Unit-1

Array

Arrays a kind of data structure that can store a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

Properties of Array

The array contains the following properties.

- Each element of an array is of same data type and carries the same size, i.e., int = 2 bytes.
- Elements of the array are stored at contiguous memory locations where the first element is stored at the smallest memory location.
- Elements of the array can be randomly accessed since we can calculate the address of each element of the array with the given base address and the size of the data element.

Advantage of C Array

Code Optimization: Less code to the access the data.

Ease of traversing: By using the for loop, we can retrieve the elements of an array easily.

Ease of sorting: To sort the elements of the array, we need a few lines of code only.

Random Access: We can access any element randomly using the array.

Disadvantage of C Array

1) **Fixed Size**: Whatever size, we define at the time of declaration of the array, we can't exceed the limit. So, it doesn't grow the size dynamically like LinkedList which we will learn later.

Declaring Arrays

To declare an array in C, a programmer specifies the type of the elements and the number of elements required by an array as follows −

```
type arrayName [ arraySize ];
```

This is called a *single-dimensional* array. The arraySize must be an integer constant greater than zeroand type can be any valid C data type. For example, to declare a 10-element array called balance of type double, use this statement −

```
double balance[10];
```

Here *balance* is a variable array which is sufficient to hold up to 10 double numbers.

Initializing Arrays

You can initialize an array in C either one by one or using a single statement as follows –

```
double balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};
```

The number of values between braces { } cannot be larger than the number of elements that we declare for the array between square brackets [ ].

If you omit the size of the array, an array just big enough to hold the initialization is created. Therefore, if you write −

```
double balance[] = {1000.0, 2.0, 3.4, 7.0, 50.0};
```

You will create exactly the same array as you did in the previous example. Following is an example to assign a single element of the array −

```
balance[4] = 50.0;
```

The above statement assigns the 5^{th} element in the array with a value of 50.0. All arrays have 0 as the index

of their first element which is also called the base index and the last index of an array will be total size of the array minus 1. Shown below is the pictorial representation of the array we discussed above −

Accessing Array Elements

An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array. For example −

```
double salary = balance[9];
```

The above statement will take the 10^{th} element from the array and assign the value to salary variable. The following example Shows how to use all the three above mentioned concepts viz. declaration, assignment, and accessing arrays −

```
#include <stdio.h>int
 main () {
   int n[ 10 ]; /* n is an array of 10 integers */int
   i,j;
    /* initialize elements of array n to 0 */for (
   i = 0; i < 10; i++ )
   {
     n[ i ] = i + 100; /* set element at location i to i + 100 */
   }
     /* output each array element's value */for (j
   = 0; j < 10; j++ ) {
     printf("Element[%d] = %d\n", j, n[j] );
   }
    return 0;
}
```

Two Dimensional Array in C

The two-dimensional array can be defined as an array of arrays. The 2D array is organized as matrices which can be represented as the collection of rows and columns. However, 2D arrays are created to implement a relational database lookalike data structure. It provides ease of holding the bulk of data at once which can be passed to any number of functions wherever required.

Declaration of two dimensional Array in C

The syntax to declare the 2D array is given below.

```
data_type array_name[rows][columns];
```

Consider the following example.

```
int twodimen[4][3];
```

Here, 4 is the number of rows, and 3 is the number of columns.

Initialization of 2D Array in C

In the 1D array, we don't need to specify the size of the array if the declaration and initialization are being done simultaneously. However, this will not work with 2D arrays. We will have to define at least the second dimension of the array. The two-dimensional array can be declared and defined in the following way.

```
int arr[4][3]={{1,2,3},{2,3,4},{3,4,5},{4,5,6}};
```

Two-dimensional array example in C

```
#include<stdio.h>
int main(){
int i=0,j=0;
int arr[4][3]={{1,2,3},{2,3,4},{3,4,5},{4,5,6}};
//traversing      2D
array
for(i=0;i<4;i++){
for(j=0;j<3;j++){
                    printf("arr[%d] [%d] = %d \n",i,j,arr[i][j]);
 }//end of j
}//end of i
return 0;
}
```

2D array example: Storing elements in a matrix and printing it.

```
#include <stdio.h>
void main ()
{
   int arr[3][3],i,j;
   for (i=0;i<3;i++)
   {
      for (j=0;j<3;j++)
      {
         printf("Enter a[%d][%d]: ",i,j);
         scanf("%d",&arr[i][j]);
      }
   }
   printf("\n printing the elements  \n");
   for(i=0;i<3;i++)
   {
      printf("\n");
      for (j=0;j<3;j++)
      {
         printf("%d\t",arr[i][j]);
      }
   }
}
```

Passing array to a function

```
#include  <stdio.h>
 void getarray(int arr[])
{
       printf("Elements of array are :
       ");for(int i=0;i<5;i++)
       {
          printf("%d ", arr[i]);
       }
   }
   int main()
   {
     int arr[5]={45,67,34,78,90};
     getarray(arr);
     return 0;
   }
```

In the above program, we have first created the array **arr[]** and then we pass this array to the function getarray(). The **getarray()** function prints all the elements of the array arr[].

Sorting Techniques:

1- Bubble Sort Algorithm:

Bubble Sort is a simple algorithm which is used to sort a given set of n elements provided in form of an array with n number of elements. Bubble Sort compares the entire element one by one and sort them based on their values.

If the given array has to be sorted in ascending order, then bubble sort will start by comparing the first element of the array with the second element, if the first element is greater than the second element, it will **swap** both the elements, and then move on to compare the second and the third element, and so on.

If we have total n elements, then we need to repeat this process for n-1 times.

It is known as **bubble sort**, because with every complete iteration the largest element in the given array, bubbles up towards the last place or the highest index, just like a water bubble rises up to the water surface.

Sorting takes place by stepping through all the elements one-by-one and comparing it with the adjacent element and swapping them if required.

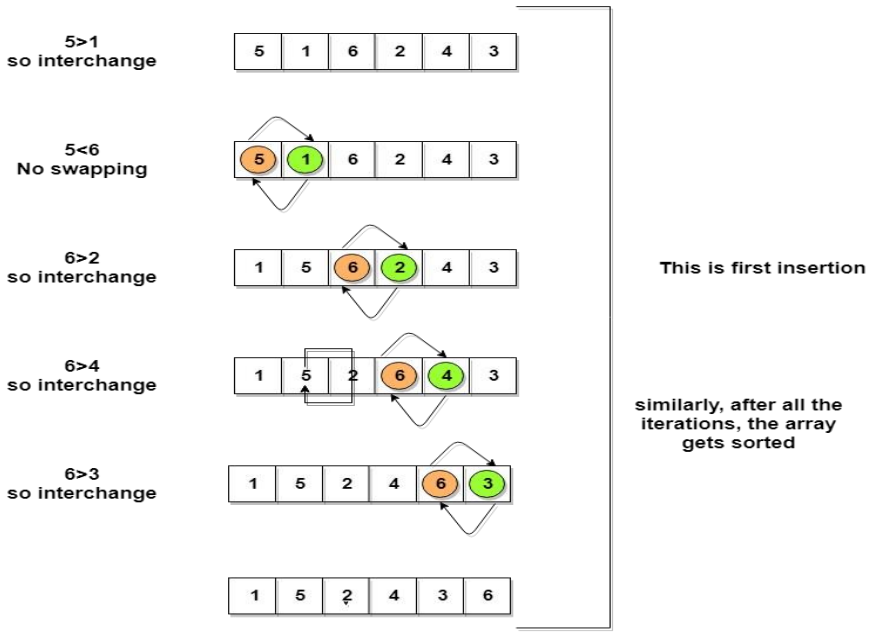
Implementing Bubble Sort Algorithm

Following are the steps involved in bubble sort (for sorting a given array in ascendingorder):

1. Starting with the first element (index = 0), compare the current element with thenext element of the array.
2. If the current element is greater than the next element of the array, swap them.
3. If the current element is less than the next element, move to the nextelement. **Repeat Step 1**.

Let's consider an array with values {5, 1, 6, 2, 4, 3}, Below, we have a pictorial representation of how bubble sort will sort the given array.

So as we can see in the representation above, after the first iteration, 6 is placed at thelast index, which is the correct position for it. Similarly after the second iteration, 5 will be at the second last index, and so on.



Code for bubble sort:

```
#include<stdio.h>
void main ()
{
   int i, j,temp;
   int a[10] = { 10, 9, 7, 101, 23, 44, 12, 78, 34, 23};
   for(i = 0; i<10; i++)
   {
      for(j = i+1; j<10; j++)
      {
         if(a[j] < a[i])
         {
            temp = a[i];
            a[i] = a[j];
            a[j] = temp;
         }
      }
   }
   printf("Printing Sorted Element List ...\n");
   for(i = 0; i<10; i++)
   {
      printf("%d\n",a[i]);
   }
}
```

2-Selection Sort Algorithm:

Selection sort is conceptually the most simplest sorting algorithm. This algorithm will first find the **smallest** element in the array and swap it with the element inthe **first** position, then it will find the **second smallest** element and swap it with the element in the **second** position, and it will keep on doing this until the entire array is sorted.

It is called selection sort because it repeatedly **selects** the next-smallest element and swaps it into the right place.

How Selection Sort Works?

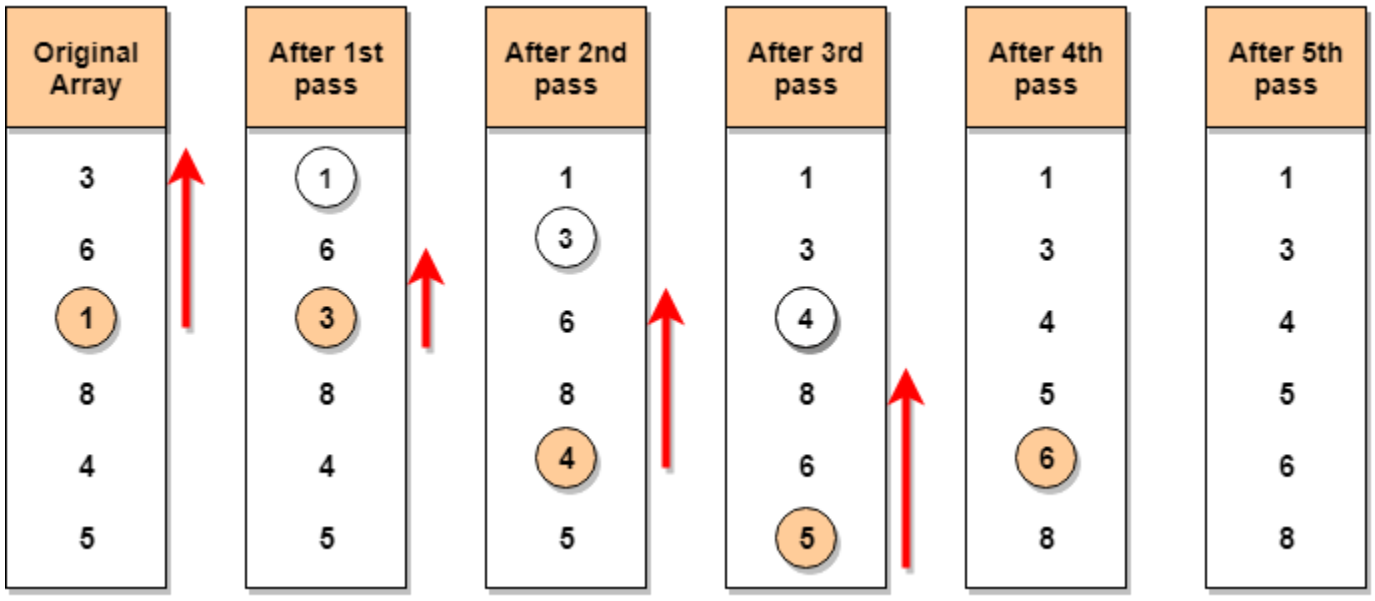
Following are the steps involved in selection sort(for sorting a given array inascending order):

1. Starting from the first element, we search the smallest element in the array, andreplace it with the element in the first position.

2. We then move on to the second position, and look for smallest element presentin the subarray, starting from index 1, till the last index.

3. We replace the element at the **second** position in the original array, or we can say at the first position in the subarray, with the second smallest element.

4. This is repeated, until the array is completely sorted. Let's consider an array with values {3, 6, 1, 8, 4, 5}
Below, we have a pictorial representation of how selection sort will sort the given array.



In the **first** pass, the smallest element will be 1, so it will be placed at the first position.

Then leaving the first element, **next smallest** element will be searched, from theremaining elements. We will get 3 as the smallest, so it will be then placed at thesecond position.

Then leaving 1 and 3(because they are at the correct position), we will search for the next smallest element from the rest of the elements and put it at third position and keep doing this until array is sorted.

Implementing Selection Sort Algorithm

In the C program below, we have tried to divide the program into small functions, sothat it's easier fo you to understand which part is doing what.

There are many different ways to implement selection sort algorithm, here is the onethat we like:

```
#include <stdio.h>int
main() {
  int arr[10]={6,12,0,18,11,99,55,45,34,2};
  int n=10;
  int i, j, position, swap;
  for (i = 0; i < (n - 1); i++) {
    position = i;
    for (j = i + 1; j < n; j++) {
      if (arr[position] > arr[j])
        position = j;
    }
    if (position != i) {
      swap = arr[i];
      arr[i] = arr[position];
      arr[position] = swap;
    }
  }
  for (i = 0; i < n; i++)
    printf("%d\t", arr[i]);
  return 0;
}
```

3-Insertion Sort Algorithm

This is exactly how **insertion sort** works. It starts from the index 1(not 0), and each index starting from index 1 is like a new card, that you have to place at the right position in the sorted subarray on the left.

Following are some of the important **characteristics of Insertion Sort**:

1. It is efficient for smaller data sets, but very inefficient for larger lists.
2. Insertion Sort is adaptive, that means it reduces its total number of steps if a partially sorted array is provided as input, making it efficient.
3. It is better than Selection Sort and Bubble Sort algorithms.
4. Its space complexity is less. Like bubble Sort, insertion sort also requires a single additional memory space.
5. It is a **stable** sorting technique, as it does not change the relative order of elements which are equal.

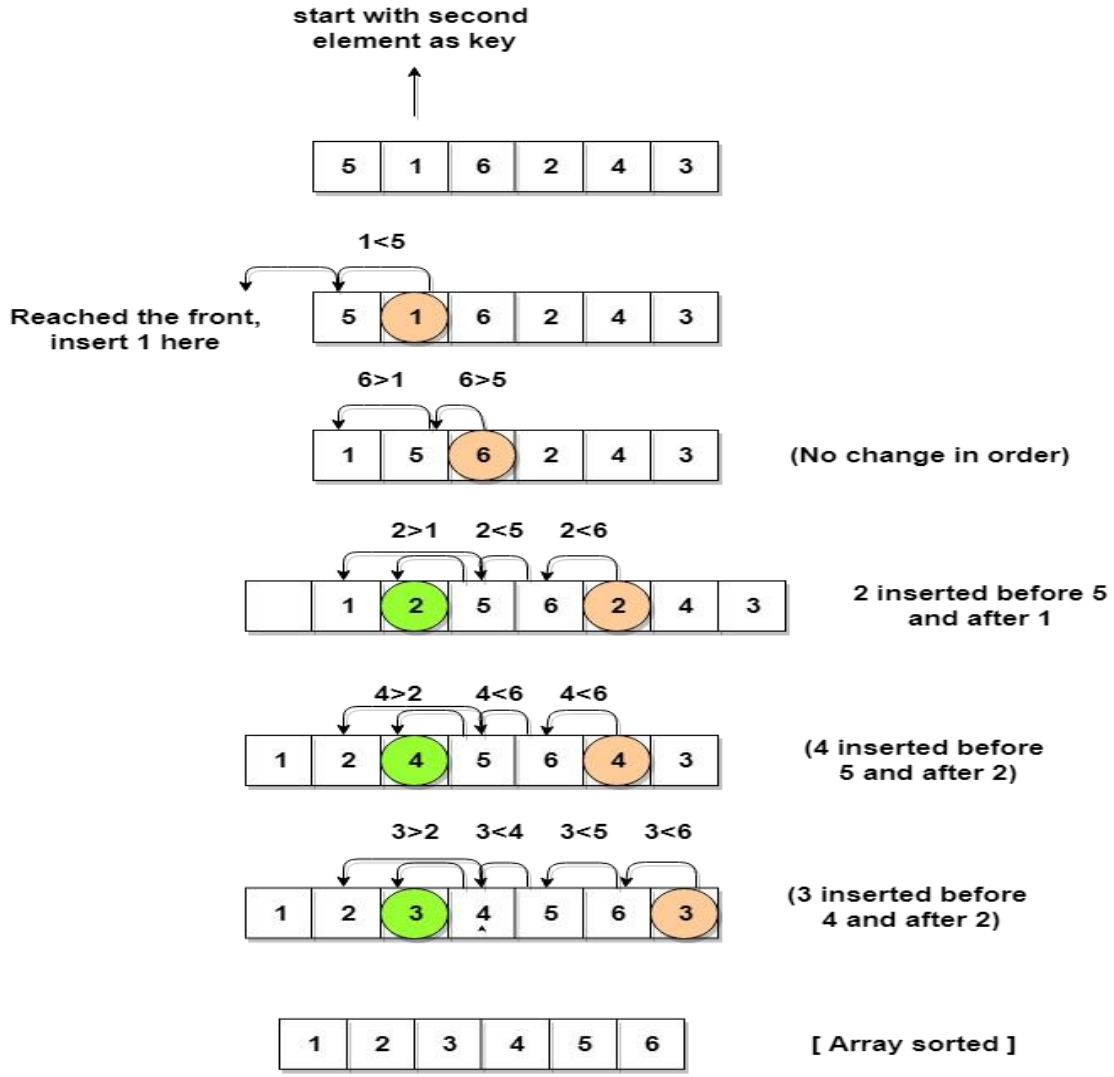
How Insertion Sort Works?

Following are the steps involved in insertion sort:

1. We start by making the second element of the given array, i.e. element at index 1, the key. The key element here is the new card that we need to add to our existing sorted set of cards(remember the example with cards above).

2. We compare the key element with the element(s) before it, in this case, element at index 0:
    - If the key element is less than the first element, we insert the key element before the first element.
    - If the key element is greater than the first element, then we insert it after the first element.
3. Then, we make the third element of the array as key and will compare it with elements to it's left and insert it at the right position.
4. And we go on repeating this, until the array is sorted.

Let's consider an array with values {5, 1, 6, 2, 4, 3}
Below, we have a pictorial representation of how bubble sort will sort the given array.



As you can see in the diagram above, after picking a key, we start iterating over the elements to the left of the key.

We continue to move towards left if the elements are greater than the key element and stop when we find the element which is less than the key element.

And, insert the key element after the element which is less than the key element.

```
Void  main()
{
int n, array[1000], c, d, t, flag = 0;
printf("Enter number of elements\n");
scanf("%d", &n);
printf("Enter %d integers\n", n);
for (c = 0; c < n; c++)
        scanf("%d", &array[c]);
for (c = 1 ; c <= n - 1; c++)
{
        t = array[c];
        for (d = c - 1 ; d >= 0; d--)
        {
                if (array[d] > t)
                {
                        array[d+1] = array[d];
                        flag = 1;
                }
                else
                        break;
        }
        if (flag==1)
                array[d+1] = t;
        }
printf("Sorted list in ascending order:\n");
for (c = 0; c <= n - 1; c++)
{
        printf("%d\n", array[c]);
}
}
```