Introduction to Computing and Programming

Searching & Sorting

Searching

i/p : An array of n-element and find element x

o/p: return the position of x if x is found in the given array

else return -1.

i/p: A[10, 20, 30, 1, 2, 3, 11, 21, 31]

o/p: x = 11?

x = 10?

x = 50?

o(1)

Linear Search

array

size of array

element

```
Linear_Search(a, n, x)
{
        for(i = 0; i ≤ n; i++)
        {
                if(a[i] == x)
                {
                        return i;
                }
        }
        return -1;
}
```

arr[n]

$O(n)$

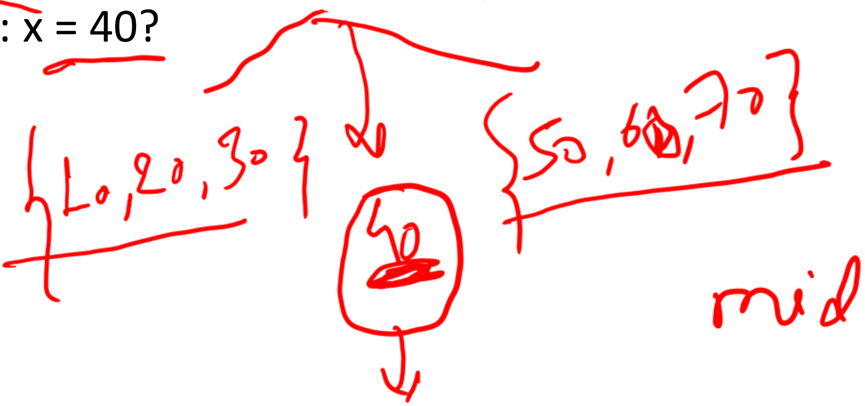
Binary Search

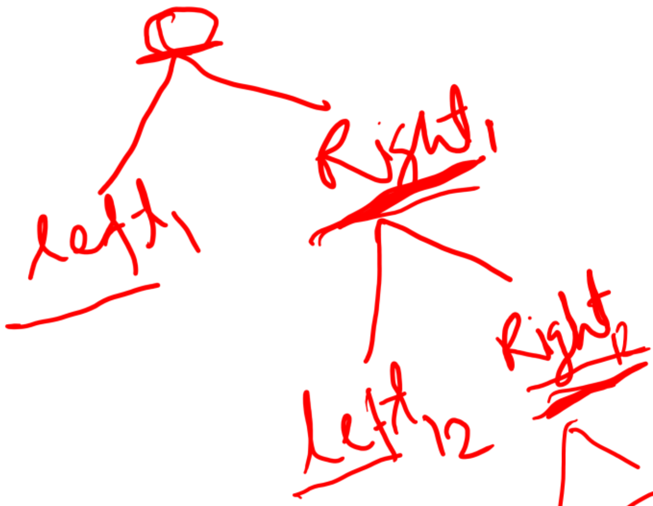
i/p : A **sorted array** of n-elements and find element x

o/p: return position of x if x is found
    else return -1.

i/p: A[10, 20, 30, 40, 50, 60, 70]

o/p: x = 40?





$$mid = \frac{l+R}{2} = \frac{0+6}{2} = 3$$

$$O(\log n)$$

Binary Search

array start end element

1 3 5 9 11

```
BinarySearch(a, i, j, x)
{
        int mid;
        if(i == j)
        {
                if(a[i] == x)

        return i;
                else

        return -1;
        }
}
```

3

11

```
else
{
        mid = (i+j)/2;
        if(a[mid] == x)
                return mid;
        else
        if(a[mid] > x)
                BinarySearch(a, i, mid-1, x);
        else
                BinarySearch(a, mid+1, j, x);
}
```

Sorting Algorithms

① insertion
② Selection
③ Bubble

Definition

Sorting is the process of:

- Taking a list of objects which could be stored in a linear order

$$(a_0, a_1, \ldots, a_{n-1})$$

e.*g*., numbers, and returning an reordering

$$(a'_0, a'_1, \ldots, a'_{n-1})$$

such that

$$a'_0 \le a'_1 \le \cdots \le a'_{n-1}$$

The conversion of an **unsorted objects** into **sorted objects.**

Insertion sort

Background

For example, consider this sorted array containing of eight sorted entries

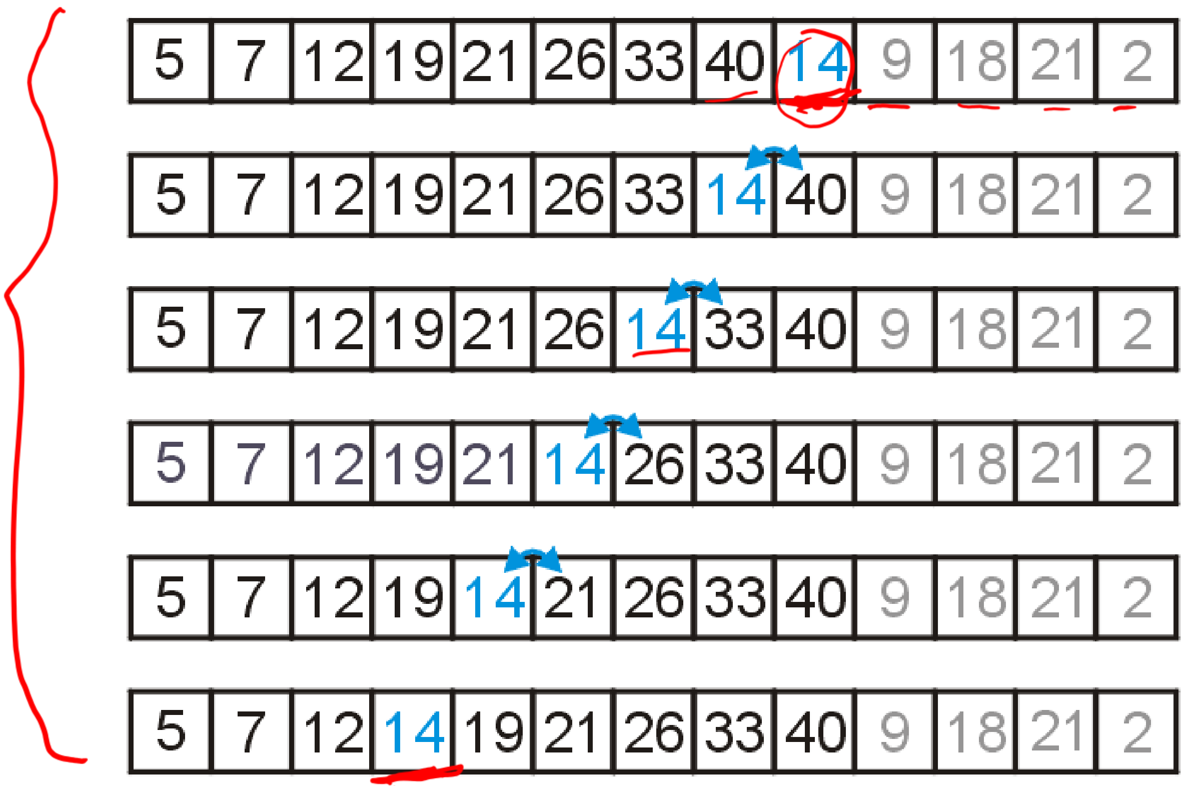
							14	40				
5	**7**	**12**	**19**	**21**	**26**	**33**	~~**40**~~	~~14~~	9	18	21	2

Suppose we want to insert 14 into this array leaving the resulting array sorted

Background

Starting at the back, if the number is greater than 14, copy it to the right

- Once an entry less than 14 is found, insert 14 into the resulting vacancy



5	7	12	19	21	26	33	40	14	9	18	21	2
5	7	12	19	21	26	33	14	40	9	18	21	2
5	7	12	19	21	26	14	33	40	9	18	21	2
5	7	12	19	21	14	26	33	40	9	18	21	2
5	7	12	19	14	21	26	33	40	9	18	21	2
5	7	12	14	19	21	26	33	40	9	18	21	2

$O(n)$

$O(n^2)$

$O(n*n)=O(n^2)$

The Algorithm

For any unsorted list:

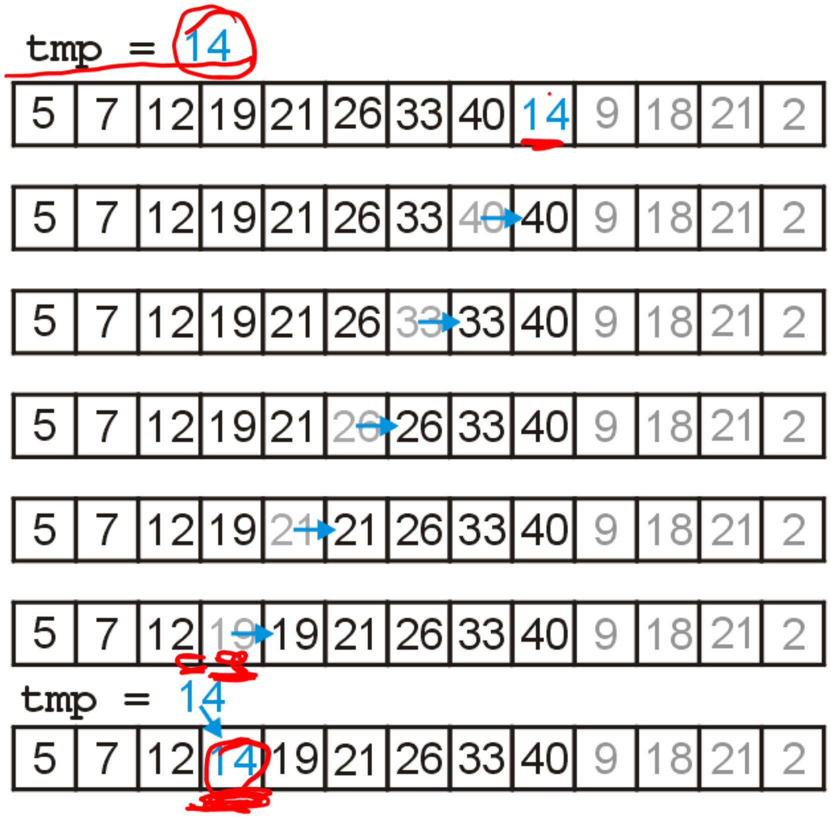
- Treat the **first element** as a sorted list of size 1

Then, given a sorted list of size $k - 1$

- Insert the k^{th} item in the sorted list
- The sorted list is now of size k

The Algorithm

Swapping is expensive, so we could just temporarily assign the new entry



tmp = 14

5 7 12 19 21 26 33 40 14 9 18 21 2

5 7 12 19 21 26 33 40 40 9 18 21 2

5 7 12 19 21 26 33 33 40 9 18 21 2

5 7 12 19 21 26 26 33 40 9 18 21 2

5 7 12 19 21 21 26 33 40 9 18 21 2

5 7 12 19 19 21 26 33 40 9 18 21 2

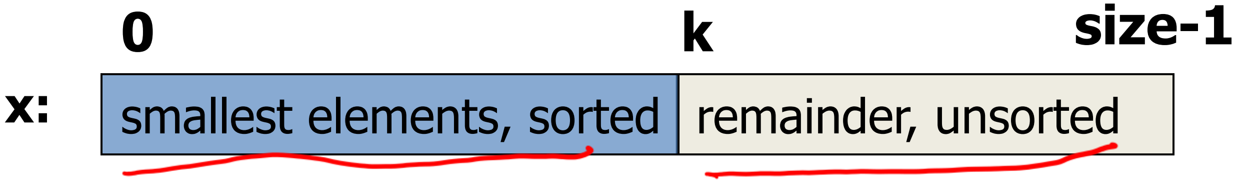
tmp = 14

5 7 12 14 19 21 26 33 40 9 18 21 2

Selection sort

Selection Sort (min at first)

General situation :



0

k

size-1

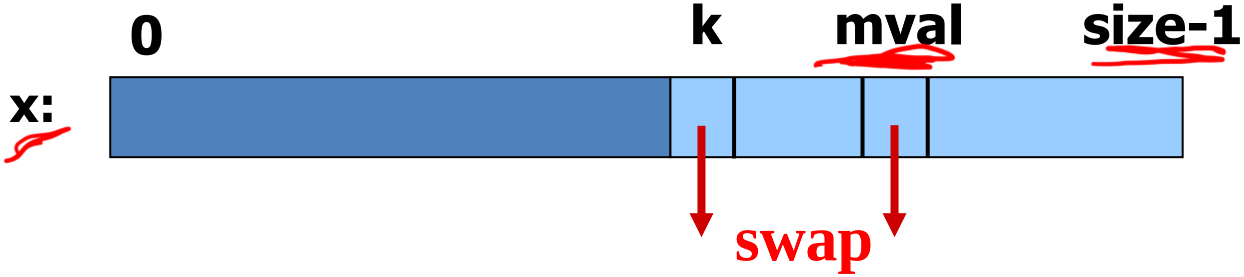
x:

smallest elements, sorted

remainder, unsorted

Steps :

- **Find smallest element, mval, in x[k…size-1]**
- **Swap smallest element with x[k], then increase k.**



0

k

mval

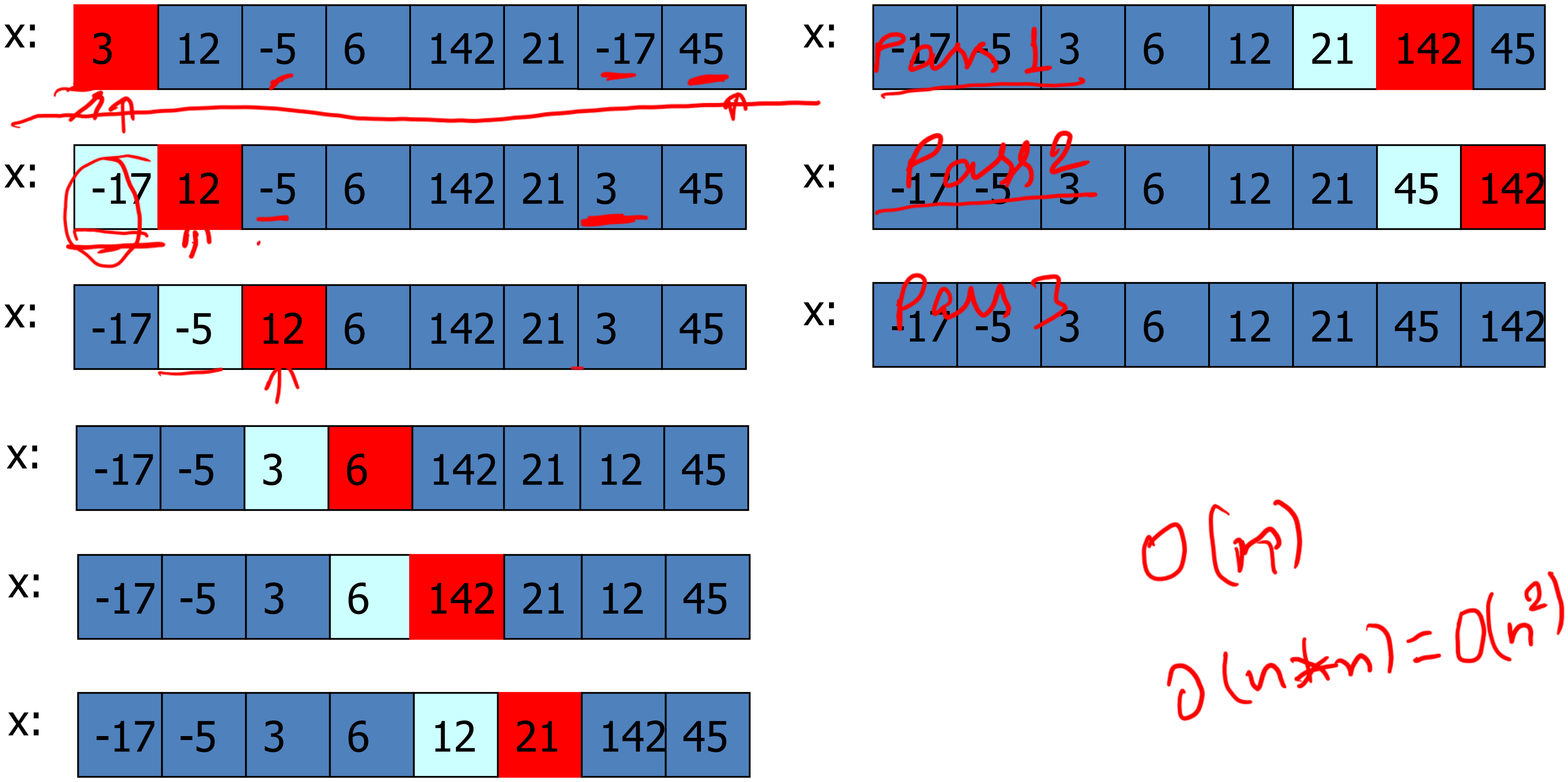
size-1

x:

swap

Selection Sort - Example

mval = ~~-5~~ -17



x:	3	12	-5	6	142	21	-17	45
x:	-17	12	-5	6	142	21	3	45
x:	-17	-5	12	6	142	21	3	45
x:	-17	-5	3	6	142	21	12	45
x:	-17	-5	3	6	142	21	12	45
x:	-17	-5	3	6	12	21	142	45

x:	-17	-5	3	6	12	21	142	45
x:	-17	-5	3	6	12	21	45	142
x:	-17	-5	3	6	12	21	45	142

Pass 1

Pass 2

Pass 3

O(n)

$O(n * n) = O(n^2)$

Bubble sort

Description

Suppose we have an array of data which is unsorted:

- Starting at the front, traverse the array, find the largest item, and move (or ***bubble***) it to the top

- With each subsequent iteration, find the **next largest** item and *bubble* it up towards the top of the array

Implementation

Starting with the first item, assume that it is the largest

Compare it with the second item:

- If the first is larger, swap the two,
- Otherwise, assume that the second item is the largest

Continue up the array, either swapping or redefining the largest item

Implementation

After one pass, the **largest** item must be the **last** in the array

Start at the front again:

- the second pass will bring the **second-largest** element into the **second-last position**

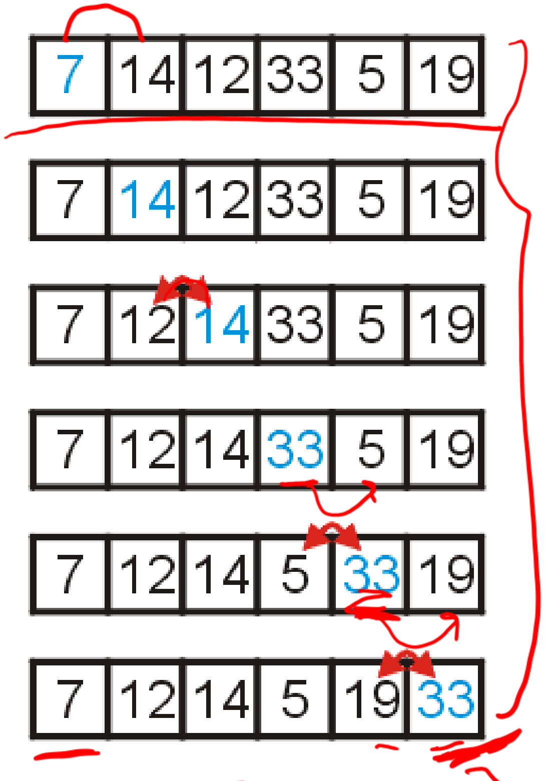
Repeat $n - 1$ times, after which, all entries will be sorted

Example

Consider the unsorted array to the right

We start with the element in the first location, and move forward:

- if the current and next items are in order, continue with the next item, otherwise
- swap the two entries



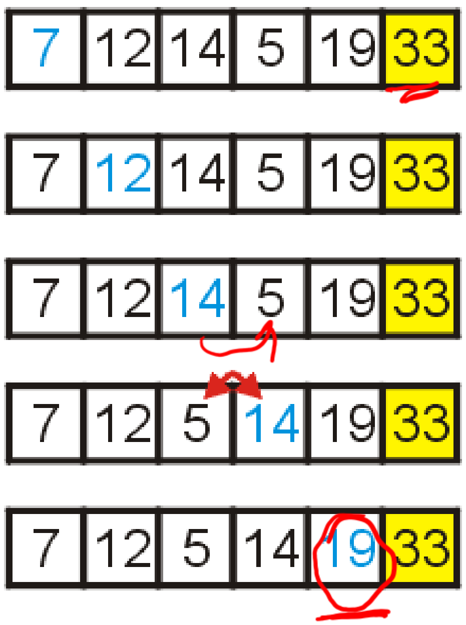
7	14	12	33	5	19
7	14	12	33	5	19
7	12	14	33	5	19
7	12	14	33	5	19
7	12	14	5	33	19
7	12	14	5	19	33

Pass 1

O(n)

Example

- After one loop, the largest element is in the last location
  - Repeat the procedure



7 12 14 5 19 33

7 12 14 5 19 33

7 12 14 5 19 33

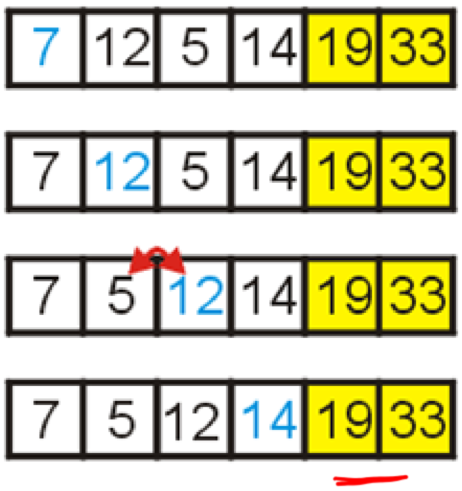
7 12 5 14 19 33

7 12 5 14 19 33

Example

Now the two largest elements are at the end

- Repeat again



7 12 5 14 19 33

7 12 5 14 19 33

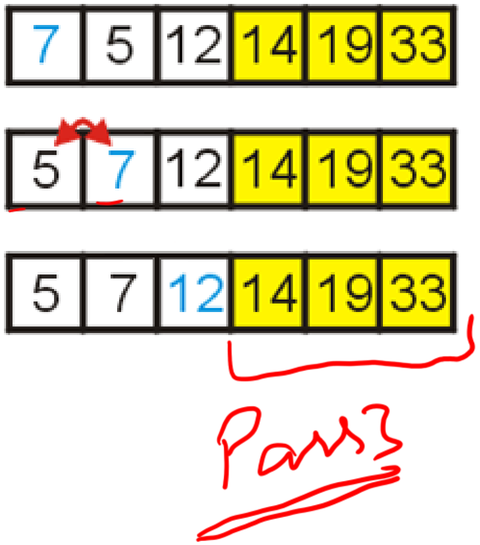
7 5 12 14 19 33

7 5 12 14 19 33

Pass 2

Example

With this loop, 5 and 7 are swapped



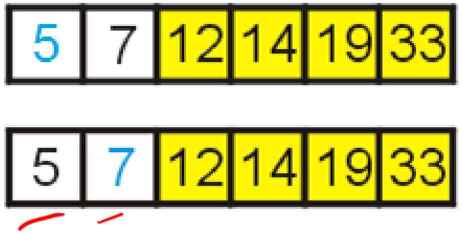
7 5 12 14 19 33

5 7 12 14 19 33

5 7 12 14 19 33

Example

At this point, we have a sorted array



5	7	12	14	19	33

5	7	12	14	19	33

$O(n^2)$

Improvements over Bubble Sort

The next few slides show a few improvements:

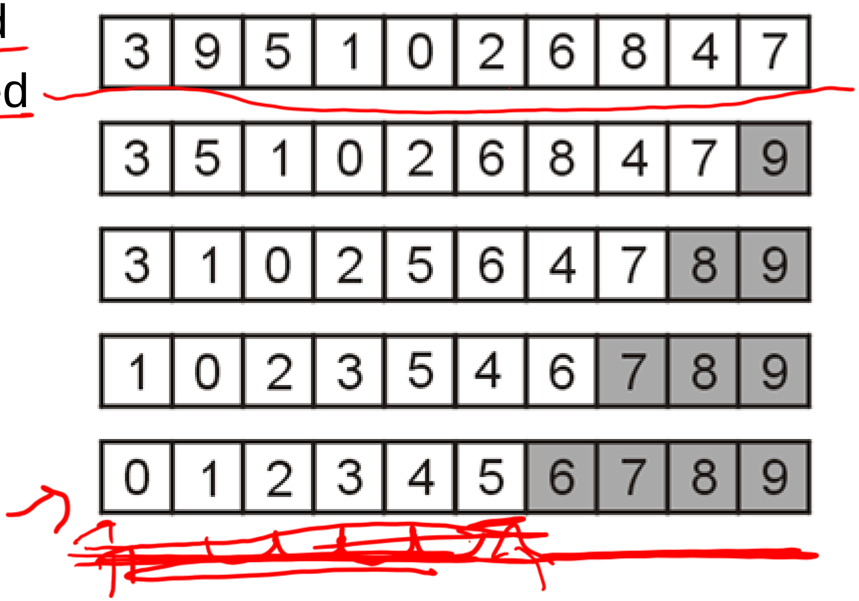
- reduce the number of swaps,
- halting if the list is sorted

Second Improvement: Flagged Bubble Sort

One useful modification would be to check if no swaps occur:

- If no swaps occur, the list is sorted
- In this example, no swaps occurred during the 5th pass

Use a **Boolean flag** to check if no swaps occurred



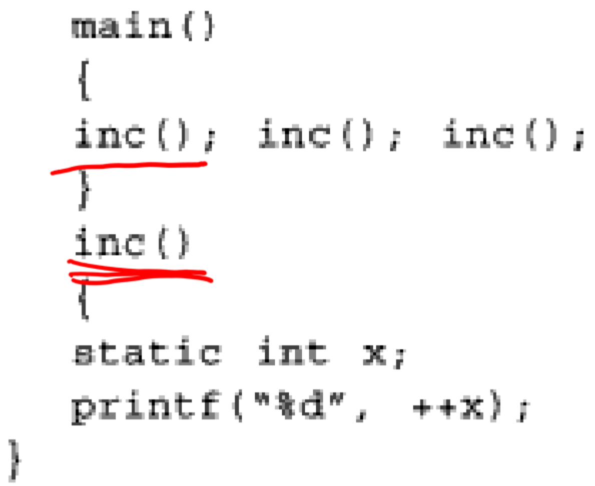
3	9	5	1	0	2	6	8	4	7
3	5	1	0	2	6	8	4	7	9
3	1	0	2	5	6	4	7	8	9
1	0	2	3	5	4	6	7	8	9
0	1	2	3	4	5	6	7	8	9

Surprise Quiz 2

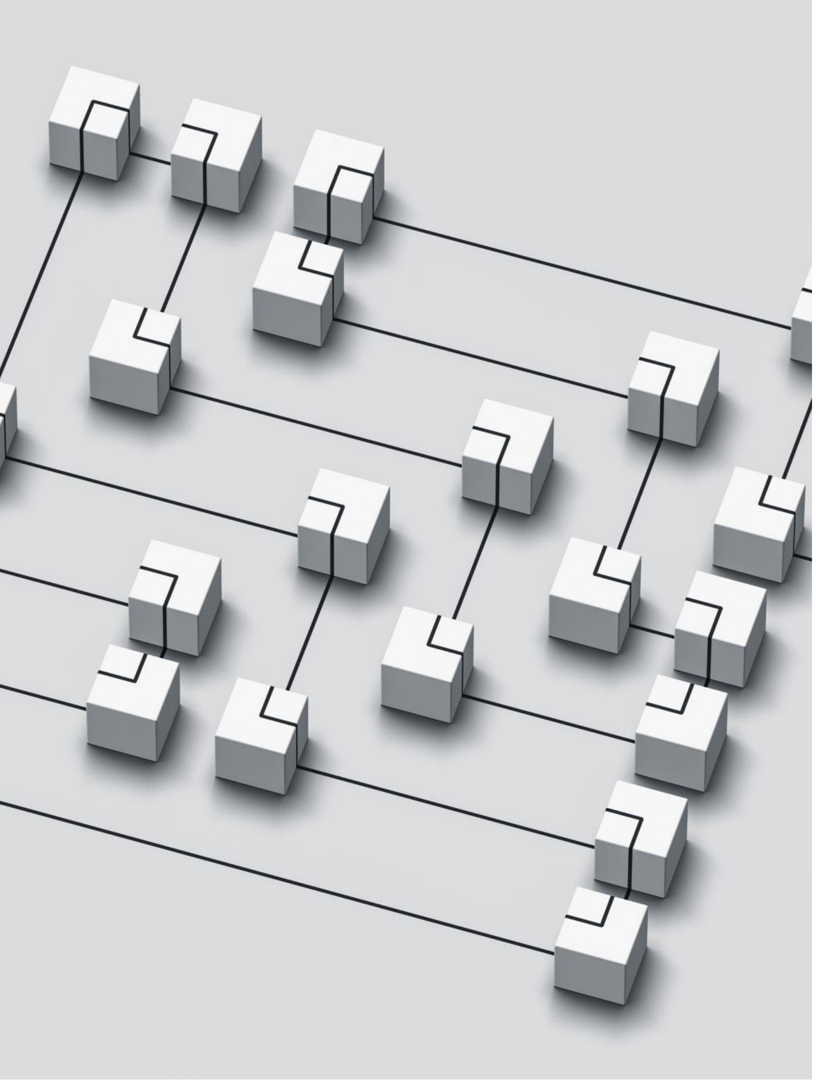
Q.1. Write a C program to Find Leap Year between 1900 to 2025?

Hint: A year divisible by 400 is a leap year.; A year divisible by 100 but not by 400 is not a leap year; A year divisible by 4 but not by 100 is a leap year; A year not divisible by 4 is not a leap year.

Q.2. What would be the output of the following program?



```
main()
{
inc(); inc(); inc();
}
inc()
{
static int x;
printf("%d", ++x);
}
```



Upcoming Slides

- **String**