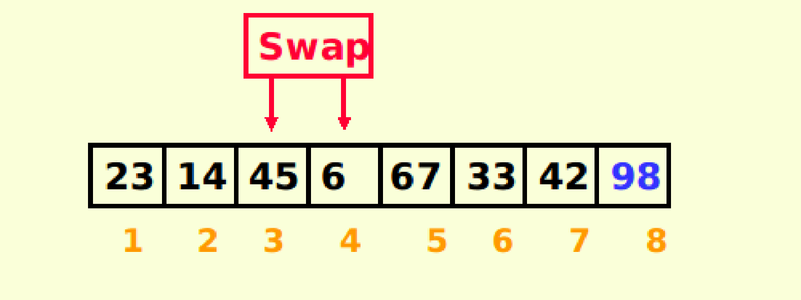

Swap
23
45
14
6
67
33
42
98
1
2
3
4
5
6
7
8


Swap
23
14
45
6
67
33
42
98
1
2
3
4
5
6
7
8

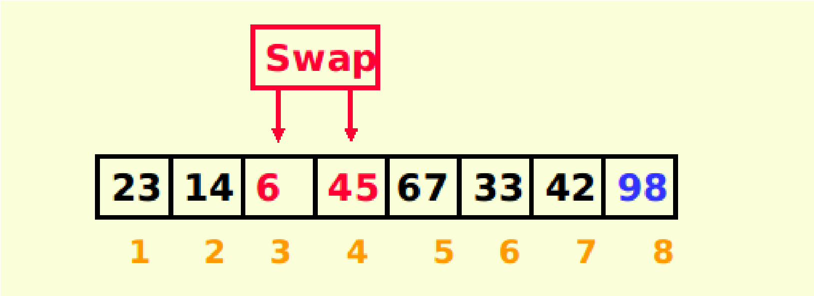
# Bubble Sort Example:



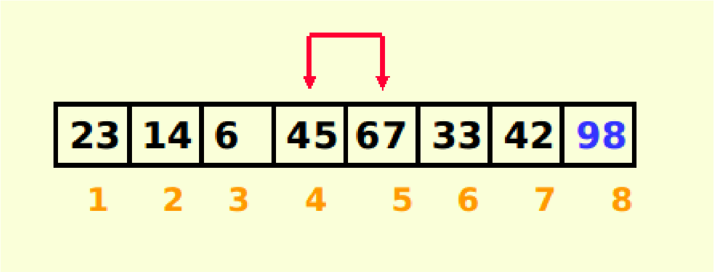
# Bubble Sort Example:



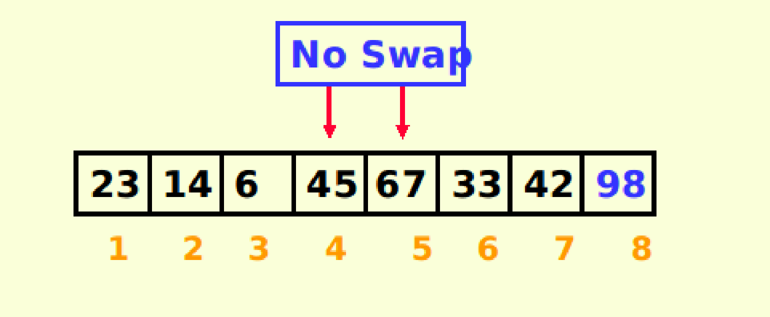
# Bubble Sort Example:



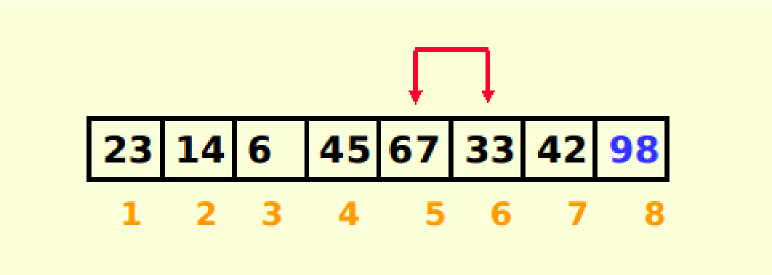
# Bubble Sort Example:



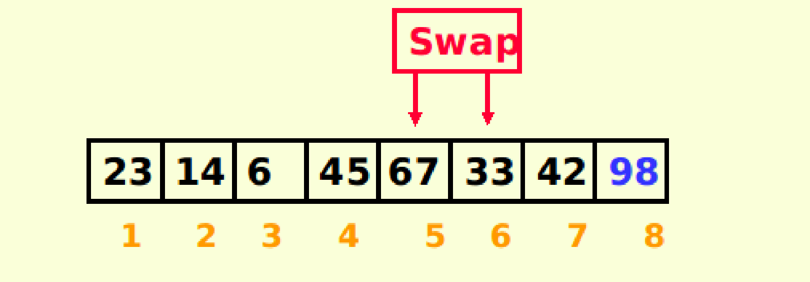
# Bubble Sort Example:



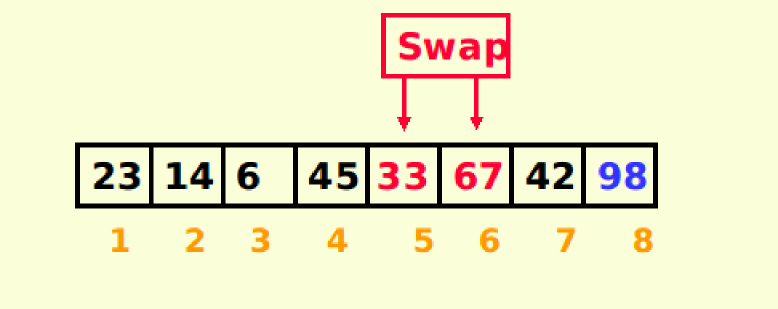
# Bubble Sort Example:



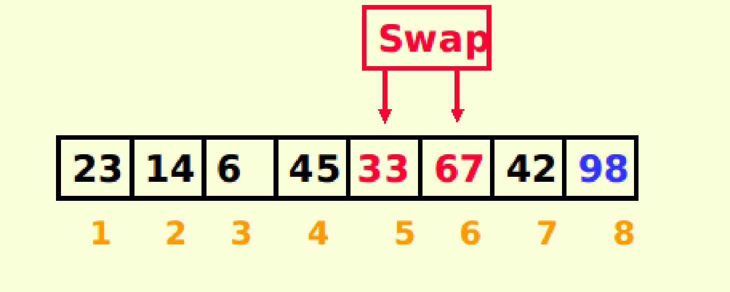
# Bubble Sort Example:



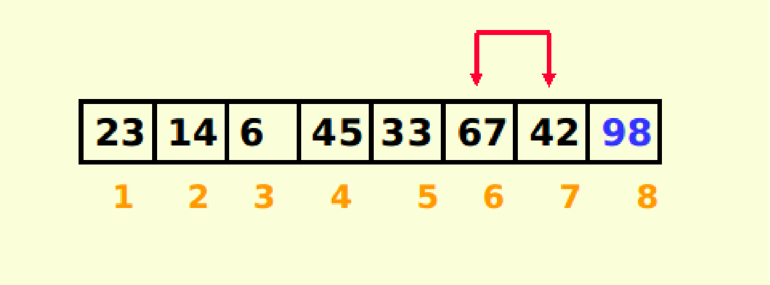
# Bubble Sort Example:



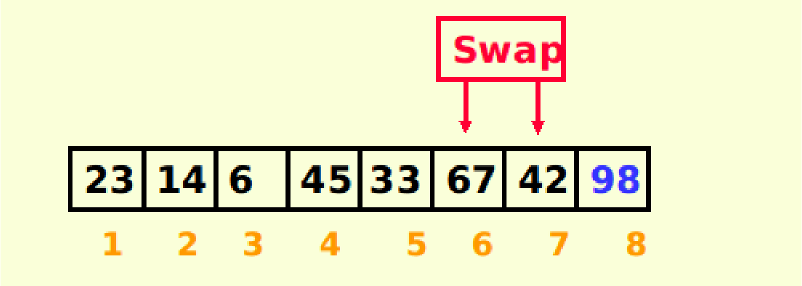
# Bubble Sort Example:



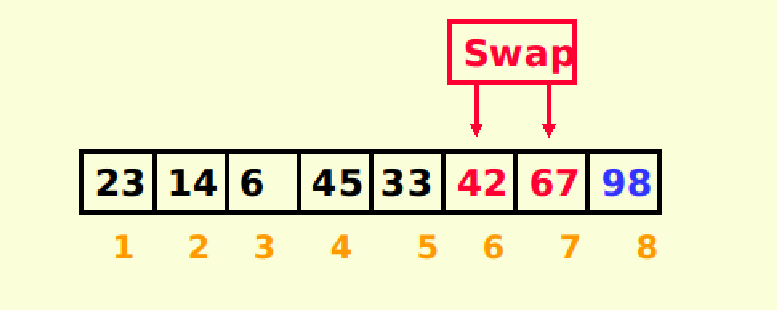
# Bubble Sort Example:



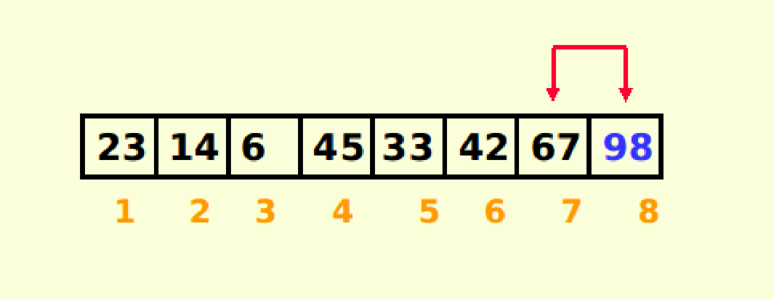
# Bubble Sort Example:

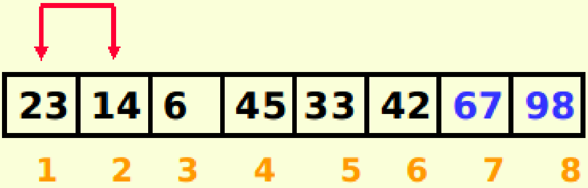


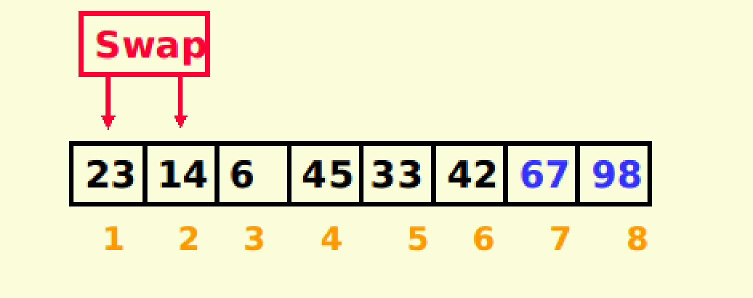
# Bubble Sort Example:

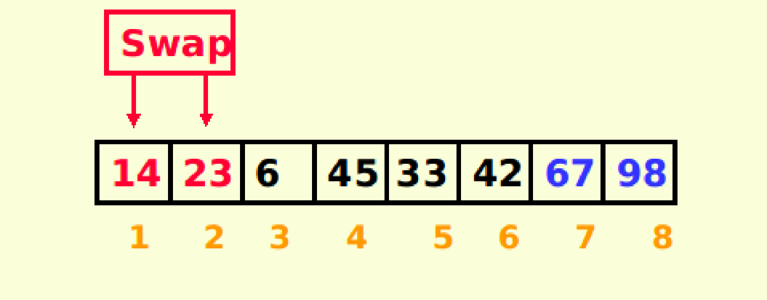


# Bubble Sort Example:

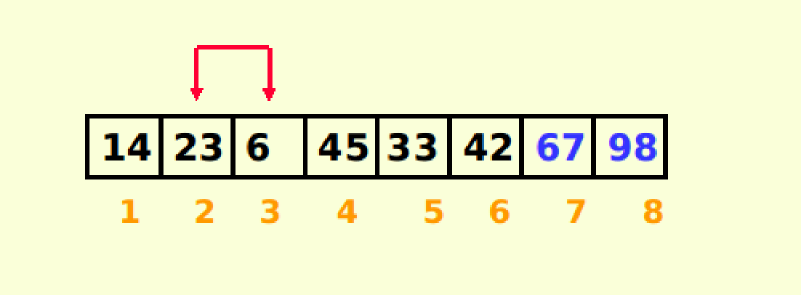



23
14
6
45
33
42
67
98
1
2
3
4
5
6
7
8

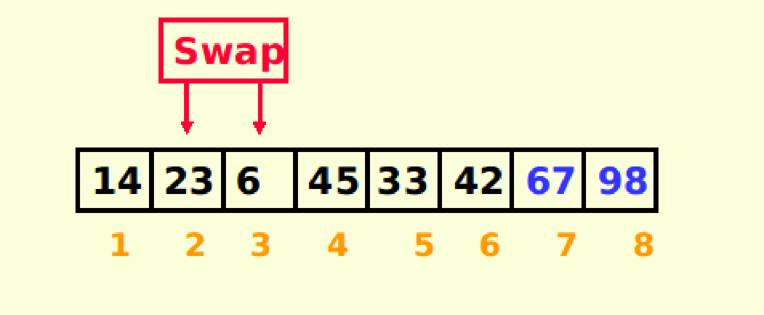

Swap
23
14
6
45
33
42
67
98
1
2
3
4
5
6
7
8

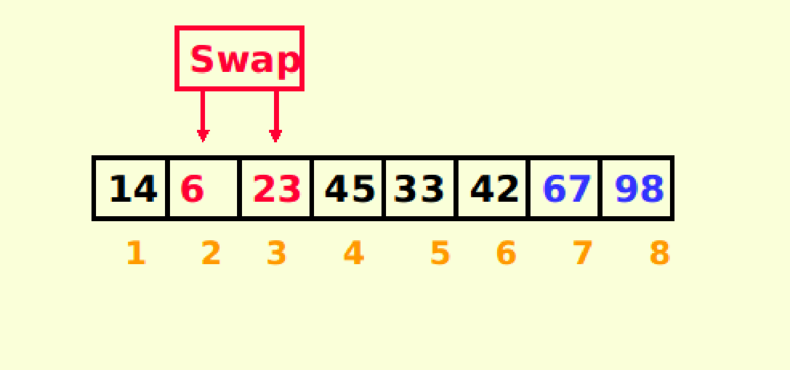

Swap
14
23
6
45
33
42
67
98
1
2
3
4
5
6
7
8

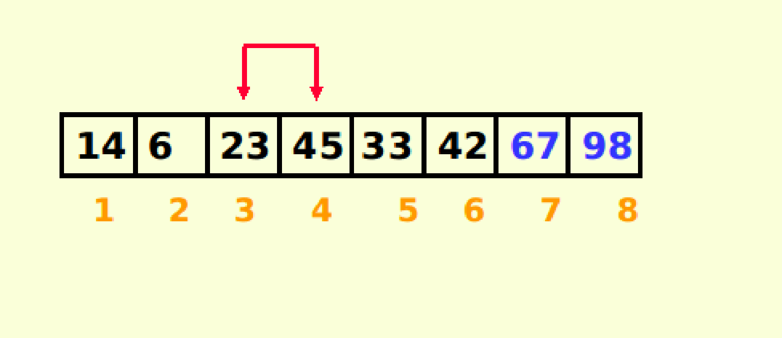
# Bubble Sort Example:

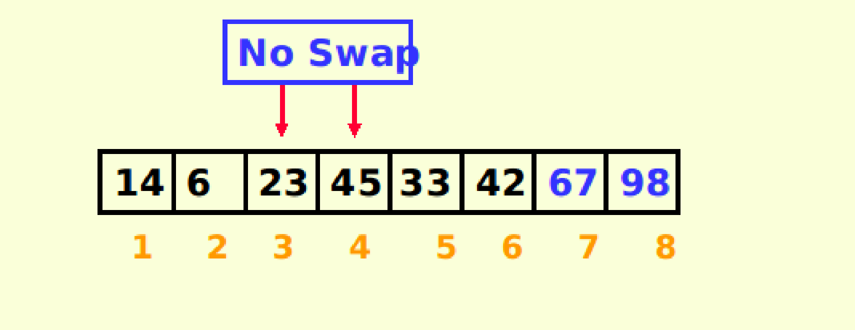


# Bubble Sort Example:

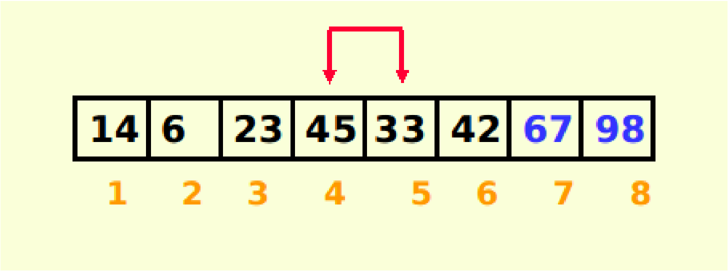



Swap
14
6
23
45
33
42
67
98
1
2
3
4
5
6
7
8

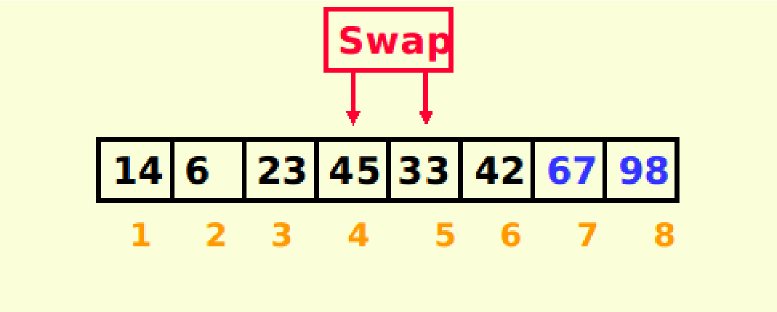

14
6
23
45
33
42
67
98
1
2
3
4
5
6
7
8


No Swap
14
6
23
45
33
42
67
98
1
2
3
4
5
6
7
8

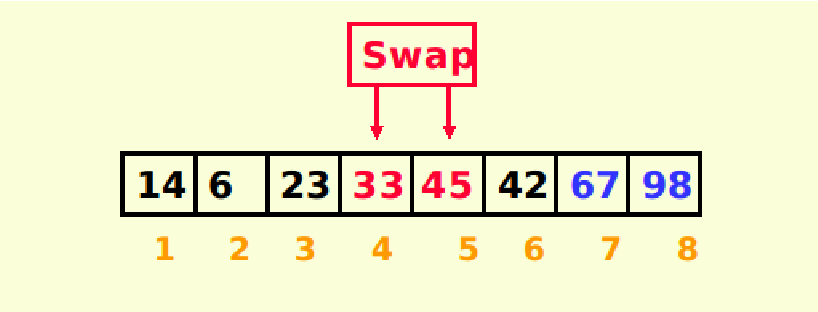
# Bubble Sort Example:



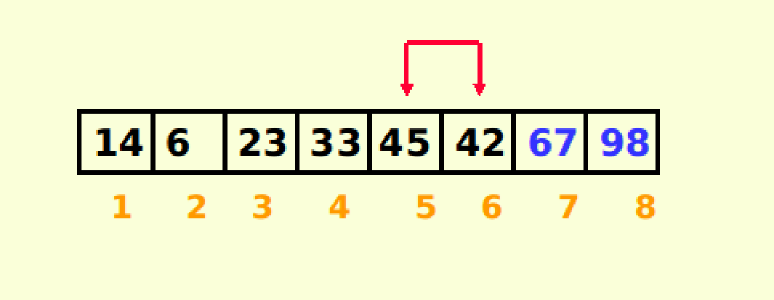
# Bubble Sort Example:



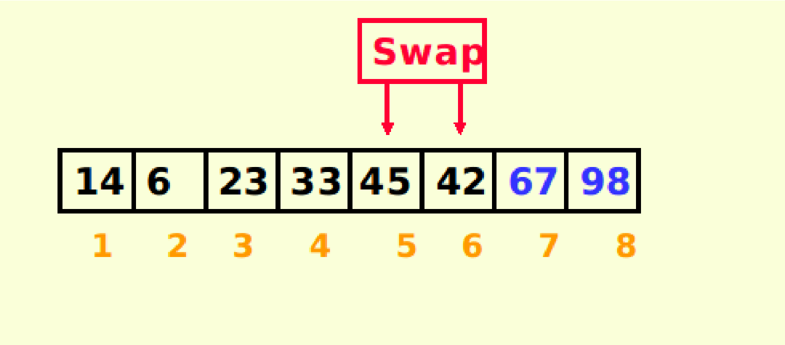
# Bubble Sort Example:



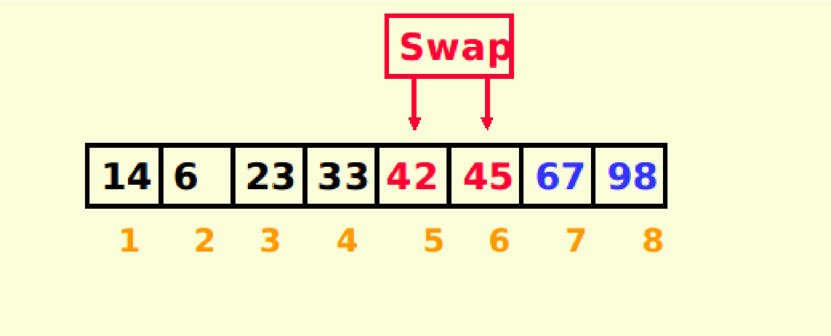
# Bubble Sort Example:



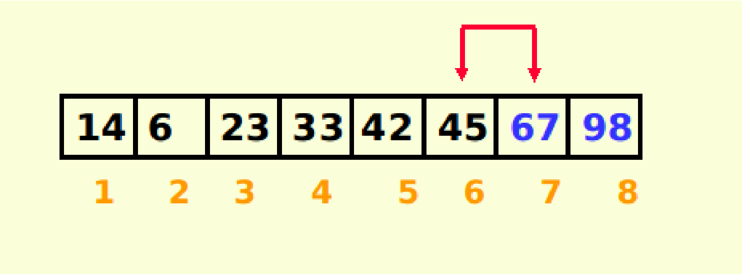
# Bubble Sort Example:



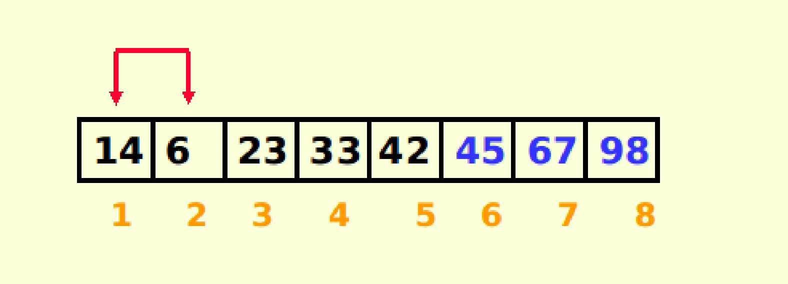
# Bubble Sort Example:

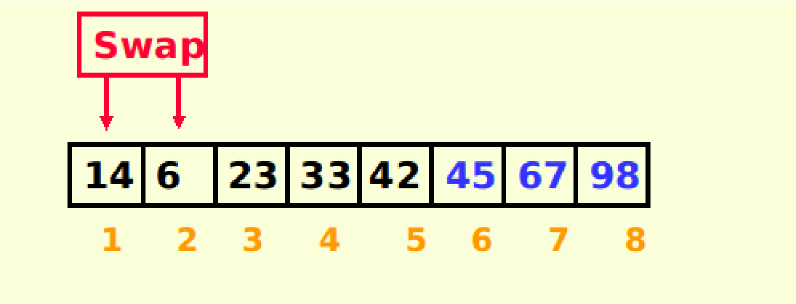


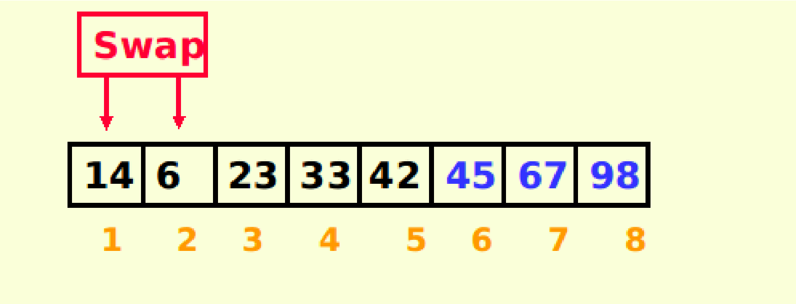
# Bubble Sort Example:



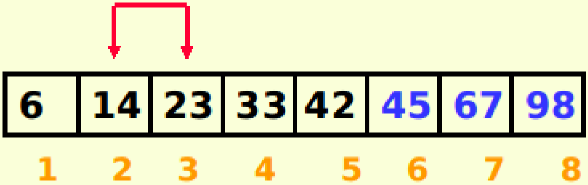
# Bubble Sort Example:

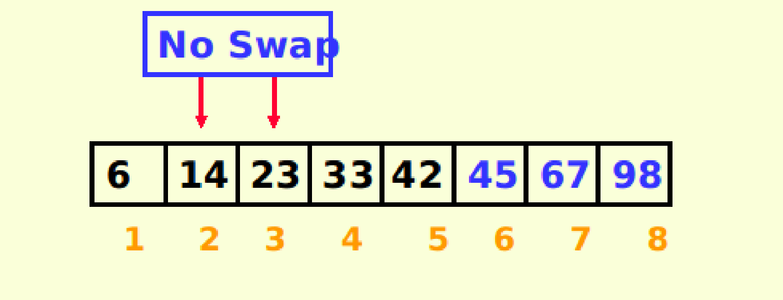


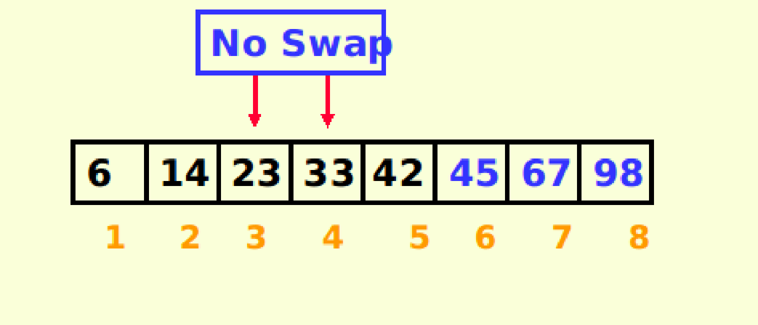

Swap
14
6
23
33
42
45
67
98
1
2
3
4
5
6
7
8


Swap
14
6
23
33
42
45
67
98
1
2
3
4
5
6
7
8

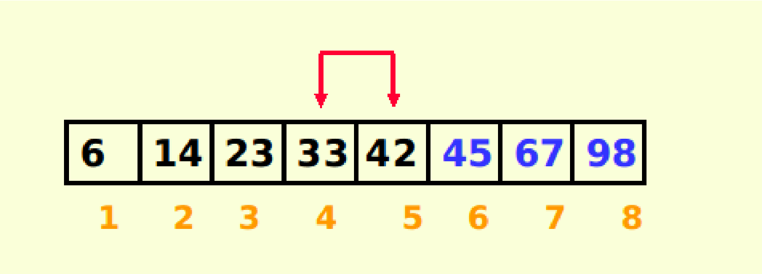
# Bubble Sort Example:



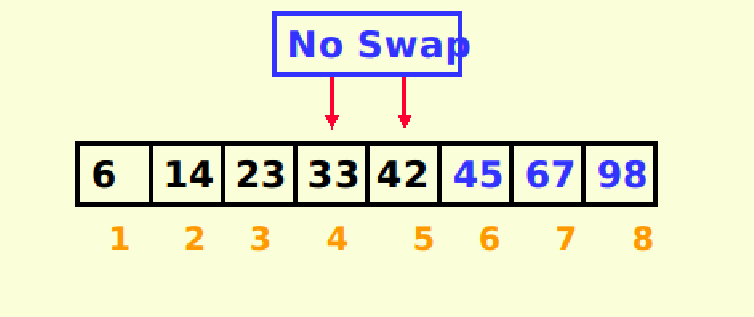

No Swap
6
14
23
33
42
45
67
98
1
2
3
4
5
6
7
8


No Swap
6
14
23
33
42
45
67
98
1
2
3
4
5
6
7
8

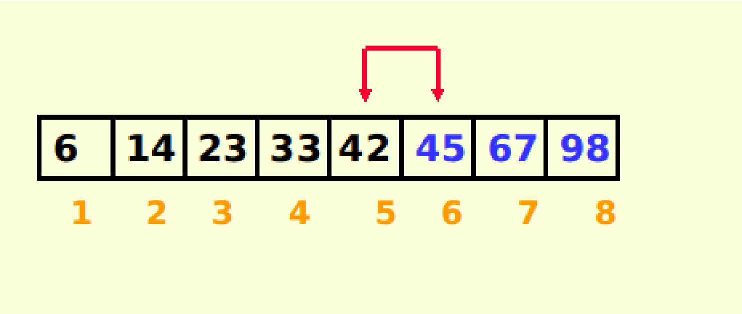
# Bubble Sort Example:

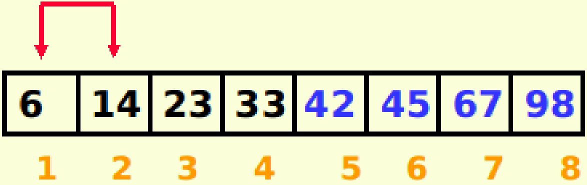


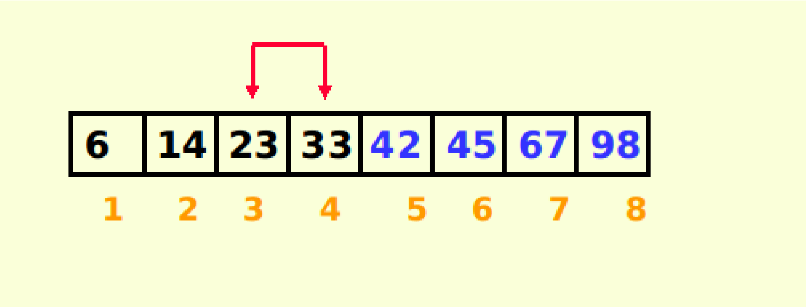
# Bubble Sort Example:

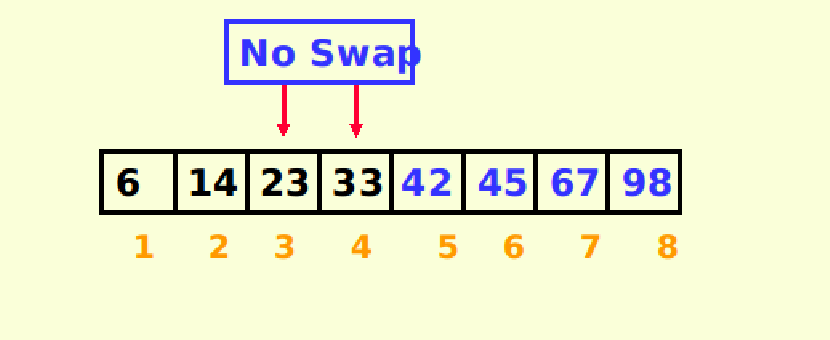


# Bubble Sort Example:

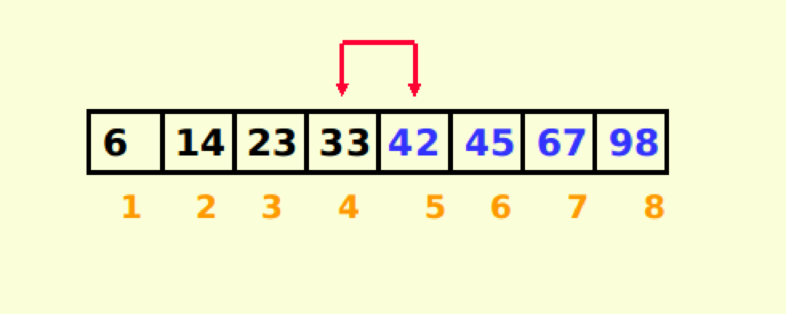



6
14
23
33
42
45
67
98
1
2
3
4
5
6
7
8


6
14
23
33
42
45
67
98
1 2 3 4 5 6 7 8


No Swap
6
14
23
33
42
45
67
98
1
2
3
4
5
6
7
8

# Bubble Sort Example:



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 6 | 14 | 23 | 33 | 42 | 45 | 67 | 98 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

- Algorithm for Bubble Sort:

**Algorithm** BubbleSort(A,n)

```
{
for i ← 0 to n − 2 do
    for j ← 0 to n − 2 − i do
        if (A[j + 1] < A[j]) then
            {
            temp ← A[j]
            A[j] ← A[j + 1]
            A[j + 1] ← temp
            }
        end if
    end for
end for
}
```

## Python Code for Bubble sort

### Python code

```
n = int(input("Enter the no of elements "))
k = []
for p in range (0,n):
    k.append(int(input("Enter a element::")))
for p in range(0,n-1):
    for l in range(0,n-1-p):
        if (k[l] > k[l+1]):
            temp = k[l]
            k[l]=k[l+1]
            k[l+1]=temp

for p in range(0,n):
            print(k[p])
```

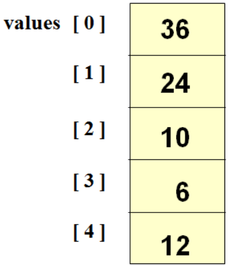
- Selection sort is among the simplest of sorting techniques.
- This Selection Sort works well for small data.
- Section sort is a good choice for sorting files with very large objects (records) and small keys.
- We can also first find the largest in the list and swap with the last position of the list.
- Then Second largest element and exchange it with the element in the second largest position. $\rightarrow$ Repeat this Process.

- Algorithm for Selection Sort

**Algorithm** Selection(A,n)

```
{
for i ← 0 to n − 2 do
    min ← i;
    for j ← i + 1 to n − 1 do
        if (A[j] < A[min]) then
            min ← j;
        end if
    end for
end for
swap A[i] and A[min];
}
```

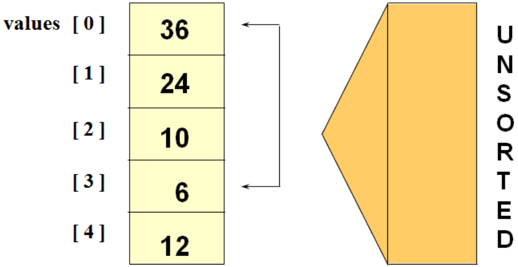
- Divides the array into two parts: already sorted, and not yet sorted.
- On each pass, finds the smallest of the unsorted elements, and swaps it into its correct place, thereby increasing the number of sorted elements by one.



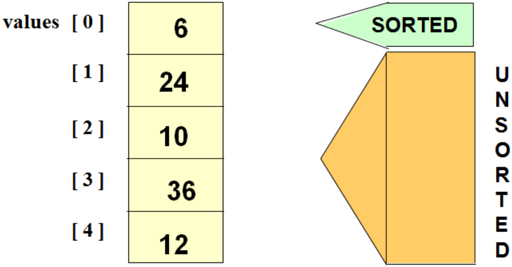
Selection Sort python function

```
def selectionSort(lyst):
    i = 0
    while i < len(lyst) - 1:         # Do n - 1 searches
        minIndex = i                 # for the smallest
        j = i + 1
        while j < len(lyst):         # Start a search
            if lyst[j] < lyst[minIndex]:
                minIndex = j
            j += 1
        if minIndex != i:            # Exchange if needed
            swap(lyst, minIndex, i)
        i += 1
```

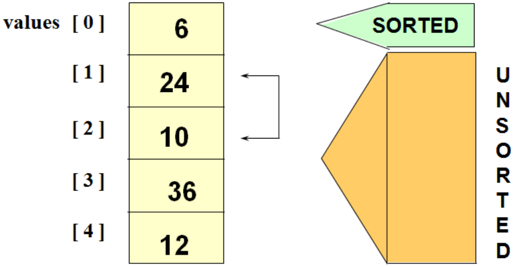
## Selection Sort: Pass One



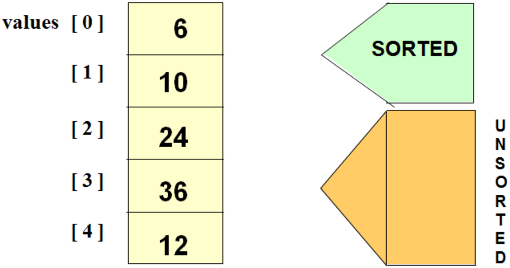
## Selection Sort: End Pass One



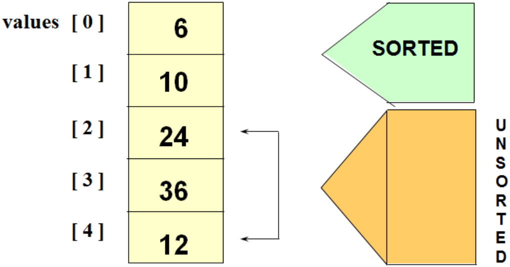
## Selection Sort: Pass Two



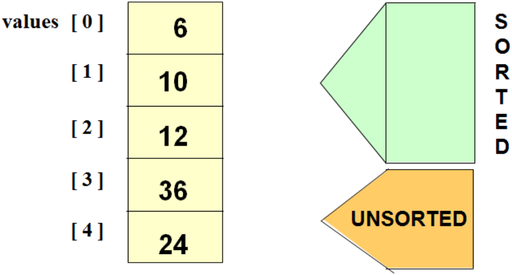
## Selection Sort: End Pass Two



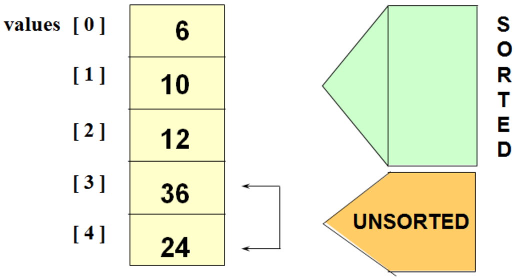
## Selection Sort: Pass Three



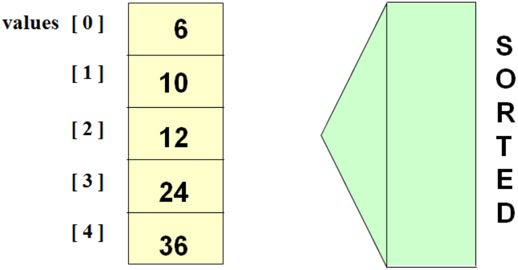
## Selection Sort: End pass Three



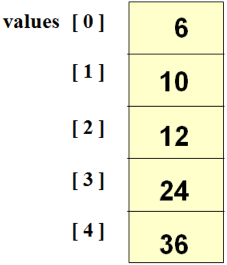
## Selection Sort: Pass Four



## Selection Sort: End Pass Four



## Selection Sort: How many comparisons?




4 comparisons for **values[0]**

3 comparisons for **values[1]**

2 comparisons for **values[2]**

1 comparison for **values[3]**

**= 4 + 3 + 2 + 1**

## A sample Python code for Selection Sort

```
def minIndex(i,l):
    minI = i
    for j in range(i+1,len(l)):
        if l[minI]>l[j]:
            minI = j
    return minI
def selection(l):
    n = len(l)
    for i in range(0,n):
        j = minIndex(i,l)
        if i!=j:
            l[i],l[j] = l[j],l[i]
    return l
```

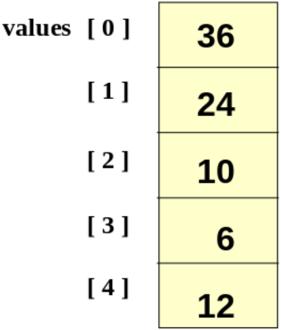
```
n = int(input())
l = []
for i in range(0,n):
    e = int(input())
    l.append(e)
l = selection(l)
print(l)
```

## Insertion Sort in Real Life

- Have you ever seen a teacher alphabetizing a couple dozen papers?
- She takes a paper from an unsorted collection and place into a sorted collection in order
- While playing cards, to sort the cards in the hand
- We extract a card, shift the remaining cards and insert the extracted card in correct place

## Insertion Sort

- Insert, one by one, each unsorted array element into its proper place.
- On each pass, this causes the number of already sorted elements to increase by one.

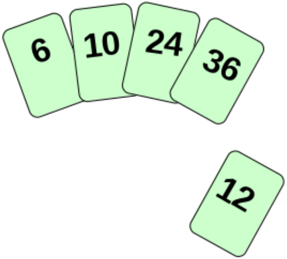


## Insertion Sort Python function

```
def insertionSort(lyst):
    i = 1
    while i < len(lyst):
        itemToInsert = lyst[i]
        j = i - 1
        while j >= 0:
            if itemToInsert < lyst[j]:
                lyst[j + 1] = lyst[j]
                j -= 1
            else:
                break
        lyst[j + 1] = itemToInsert
        i += 1
```

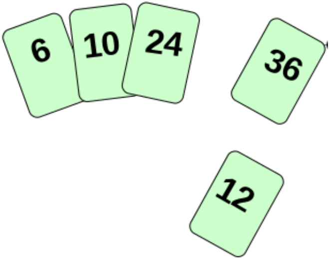
## Insertion Sort

- Works like someone who "inserts" one more card at a time into a hand of cards that are already sorted.
- To insert 12, we need to make room for it by moving first 36 and then 24.



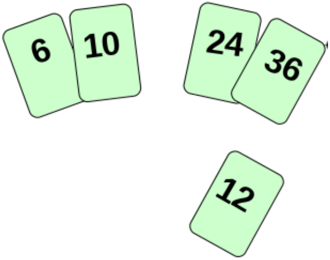
## Insertion Sort

- Works like someone who “inserts” one more card at a time into a hand of cards that are already sorted.
- To insert 12, we need to make room for it by moving first 36 and then 24.



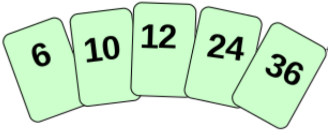
## Insertion Sort

- Works like someone who "inserts" one more card at a time into a hand of cards that are already sorted.
- To insert 12, we need to make room for it by moving first 36 and then 24.



**Insertion Sort**

- Works like someone who "inserts" one more card at a time into a hand of cards that are already sorted.
- To insert 12, we need to make room for it by moving first 36 and then 24.



**A sample Python code for Insertion Sort**

```
def insertion_sort(items):
    """ Implementation of insertion sort """
    for i in range(1, len(items)):
        j = i
        while j > 0 and items[j] > items[j-1]:
            items[j], items[j-1] = items[j-1], items[j]
            j -= 1
```

## Exercise 1

- Given a sorted list with an unsorted number V in the rightmost cell, can you write some simple code to insert V into the array so that it remains sorted? Print the array every time a value is shifted in the array until the array is fully sorted.
- Guideline: You can copy the value of V to a variable and consider its cell "empty". Since this leaves an extra cell empty on the right, you can shift everything over until V can be inserted.

## Exercise 2

- Using the same approach as exercise 1, sort an entire unsorted array?
- Guideline: You already can place an element into a sorted array. How can you use that code to build up a sorted array, one element at a time? Note that in the first step, when you consider an element with just the first element - that is already "sorted" since there's nothing to its left that is smaller.
- In this challenge, don't print every time you move an element. Instead, print the array after each iteration of the insertion-sort, i.e., whenever the next element is placed at its correct position.
- Since the array composed of just the first element is already "sorted", begin printing from the second element and on.


Thank you