UNIT – IV: Sorting and Searching

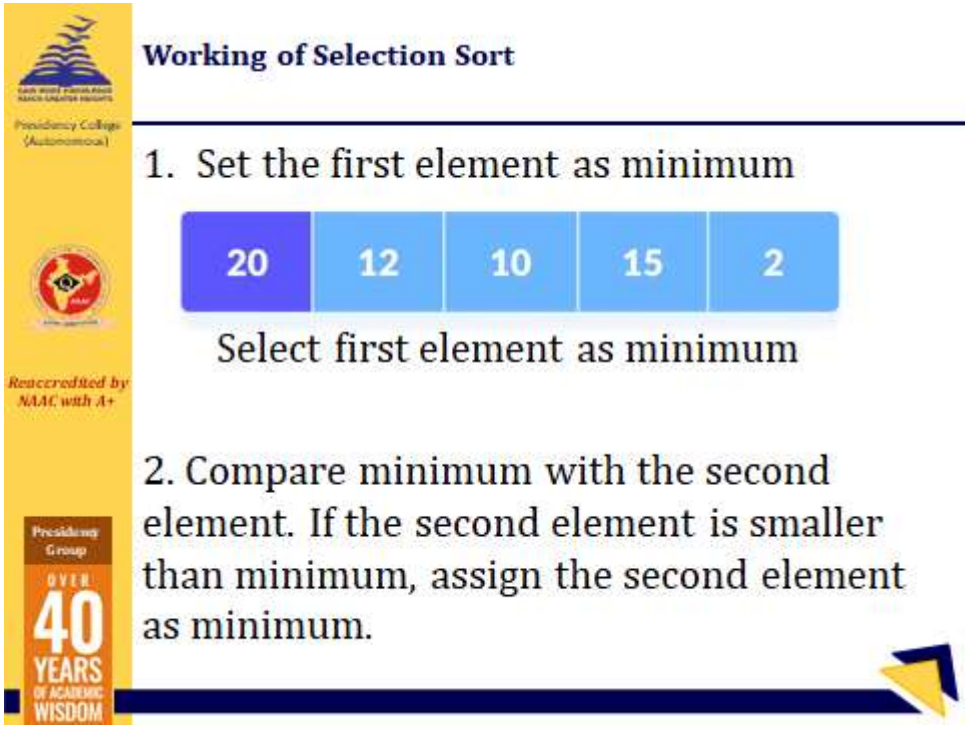
Sorting:

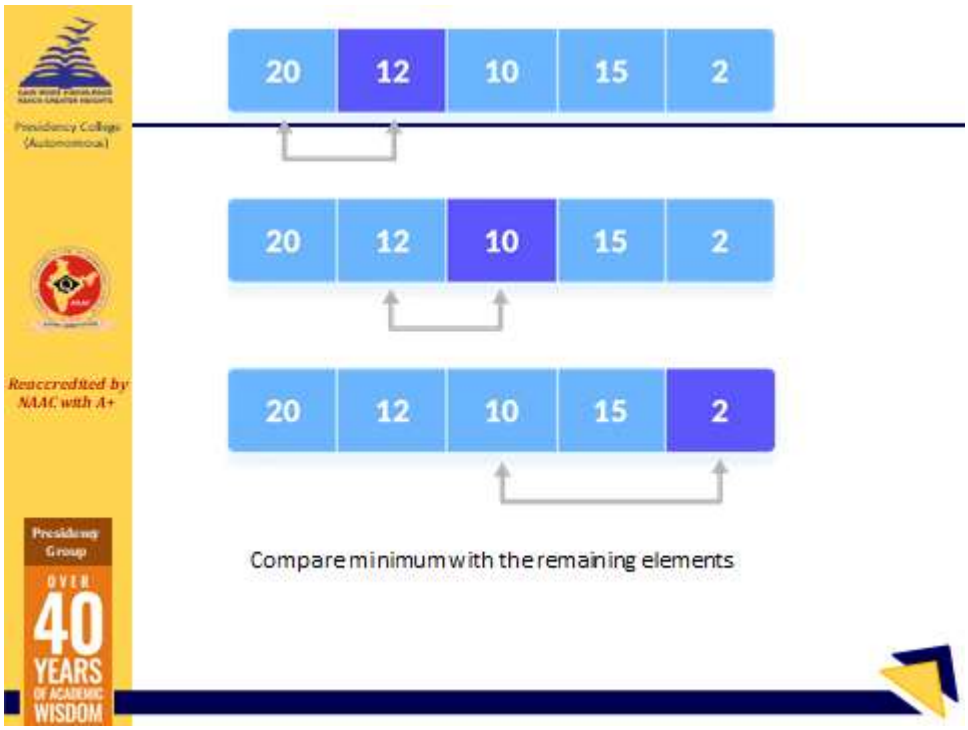
Sorting is **the process of arranging elements either in ascending (or) descending order**

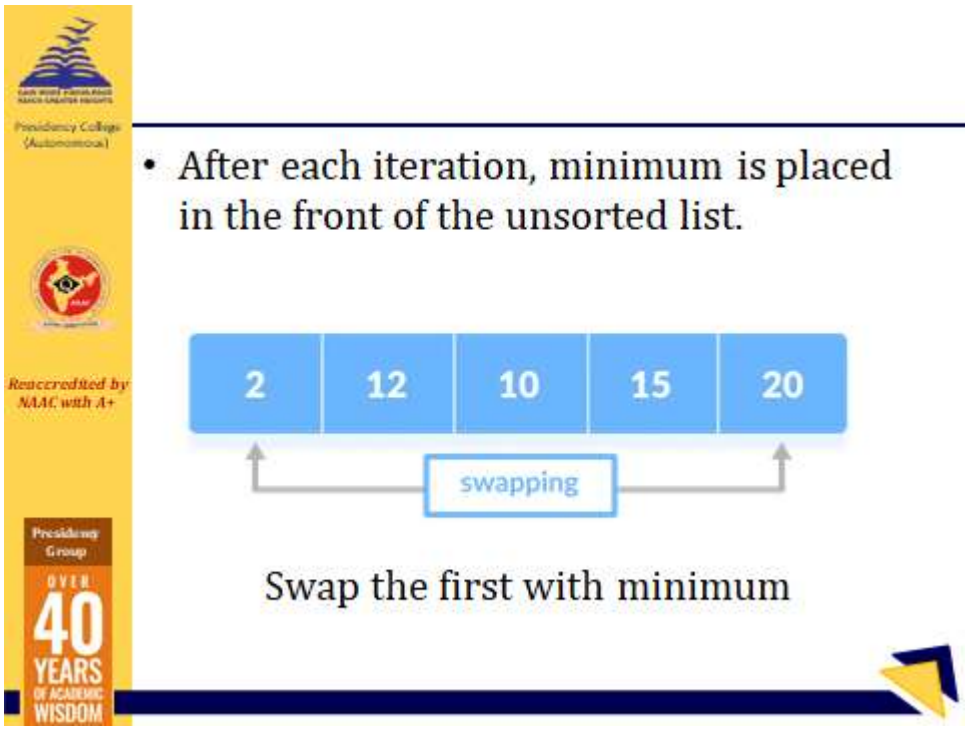
Sorting by selection

- Selection sort is a sorting algorithm that selects the smallest element from an unsorted list in each iteration and places that element at the beginning of the unsorted list.

Working of Selection Sort

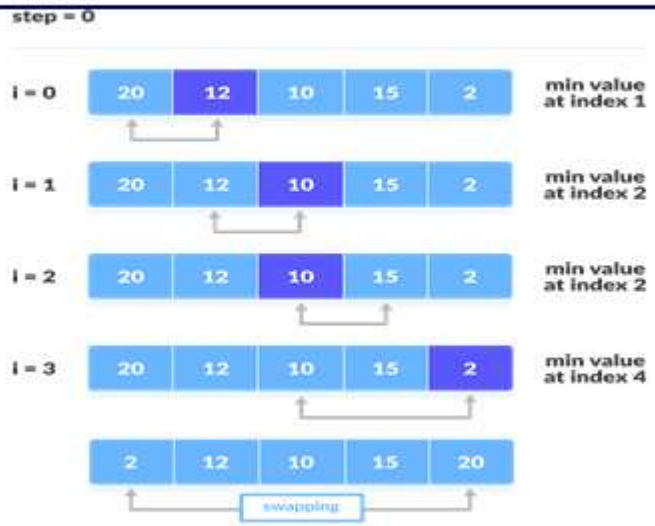



20 12 10 15 2
20 12 10 15 2
20 12 10 15 2
Compare minimum with the remaining elements

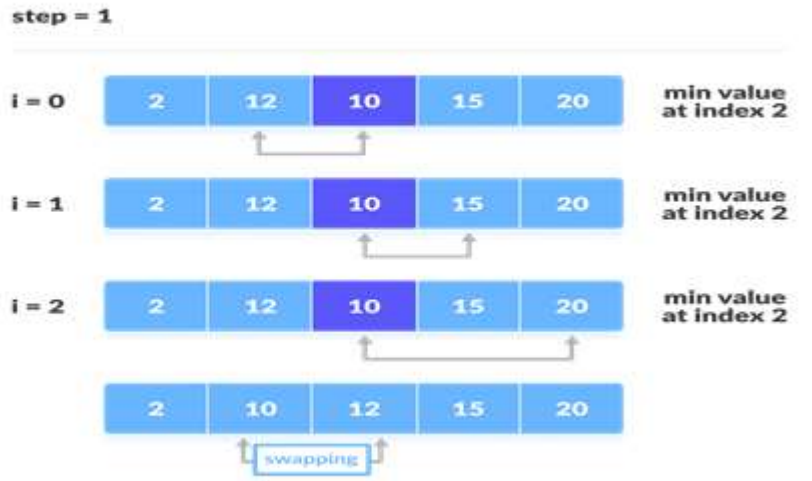

After each iteration, minimum is placed in the front of the unsorted list.
2 12 10 15 20
swapping
Swap the first with minimum

- For each iteration, indexing starts from the first unsorted element. Step 1 to 3 are repeated until all the elements are placed at their correct positions.

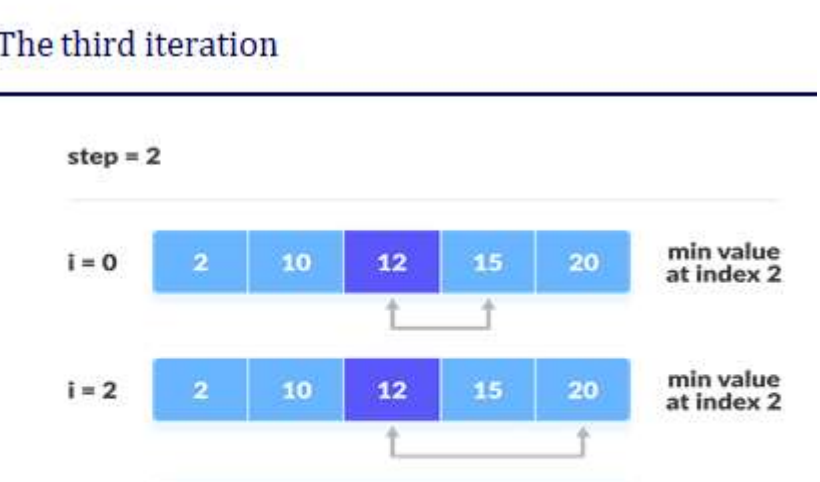
The first iteration

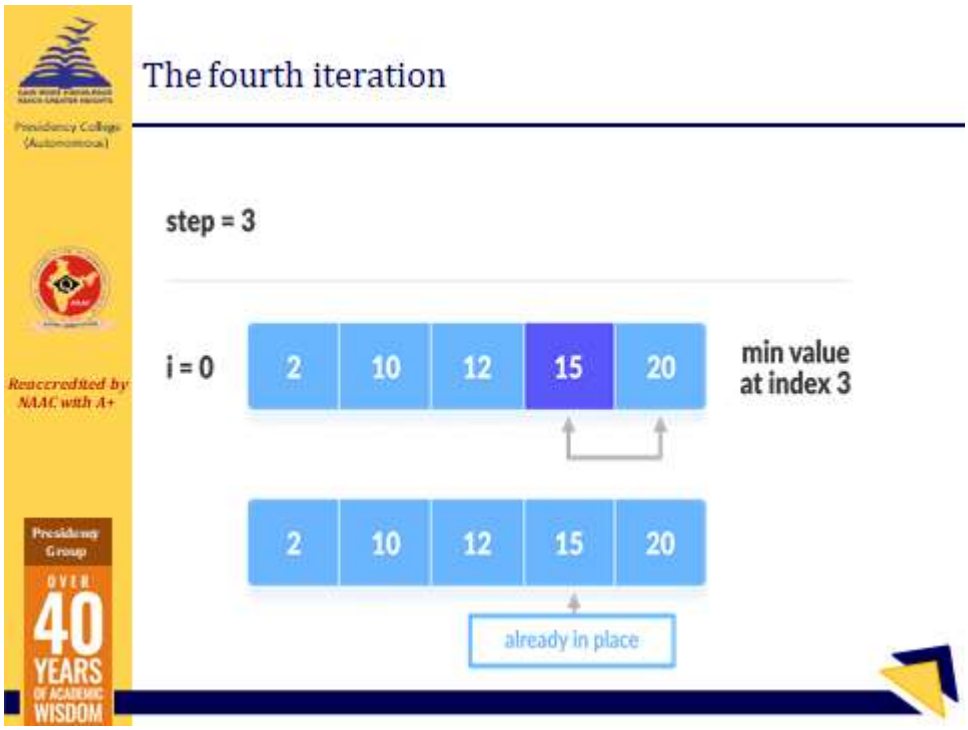


The second iteration



The third iteration



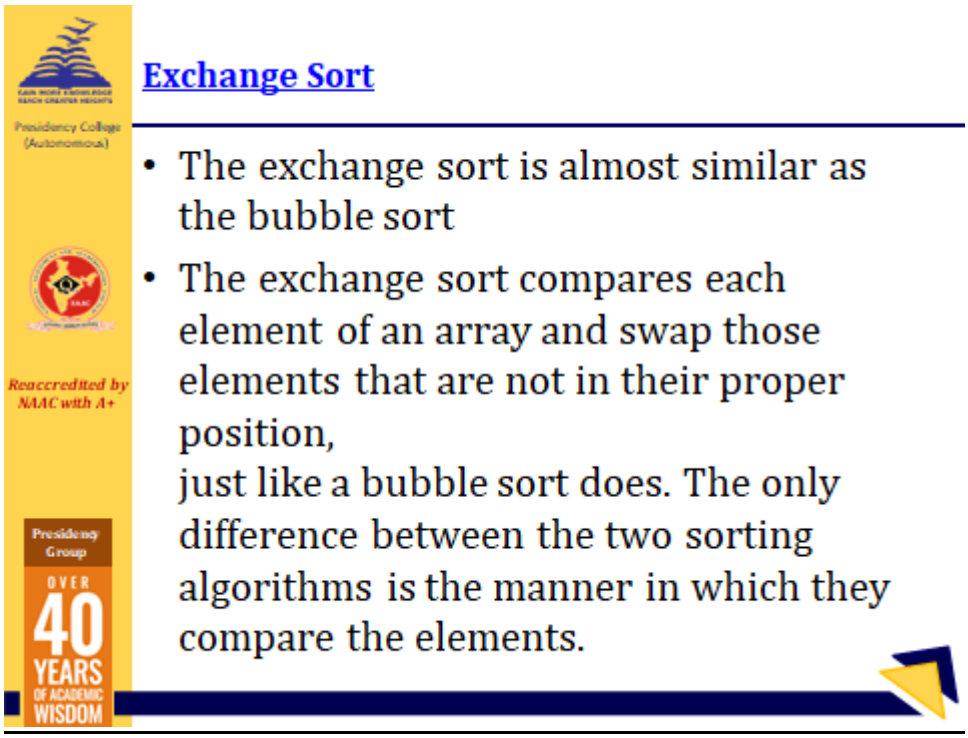


Selection sort Program Example:

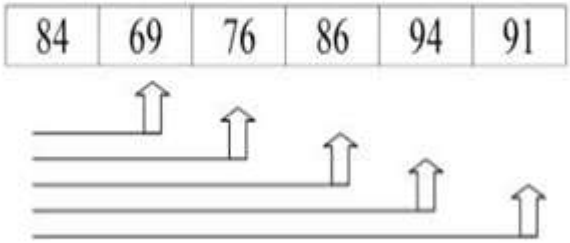
```
#include <stdio.h>
 void selection(int arr[], int n)
{
   int i, j, small,temp;
    for (i = 0; i < n-1; i++)    // One by one move boundary of unsorted subarray
   {
      small = i; //minimum element in unsorted array
        for (j = i+1; j < n; j++)
      if (arr[j] < arr[small])
          small = j;
// Swap the minimum element with the first element
   temp = arr[small];
   arr[small] = arr[i];
   arr[i] = temp;
   }
}
 void printArr(int a[], int n) /* function to print the array */
{
   int i;
   for (i = 0; i < n; i++)
```

```
        printf("%d ", a[i]);
}
  void main()
{
    int a[] = { 12, 31, 25, 8, 32, 17 };
    int n;
    clrscr();
     n = sizeof(a) / sizeof(a[0]);
    printf("Before sorting array elements are - \n");
    printArr(a, n);
    selection(a, n);
    printf("\nAfter sorting array elements are - \n");
    printArr(a, n);
    getch();
}
```

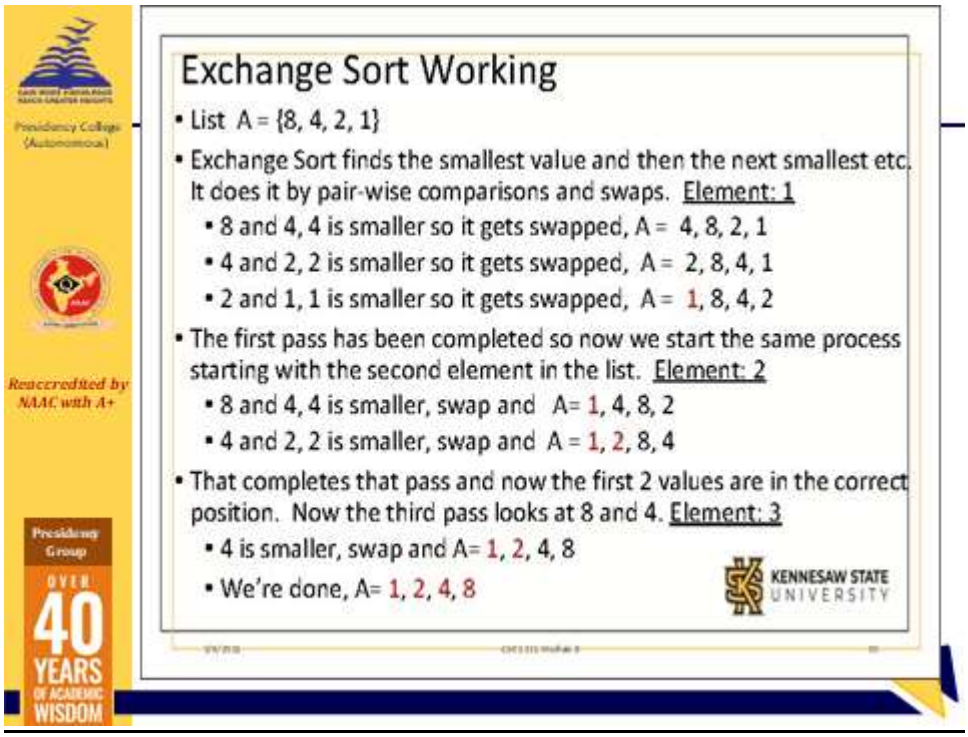
sorting by exchange:



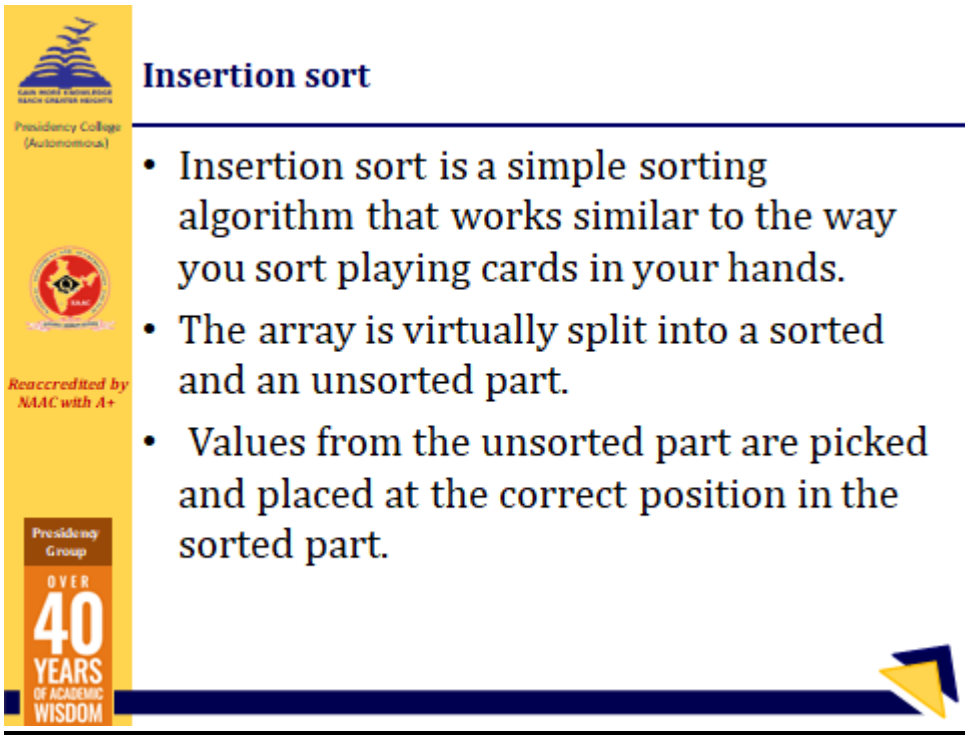
- The exchange sort compares the first element with each element of the array, making a swap where is necessary.

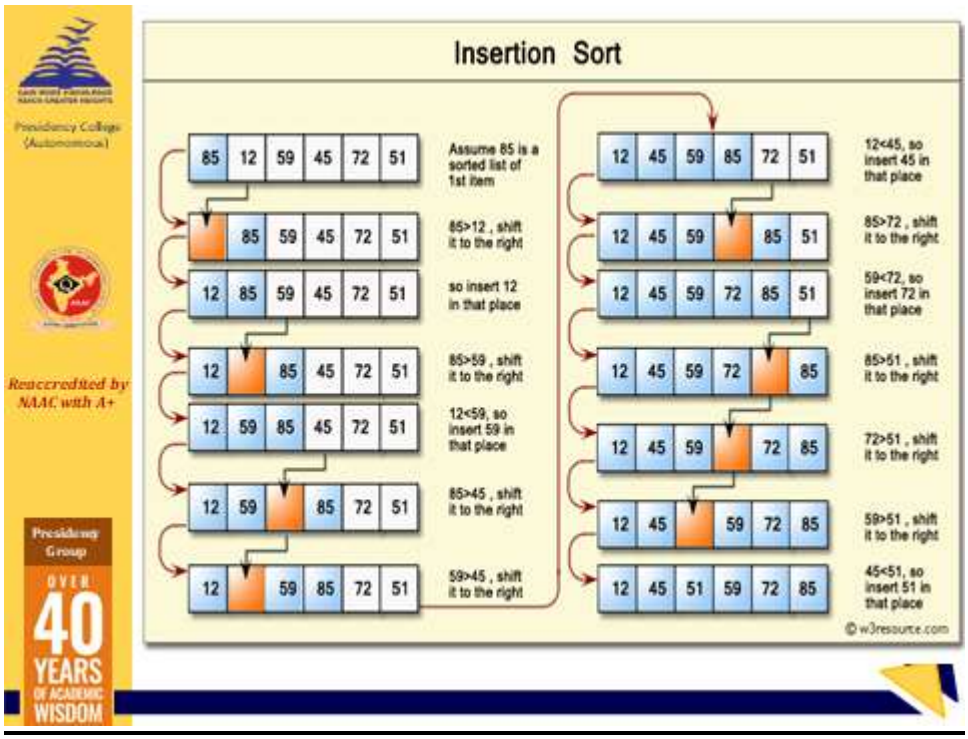


- When the first pass through the array is complete, the exchange sort then takes the second element and compares it with each following element of the array swapping elements that are out of order. This sorting process continues until the entire array is ordered.



Insertion Sort:





Example:

```
#include <stdio.h>

void insert(int a[], int n) /* function to sort an aay with
insertion sort */
{
   int i, j, temp;
   for (i = 1; i < n; i++) {
      temp = a[i];
      j = i - 1;
```

```
        while(j>=0 && temp <= a[j])  /* Move the elements
greater than temp to one position ahead from their
current position*/
        {
            a[j+1] = a[j];
            j = j-1;
        }
        a[j+1] = temp;
    }
}
void printArr(int a[], int n) /* function to print the array
*/
{
    int i;
    for (i = 0; i < n; i++)
   printf("%d ", a[i]);
}

void main()
{
    int a[] = { 12, 31, 25, 8, 32, 17 };
    int n = sizeof(a) / sizeof(a[0]);
    clrscr();
    printf("Before sorting array elements are - \n");
    printArr(a, n);
    insert(a, n);
    printf("\nAfter sorting array elements are - \n");
```

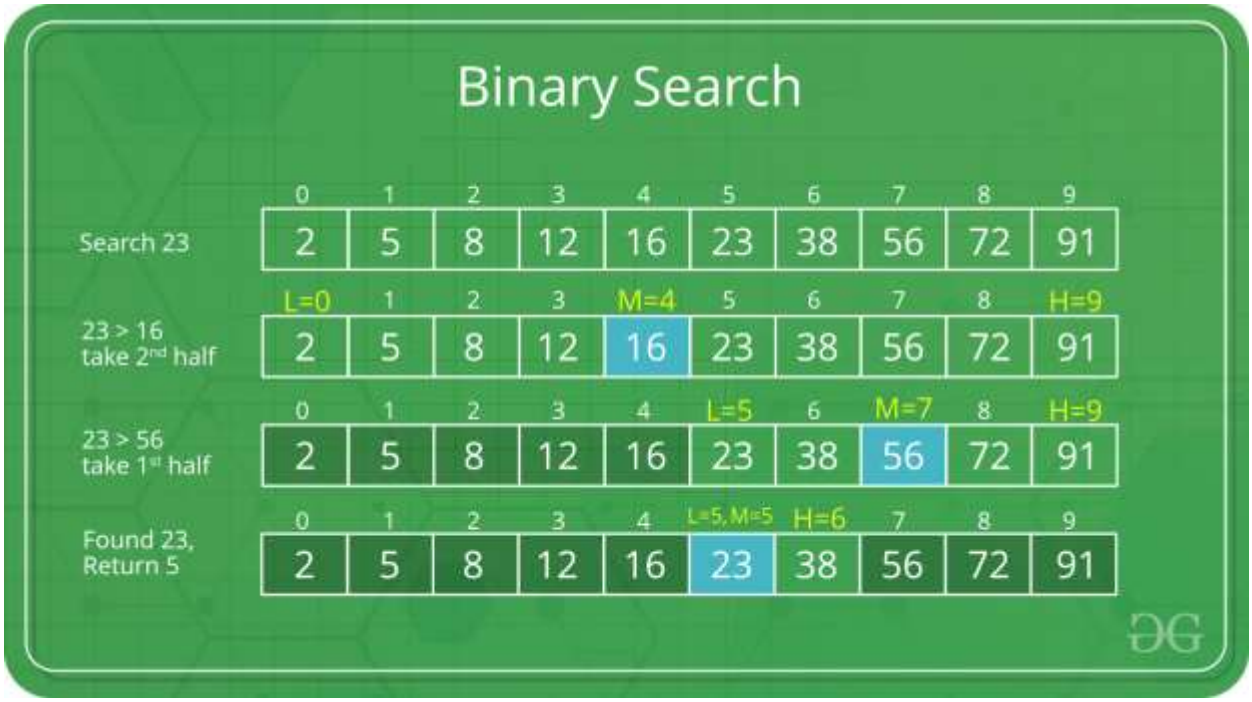
```
    printArr(a, n);
    getch();
}
```

Binary Search

Binary Search is a searching algorithm used in a sorted array by repeatedly dividing the search interval in half

steps to perform Binary Search are:

- Begin with an interval covering the whole array.
- If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half.
- Otherwise, narrow it to the upper half.
- Repeatedly check until the value is found or the interval is empty.



Refer lab program for example of binary search

Text processing and Pattern searching:

Pattern searching

- Pattern searching is a very crucial problem in computer science. Whenever we seek for a string in notepad/word file or browser or database or in some information, pattern searching algorithms are used to show the search results.

```
#include <stdio.h>
 #include <string.h>

int main (){
char txt[] =
"tutorialsPointisthebestplatformforprogrammers";
 char pat[] = "a";
int M = strlen (pat);
int N = strlen (txt);
for (int i = 0; i <= N - M; i++){
 int j;
 for (j = 0; j < M; j++)
  if (txt[i + j] != pat[j])
  break;     if (j == M)
printf ("Pattern matches at index %d \n", i);
 }
return 0; }
```

- Output

Pattern matches at 6

Pattern matches at 25

Pattern matches at 39

Structures and unions:

structure

- A structure is a user defined data type in C/C++. A structure creates a data type that can be used to group items of possibly different types into a single type(heterogeneous type of data).
- Each element of a structure is called a member.

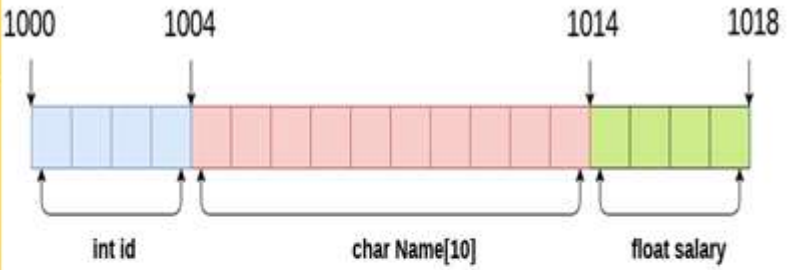
Syntax:

- The ,**struct** keyword is used to define the structure. Let's see the syntax to define the structure in c.

```
struct structure_name
{
  data_type member1;
  data_type member2;
  .
  .
  data_type memeberN;
};
```

Example:

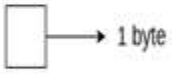
```
struct employee
{  int id;
   char name[20];
   float salary;
};
```



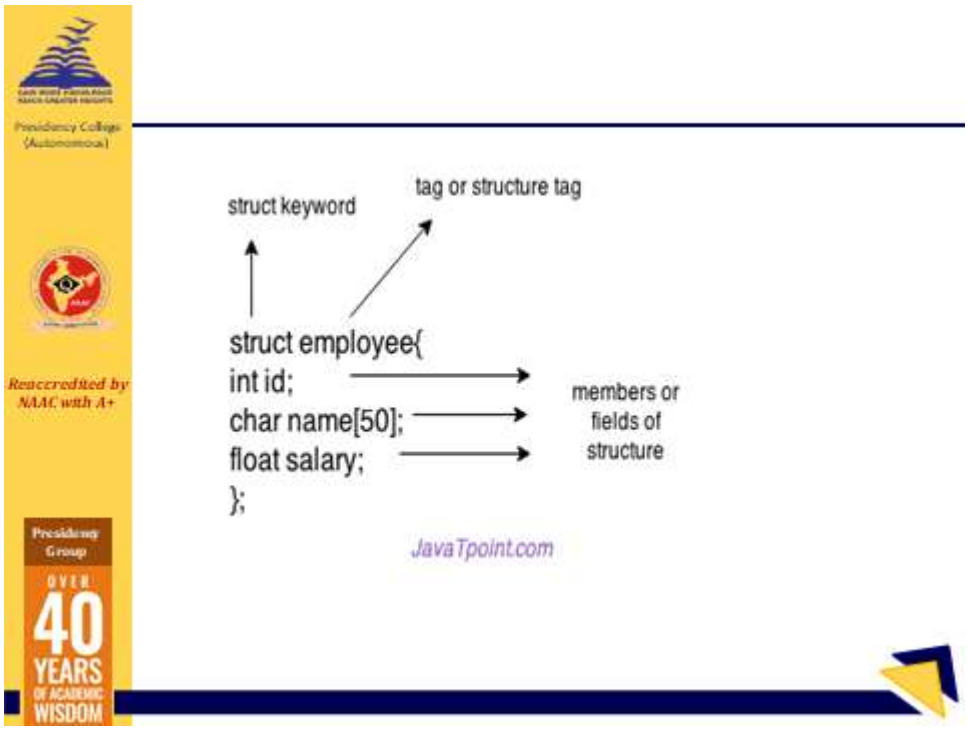

```
struct Employee
{
    int id;
    char Name[10];
    float salary;
} emp;
```

sizeof (emp) = 4 + 10 + 4 = 18 bytes

where;
sizeof (int) = 4 byte
sizeof (char) = 1 byte
sizeof (float) = 4 byte



- Here, struct is the keyword; employee is the name of the structure; id, name, and salary are the members or fields of the structure.



Declaring structure variable

We can declare a variable for the structure so that we can access the member of the structure easily. There are two ways to declare structure variable:

- By struct keyword within main() function
- By declaring a variable at the time of defining the structure.

```
struct employee
{  int id;
 char name[50];
  float salary;
};
```

- Now write given code inside the main() function.

```
struct employee e1, e2;
```

2nd way:

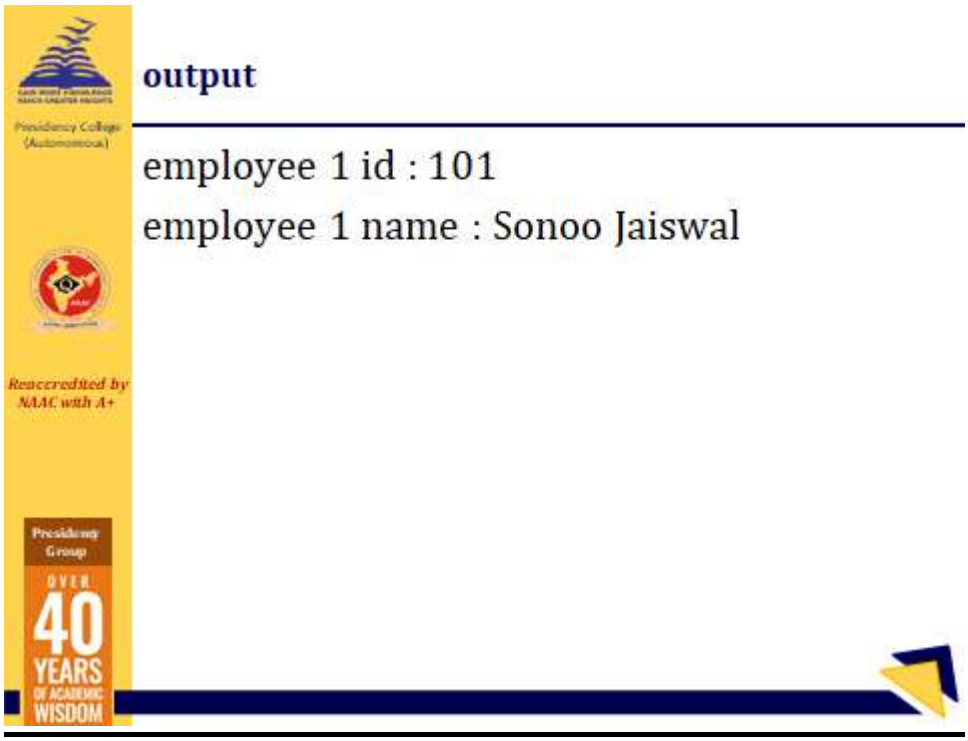
```
struct employee
{  int id;
 char name[50];
float salary;
}e1,e2;
```

Accessing members of the structure

There are two ways to access structure members:

- By . (member or dot operator)
- By -> (structure pointer operator)

```
#include<stdio.h>
#include <string.h>
struct employee
{  int id;
   char name[50];
}e1;  //declaring e1 variable for structure

int main( )
{
  //store first employee information
  e1.id=101;
  strcpy(e1.name, "Sonoo Jaiswal");//copying string into char
array
  //printing first employee information
  printf( "employee 1 id : %d\n", e1.id);
  printf( "employee 1 name : %s\n", e1.name);
return 0;
}
```



Union:

A union is a user-defined type similar to structs in C except for one key difference.

Structures allocate enough space to store all their members, whereas **unions can only hold one member value at a time**.

Union

- Union is an user defined datatype in C programming language.
- It is a collection of variables of different datatypes in the same memory location.
- We can define a union with many members, but at a given point of time only one member can contain a value

Need for Union in C programming

- C unions are used to save memory.
- C unions allow data members which are mutually exclusive to share the same memory

```
union union_name
{
datatype field_name;
datatype field_name;
// more variables
};
```

Note : Size of the union is the the size of its largest field because sufficient number of bytes must be reserved to store the largest sized field.

- To access the fields of a union, use dot(.) operator i.e., the variable name followed by dot operator followed by field name.

Example:

```
#include <stdio.h>
union Job {
   float salary;
   int workerNo;
} j;

int main() {
   j.salary = 12.3;

   // when j.workerNo is assigned a value,
   // j.salary will no longer hold 12.3
   j.workerNo = 100;

   printf("Salary = %.1f\n", j.salary);
   printf("Number of workers = %d", j.workerNo);
   return 0;
}
```

Output

```
Salary = 0.0
Number of workers = 100
```

Differences between Structure and Union:

	STRUCTURE	UNION
Keyword	The keyword **struct** is used to define a structure	The keyword **union** is used to define a union.
Size	When a variable is associated with a structure, the compiler allocates the memory for each member. The size of structure is **greater than or equal to the sum of sizes of its members.**	when a variable is associated with a union, the compiler allocates the memory by considering the size of the largest memory. So, size of **union is equal to the size of largest member.**
Memory	Each member within a structure is assigned unique storage area of location.	Memory allocated is shared by individual members of union.
Value Altering	Altering the value of a member will not affect other members of the structure.	Altering the value of any of the member will alter other member values.
Accessing members	Individual member can be accessed at a time.	Only one member can be accessed at a time.
Initialization of Members	Several members of a structure can initialize at once.	Only the first member of a union can be initialized.

Difference between unions and structures

Let's take an example to demonstrate the difference between unions and structures:

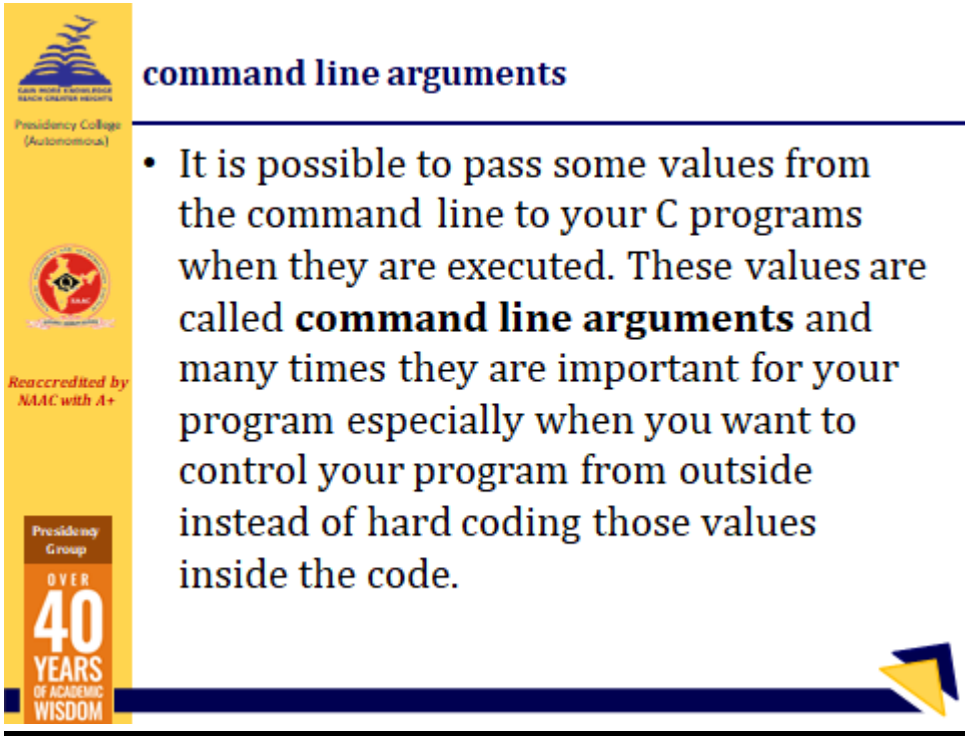
```
#include <stdio.h>
union unionJob
{
   //defining a union
   char name[32];
   float salary;
   int workerNo;
} uJob;

struct structJob
{
   char name[32];
   float salary;
   int workerNo;
} sJob;
```

```
int main()
{
   printf("size of union = %d bytes", sizeof(uJob));
   printf("\nsize of structure = %d bytes", sizeof(sJob));
   return 0;
}
```

Output

```
size of union = 32
size of structure = 40
```

Command line arguments.



int main(**int** argc, **char** *argv[])

- The command line arguments are handled using main() function arguments where **argc** refers to the number of arguments passed, and **argv[]** is a pointer array which points to each argument passed to the program

Example:

```
#include <stdio.h>
Void  main( int argc, char *argv[] ) {
 if( argc == 2 ) {
printf("The argument supplied is %s\n",
argv[1]); }
 else if( argc > 2 ) {
printf("Too many arguments supplied.\n");
}
else {
printf("One argument expected.\n");
} }
```