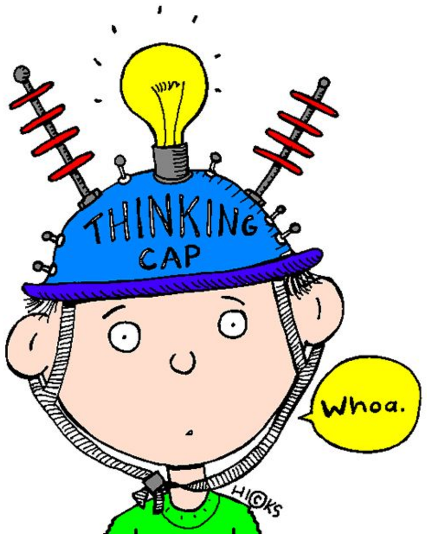
XCELERATOR

Transforming Education

Ice Breaker (5 mins)

If you were stranded on a desert island, what three items would you want to have with you?



Sorting Techniques-I

Concepts

- Sorting Techniques
  - Internal Sorting
    - Bubble Sort
    - Selection Sort
    - Insertion Sort
  - External Sorting

Questions for this session

We will answer the following questions in this session-

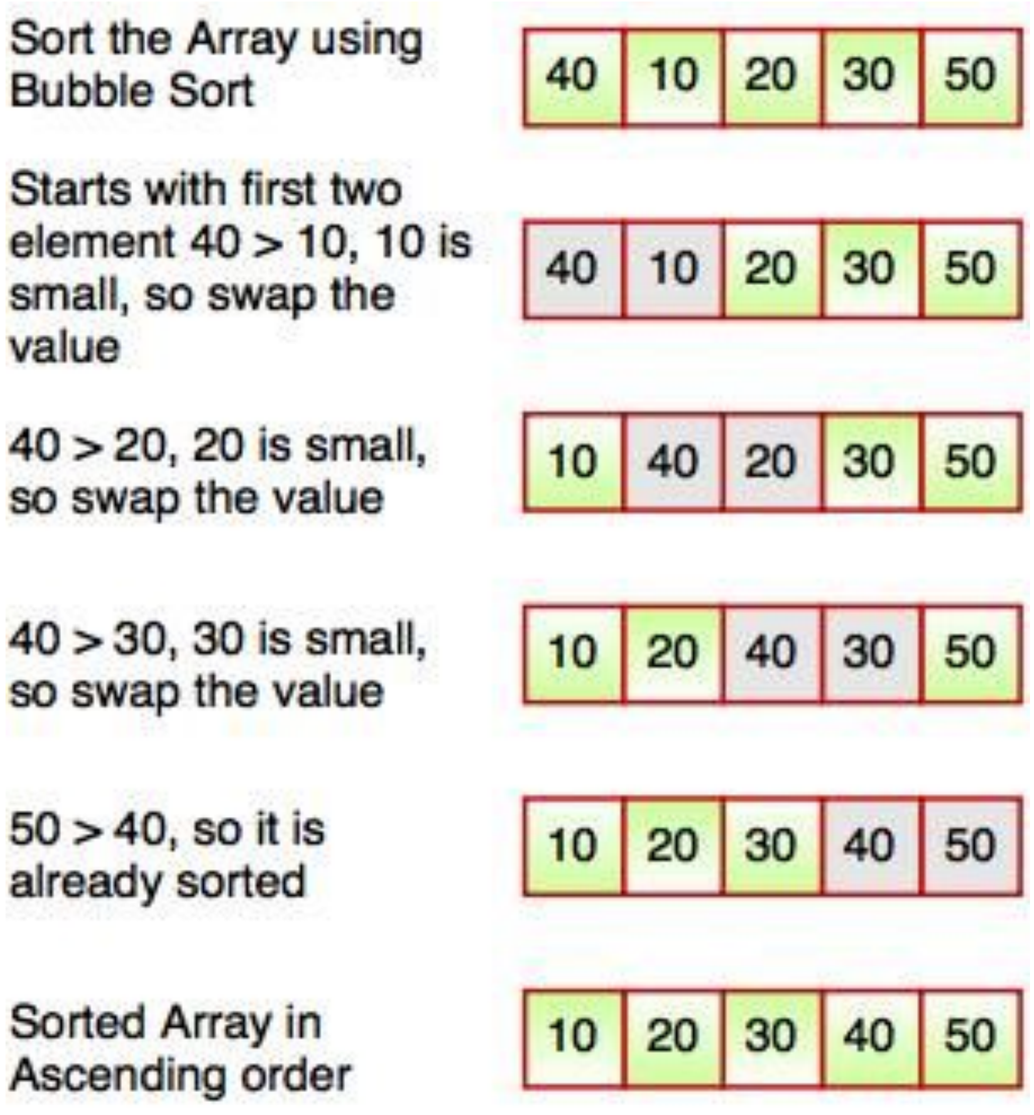
- How can we sort elements in a 1-D array?

Sorting Techniques

- **Internal Sorting Techniques**
  - Bubble Sort
  - Selection Sort
  - Insertion Sort
  - Quick Sort
  - Radix Sort
- **External Sorting Techniques**
  - Merge Sort

Bubble Sort

- Used for sorting small set of numbers
- Compares all the elements one by one and sorts them based on their values
- n-1 iterations/ passes are required for sorting n numbers



Sort the Array using Bubble Sort	40	10	20	30	50
Starts with first two element 40 > 10, 10 is small, so swap the value	40	10	20	30	50
40 > 20, 20 is small, so swap the value	10	40	20	30	50
40 > 30, 30 is small, so swap the value	10	20	40	30	50
50 > 40, so it is already sorted	10	20	30	40	50
Sorted Array in Ascending order	10	20	30	40	50

Fig. Working of Bubble Sort

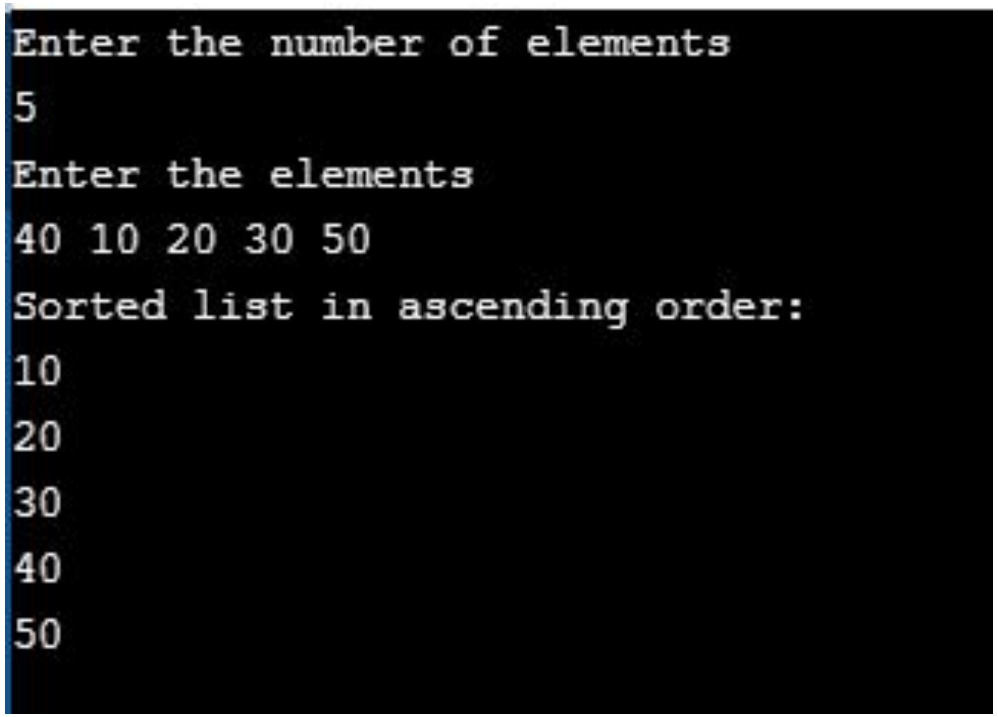
Algorithm/Pseudocode: Bubble Sort

```
Algorithm Bubble_Sort ( A, n) where A is a 1-D array of n elements to be sorted
for (i = 0 ; i < ( n - 1 ); i++)
 {
   for (j = 0 ; j < (n - i - 1); j++)
   {
     if (a[j] > a[j+1])
     {
      Swap a[j] and a[j+1]
     }
   }
 }
```

C Program: Bubble Sort

```
#include <stdio.h>
void bubble_sort(int[], int);
int main()   {
 int a[100], n, i;
 printf("Enter the number of elements\n");
 scanf("%d", &n);
 printf("Enter the elements\n");
 for (i = 0; i < n; i++)
   scanf("%d", &a[i]);
 bubble_sort(a, n);
 printf("Sorted list in ascending order:\n");
 for ( i = 0 ; i < n ; i++ )
    printf("%d\n", a[i]);
 return 0;    }
```

```
void bubble_sort(int a[], int n)
{
 int i, j, t;
 for (i = 0 ; i < ( n - 1 ); i++)
 {
   for (j = 0 ; j < (n - i - 1); j++)
   {
     if (a[j] > a[j+1])
     {
       t = a[j];
       a[j] = a[j+1];
       a[j+1] = t;
     } } } }
```



```
Enter the number of elements
5
Enter the elements
40 10 20 30 50
Sorted list in ascending order:
10
20
30
40
50
```

Complexity, Advantages and Disadvantage: Bubble Sort

Time Complexity of Bubble Sort-

- Best Case Complexity- **O(n^2)**
- Average Case Complexity- **O(n^2)**
- Worst Case Complexity- **O(n^2)**

Advantages-

- It is popular and easy to implement
- It sorts the numbers without using additional temporary storage

Disadvantage-

- This approach does not work for lists having large number of elements

Insertion Sort

- Simple sorting algorithm
- Sorts the elements by shifting them one at a time
- n-1 iterations/passes
- Sorting starts with the second element as the key
- Key compared with elements before it
- Key is then put in its appropriate location

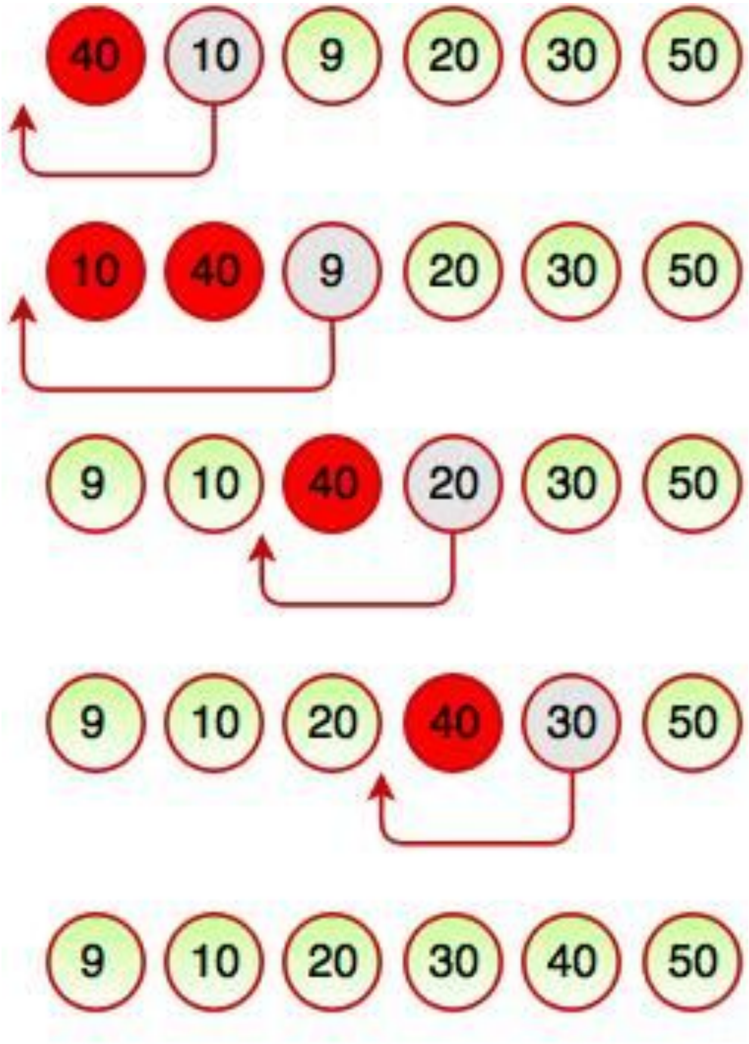


Fig. Working of Insertion Sort

Algorithm/Pseudocode: Insertion Sort

Algorithm **Insertion_Sort** (A, n) where A is a 1-D array of n elements to be sorted

```
 for (int i = 1 ; i <= n - 1; i++)
  {
     int j = i, t;
     while ( j > 0)
     {
         if(arr[j] < arr[j-1])
         {
           Swap arr[j] and arr[j-1]
         }
        j--;
     }
  }
```

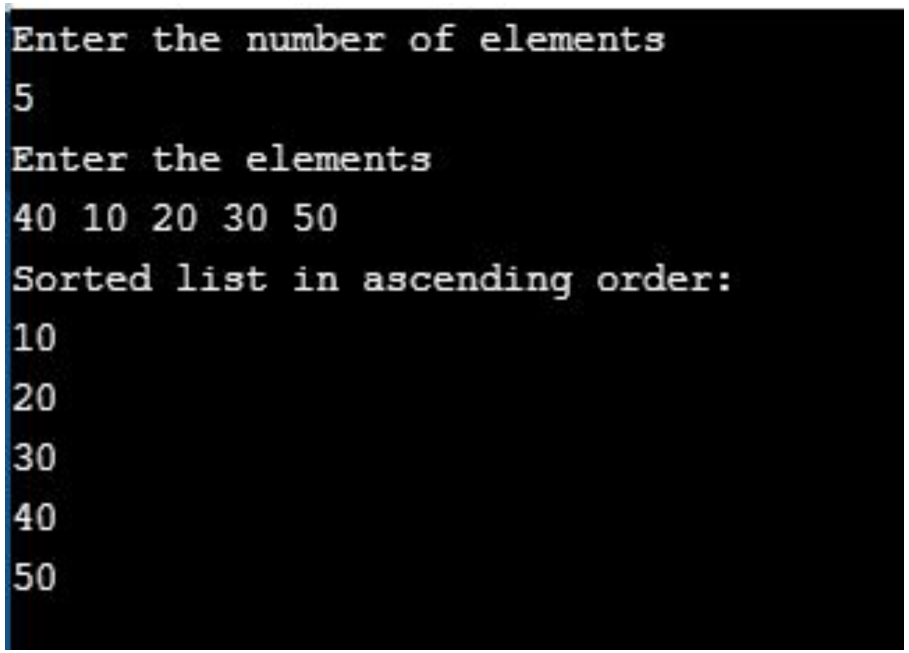
Identify the solution: Insertion Sort (5 mins)

Can you quickly write the C program for implementing Insertion Sort algorithm?

Solution: Insertion Sort: C Program

```
#include <stdio.h>
void insertion_sort(int[], int);
int main()   {
  int a[100], n, i;
  printf("Enter the number of elements\n");
  scanf("%d", &n);
  printf("Enter the elements\n");
  for (i = 0; i < n; i++)
    scanf("%d", &a[i]);
  insertion_sort(a, n);
  printf("Sorted list in ascending order:\n");
  for ( i = 0 ; i < n ; i++ )
     printf("%d\n", a[i]);
  return 0;    }
```

```
void insertion_sort(int a[], int n)
{
for (int i = 1 ; i <= n - 1; i++)
  {
     int j = i, t;
     while ( j > 0)
     {
         if(arr[j] < arr[j-1])
         {
            t = arr[j];
            arr[j]    = arr[j-1];
            arr[j-1] = t;
         }
       j--;   }  }  }
```



```
Enter the number of elements
5
Enter the elements
40 10 20 30 50
Sorted list in ascending order:
10
20
30
40
50
```

Complexity, Advantages and Disadvantages: Insertion Sort

Time Complexity of Bubble Sort-

- Best Case Complexity- **O(n)**
- Average Case Complexity- **O(n^2)**
- Worst Case Complexity- **O(n^2)**

Advantages-

- This algorithm has the simplest implementation
- It is stable and does not change the relative ordering of elements with equal values
- It is more efficient compared to Bubble and Selection Sort

Disadvantages-

- This algorithm works well for smaller data sets but is not suitable for larger data sets
- It requires additional memory space for sorting the elements

InClass Activity: Selection Sort

Instructions:

Activity Type- Exploration

Students can divide themselves into groups of 2

Time Allotted for this activity is 20 minutes

Question:

Develop and Analyze the algorithm/ C program for implementing Selection Sort technique

Solution: Selection Sort

- Simple sorting technique
- Uses n-1 iterations/passes for sorting n elements
- Works as follows-
  - 1st iteration- Replaces smallest array element with the element in 0th position/ index
  - 2nd iteration- Replaces second smallest array element with the element in 1st index/ position and so on
  - This continues till all elements are sorted

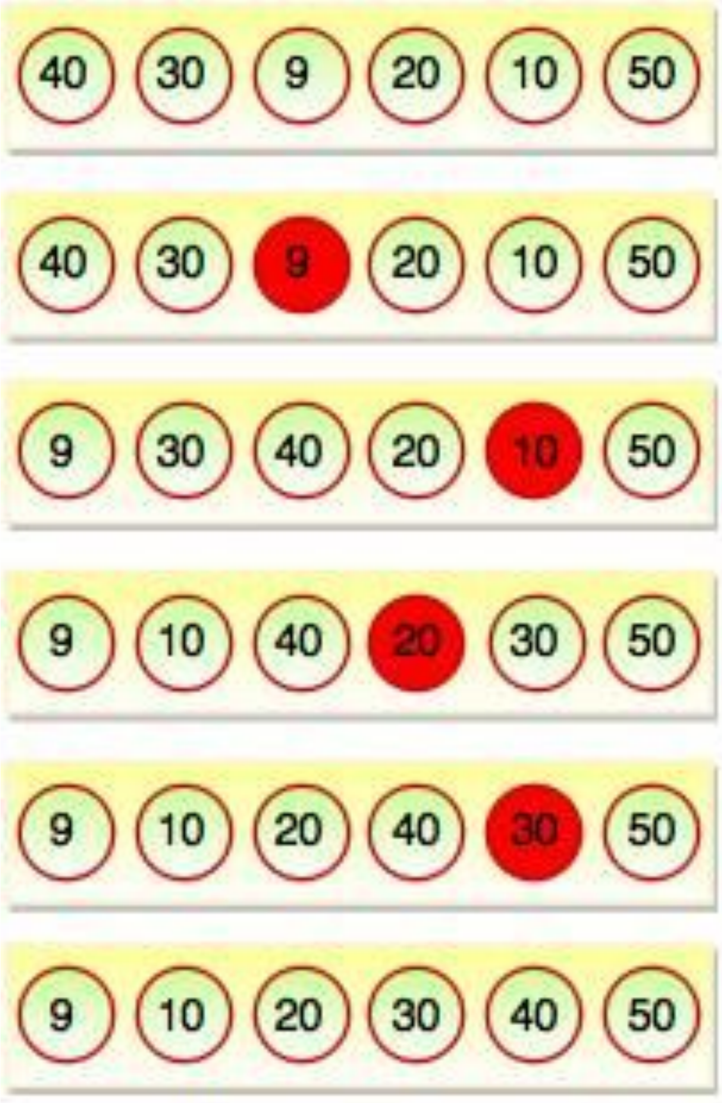


Fig. Working of Selection Sort

Solution: Algorithm of Selection Sort

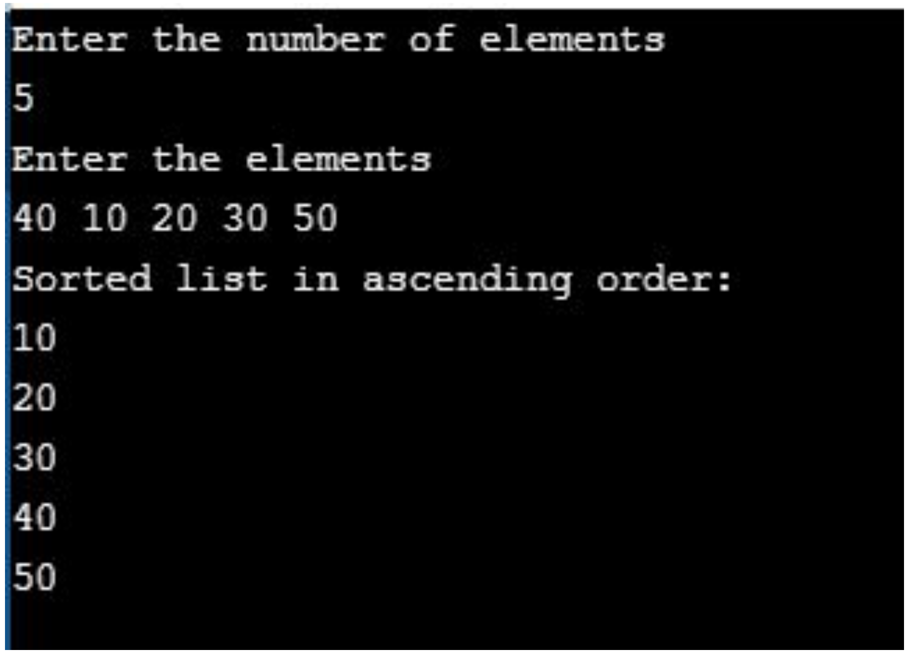
Algorithm **Selection_Sort** (A, n) where A is a 1-D array of n elements to be sorted

```
for (int i = 0 ; i < ( n - 1 ) ; i++ )
  {
    temp = i;
    for (j = i + 1 ; j < n ; j++ )
    {
      if ( arr[temp] > arr[j] )
        temp = j;
    }
    if ( temp != i )
    {
     Swap a[i] and a[temp]
     }
  }
```

Solution: C Program for Selection Sort

```
#include <stdio.h>
void selection_sort(int[], int);
int main()   {
  int a[100], n, i;
  printf("Enter the number of elements\n");
  scanf("%d", &n);
  printf("Enter the elements\n");
  for (i = 0; i < n; i++)
    scanf("%d", &a[i]);
  selection_sort(a, n);
  printf("Sorted list in ascending order:\n");
  for ( i = 0 ; i < n ; i++ )
     printf("%d\n", a[i]);
  return 0;    }
```

```
void selection_sort(int a[], int n)  {
  int temp,t,j;
   for (int i = 0 ; i < ( n - 1 ) ; i++ )  {
      temp = i;
      for (j = i + 1 ; j < n ; j++ ) {
         if ( arr[temp] > arr[j] )
            temp = j;
      }
      if ( temp != i ) {
         t = arr[i];
         arr[i] = arr[temp];
         arr[temp] = t;
      }  }  }
```



```
Enter the number of elements
5
Enter the elements
40 10 20 30 50
Sorted list in ascending order:
10
20
30
40
50
```

Solution: Complexity, Advantages and Disadvantage of Selection Sort

Time Complexity of Selection Sort-

- Best Case Complexity- **O(n^2)**
- Average Case Complexity- **O(n^2)**
- Worst Case Complexity- **O(n^2)**

Advantages-

- It is popular and easy to implement
- It sorts the numbers without using additional temporary storage

Disadvantage-

- This approach does not work for lists having large number of elements

Learning Outcomes

In this session, you have learnt to:

1. Define the basic concepts of sorting the array elements
2. Explain the algorithms for internal sorting techniques
3. Design and implement the appropriate sorting technique for developing solutions to real-world problems and applications in C
4. Students will be able to analyze the complexity of sorting algorithms
5. Students will be able to sort a list of elements by deciding and choosing the appropriate sorting algorithm

Go through the following learning resources on the platform

- Sorting Techniques-I

Q & A

If you have more questions, please post them in the community on the platform.

What Next?

In the next session the following concepts will be covered

- Sorting Techniques
  - Quick Sort
  - Merge Sort
  - Radix Sort

Go through the following learning resources on the platform

- Sorting Techniques-II