1.What is array ? Explain the declaration and initialization of one dimensional and two dimensional array with an example

An array is a data structure in C that stores a fixed-size sequence of elements of the same data type. It allows you to store multiple values in a single variable, instead of declaring separate variables for each value.

Declaration of one-dimensional array:

```
data_type array_name[size];
```

For example:

```
int arr[5]; //declaring an array of 5 integers
```

Initialization of one-dimensional array:

```
data_type array_name[size] = {value1, value2, value3, ...};
```

For example:

```
int arr[5] = {1, 2, 3, 4, 5}; //initializing an array of 5 integers with values 1, 2, 3, 4, 5
```

Declaration of two-dimensional array:

```
data_type array_name[row_size][column_size];
```

For example:

```
int arr[2][3]; //declaring a two-dimensional array of 2 rows and 3 columns
```

Initialization of two-dimensional array:

```
data_type array_name[row_size][column_size] = {
                                    {value1, value2, value3},
                                    {value4, value5, value6}
                                };
```

For example:

```
int arr[2][3] = { {1, 2, 3}, {4, 5, 6} }; //initializing a two-dimensional array with values 1, 2, 3, 4, 5, 6
```

It's also possible to initialize only certain elements of the array, and leave the rest uninitialized.

2.Mention some advantages and disadvantages of Arrays

Advantages of Arrays:

1. Arrays allow for efficient index-based access to elements, making it easy to access and manipulate specific elements in the array.
2. Arrays use less memory than individual variables for storing similar data type values.
3. Arrays provide an efficient way to implement data structures like stack, queue, and priority queue.
4. They can be passed as function arguments, which can be useful for certain types of operations.
5. They are easy to implement and use.

Disadvantages of Arrays:

1. Arrays have a fixed size, which means that once an array is created, its size cannot be changed.
2. Inserting or deleting elements from an array can be time-consuming, as it may require shifting of other elements in the array.
3. Arrays require contiguous memory allocation, which can lead to wasted memory if the array is not fully utilized.
4. The size of an array must be fixed at compile time, which can be a limitation if the size is not known until runtime.
5. Accessing elements in a multi-dimensional array can be more complex than accessing elements in a single-dimensional array.

3.What is the time complexity for performing basic operations in an array?

The time complexity for performing basic operations in an array is as follows:

1. **Accessing an element: O(1)** - This is because arrays use indexing, so accessing an element at a specific index takes constant time regardless of the size of the array.
2. **Inserting an element: O(n) -** This is because inserting an element at a specific position in an array requires shifting all elements after that position to make room for the new element. The time complexity is O(n) because the number of elements that need to be shifted increases linearly with the size of the array.
3. **Deleting an element: O(n) -** Similar to inserting an element, deleting an element at a specific position in an array requires shifting all elements after that position to fill the gap left by the deleted element. The time complexity is also O(n) because the number of elements that need to be shifted increases linearly with the size of the array.
4. **Searching for an element: O(n) -** This is because searching for an element in an unsorted array requires visiting each element in the array, one by one, until the desired element is found. The time complexity is O(n) because the number of elements that need to be visited increases linearly with the size of the array.

5. **Sorting an array:** The time complexity for sorting an array depends on the sorting algorithm used. For example, the time complexity for bubble sort is O(n^2) while for quicksort it is O(n log n)

4. Difference between pointer and array in C?

Array	Pointer
Array syntax data type array_name[data type]	Pointer syntax data type *pointer_name
An array is a set of objects of the same type.	Whereas a pointer is a variable that contains the address of another variable.
The number of variables that an array may hold is determined by its size.	But a pointer variable can only store the address of one variable.
Arrays can be initialized while defining. For example arr[5] = {1,2,3,4,5}.	In comparison, pointers cannot be initialized at the definition.
Arrays are static in nature. The array size cannot be resized(allocated or freed) as per user requirements during runtime.	Pointers, on the other hand, are dynamic in nature. The memory allocation can be allocated or freed at any point in time.
Arrays are allocated at compile time.	Pointers are allocated at runtime.
We can not increment the array.	We can increment the pointer.
The scope has complete control over an array. It will correctly allocate the required memory, and when the variable is no longer in scope, the memory will be automatically released.	Pointers can cause a memory leak if we build a local pointer that points to dynamic memory and then forgets to release it.
Java has support for arrays.	But, Java doesn't have pointers; Java has references.
Memory allocation of an array is sequential.	Memory allocation is random.
The assembly code of the array is different from a pointer.	The assembly code of the pointer is different from an array.

6.What is the difference between a one-dimensional and multi-dimensional array?

Parameters	One-dimensional array(1D)	Two-dimensional array(2D)
Definition	A simple data structure that sequentially stores elements of the same data type.	A two-dimensional array stores a list of arrays with similar data types.
Declaration	Syntax: type variable_name[size]; Here type refers to the data type, and size refers to the number of elements that the array can hold.	Syntax: type variable_name[size1][size2]; Here type refers to datatype size1 refers to the number of rows, and size2 to the number of columns of the array.
Dimensions	A one-dimensional array has only one dimension.	A two-dimensional array has a total of two dimensions.
Size(bytes)	The size of the 1D array is: Total number of Bytes =sizeof(datatype of array variable)* size of the array.	The size of the 2D array is: Total number of Bytes = size of(datatype of the variable of the array)* the size of the first index * the size of the second index.
Representation	The 1D array represents multiple data items in the form of a list.	The 2D array represents multiple data items in the form of a table consisting of rows and columns.
Address calculation	Address of element arr[i] = b+w*i Here b is the base address, w is the size of each element, and i is the index of the array.	Address of arr[i[[j] can be calculated in two ways: **Row Major:** Address of element arr[i][j]=b+(n(i-l1)+(j-l2)) **Column Major:** Address of element arr[i][j]=b+(m(j-l2)+(i-l1)) Here b is the base address, w is the size of each element, n is the number of rows, and m is the number of the column. l1 specifies the lower bound of the row, and l2 identifies the lower bound of the column.
Row column matrix	A one-dimensional array has no row-column matrix.	A two-dimensional array has a row-column matrix.

What is the difference between an array and a pointer in C

Array	Pointer
Array syntax data type array_name[data type]	Pointer syntax data type *pointer_name
An array is a set of objects of the same type.	Whereas a pointer is a variable that contains the address of another variable.
The number of variables that an array may hold is determined by its size.	But a pointer variable can only store the address of one variable.
Arrays can be initialized while defining. For example arr[5] = {1,2,3,4,5}.	In comparison, pointers cannot be initialized at the definition.
Arrays are static in nature. The array size cannot be resized(allocated or freed) as per user requirements during runtime.	Pointers, on the other hand, are dynamic in nature. The memory allocation can be allocated or freed at any point in time.
Arrays are allocated at compile time.	Pointers are allocated at runtime.
We can not increment the array.	We can increment the pointer.
The scope has complete control over an array. It will correctly allocate the required memory, and when the variable is no longer in scope, the memory will be automatically released.	Pointers can cause a memory leak if we build a local pointer that points to dynamic memory and then forgets to release it.
Java has support for arrays.	But, Java doesn't have pointers; Java has references.
Memory allocation of an array is sequential.	Memory allocation is random.
The assembly code of the array is different from a pointer.	The assembly code of the pointer is different from an array.

11. How do you search for a specific element in an array?

Example
Input
Input size of array: 10
Input elements in array: 10, 12, 20, 25, 13, 10, 9, 40, 60, 5
Output
Element to search is: 25
Element found at index 3

Algorithm to search element in array

There are two searching techniques linear and binary. For simplicity, I am implementing linear search algorithm to search element in array.

Step by step descriptive logic to search element in array using linear search algorithm.

1. Input size and elements in array from user. Store it in some variable say `size` and `arr`.
2. Input number to search from user in some variable say `toSearch`.
3. Define a flag variable as `found = 0`. I have initialized `found` with 0, which means initially I have assumed that searched element does not exists in array.
4. Run loop from 0 to `size`. Loop structure should look like `for(i=0; i<size; i++)`.
5. Inside loop check if current array element is equal to searched number or not. Which is `if(arr[i] == toSearch)` then set `found = 1` flag and terminate from loop. Since element is found no need to continue further.
6. Outside loop `if(found == 1)` then element is found otherwise not.

Program to search element in array

```c
#include <stdio.h>
#define MAX_SIZE 100 // Maximum array size
int main()
{
    int arr[MAX_SIZE];
    int size, i, toSearch, found;

    /* Input size of array */
    printf("Enter size of array: ");
    scanf("%d", &size);

    /* Input elements of array */
    printf("Enter elements in array: ");
    for(i=0; i<size; i++)
    {
        scanf("%d", &arr[i]);
    }
```

```
    printf("\nEnter element to search: ");
    scanf("%d", &toSearch);

    /* Assume that element does not exists in array */
    found = 0;

    for(i=0; i<size; i++)
    {
        /*
         * If element is found in array then raise found flag
         * and terminate from loop.
         */
        if(arr[i] == toSearch)
        {
            found = 1;
            break;
        }
    }

    /*
     * If element is not found in array
     */
    if(found == 1)
    {
        printf("\n%d is found at position %d", toSearch, i + 1);
    }
    else
    {
        printf("\n%d is not found in the array", toSearch);
    }

    return 0;
}
```

14) What is a dynamic array and how is it implemented in C?

A dynamic array is an array with a size that can be changed during runtime. In C, dynamic arrays are typically implemented by allocating memory on the heap using the malloc function and storing a pointer to the first element of the array. The size of the array can be modified by reallocating memory with a different size using realloc. The memory must be manually deallocated when it is no longer needed using the free function. Here is an example of how a dynamic array of integers might be implemented in C:

```
int* arr;

int size = 10;

arr = (int*) malloc(size * sizeof(int));

// Use arr like a regular array
```

```
arr[0] = 5;

arr[1] = 3;

// Resize the array

size = 20;

arr = (int*) realloc(arr, size * sizeof(int));

// Use the resized array

arr[10] = 7;

arr[19] = 8;

// Remember to free the memory when done

free(arr);
```

16) How does linear search algorithm work and what is its time complexity?

Linear search is an algorithm that is used to find the position of an element in an array or a list. The basic idea behind linear search is to iterate through the array or list, one element at a time, starting from the first element, until the element being searched for is found.

Here is an example of how the linear search algorithm might be implemented in C:

```
int linear_search(int arr[], int n, int x) {
   for (int i = 0; i < n; i++) {
      if (arr[i] == x) {
         return i;
      }
   }
   return -1;
}
```

The time complexity of linear search is O(n), where n is the number of elements in the array or list. This means that the time it takes for the algorithm to complete increases linearly with the size of the input. Linear search is efficient for small arrays or lists, but it becomes less efficient as the size of the input increases.

It's worth noting that there are other search algorithms that have a better time complexity such as binary search which is O(log(n)) for a sorted array.

18)What are the advantages and disadvantages of using linear search vs binary search?

Basis of comparison	Linear search	Binary search
Definition	The linear search starts searching from the first element and compares each element with a searched element till the element is not found.	It finds the position of the searched element by finding the middle element of the array.
Sorted data	In a linear search, the elements don't need to be arranged in sorted order.	The pre-condition for the binary search is that the elements must be arranged in a sorted order.
Implementation	The linear search can be implemented on any linear data structure such as an array, linked list, etc.	The implementation of binary search is limited as it can be implemented only on those data structures that have two-way traversal.
Approach	It is based on the sequential approach.	It is based on the divide and conquer approach.
Size	It is preferrable for the small-sized data sets.	It is preferrable for the large-size data sets.
Efficiency	It is less efficient in the case of large-size data sets.	It is more efficient in the case of large-size data sets.
Worst-case scenario	In a linear search, the worst- case scenario for finding the element is O(n).	In a binary search, the worst-case scenario for finding the element is $O(\log_2 n)$.
Best-case scenario	In a linear search, the best-case scenario for finding the first element in the list is O(1).	In a binary search, the best-case scenario for finding the first element in the list is O(1).

Dimensional array	It can be implemented on both a single and multidimensional array.	It can be implemented only on a multidimensional array.

24) How can you sort an array in descending order?

Algorithm

1. Declare and initialize an array.
2. Loop through the array and select an element.
3. Inner loop will be used to compare selected element from outer loop with rest of the elements of array.
4. If any element is greater than the selected element then swap the values.
5. Continue this process till entire list is sorted in descending order.

```
#include <stdio.h>
 int main()
{
   //Initialize array
   int arr[] = {5, 2, 8, 7, 1};
   int temp = 0;

   //Calculate length of array arr
   int length = sizeof(arr)/sizeof(arr[0]);

   //Displaying elements of original array
   printf("Elements of original array: \n");
   for (int i = 0; i < length; i++) {
      printf("%d ", arr[i]);
   }

   //Sort the array in descending order
   for (int i = 0; i < length; i++) {
      for (int j = i+1; j < length; j++) {
        if(arr[i] < arr[j]) {
            temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
      }
```

```
    }
    printf("\n");
    //Displaying elements of array after sorting
    printf("Elements of array sorted in descending order: \n");
    for (int i = 0; i < length; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

Output:

```
Elements of original array:
5 2 8 7 1
Elements of array sorted in descending order:
8 7 5 2 1
```

25)How can you sort an array in ascending order?

ALGORITHM:

- **STEP 1:** START
- **STEP 2:** INITIALIZE arr[] ={5, 2, 8, 7, 1 }..
- **STEP 3:** SET temp =0
- **STEP 4:** length= sizeof(arr)/sizeof(arr[0])
- **STEP 5:** PRINT "Elements of Original Array"
- **STEP 6:** SET i=0. REPEAT STEP 7 and STEP 8 UNTIL i<length
- **STEP 7:** PRINT arr[i]
- **STEP 8:** i=i+1.
- **STEP 9:** SET i=0. REPEAT STEP 10 to STEP UNTIL i<n
- **STEP 10:** SET j=i+1. REPEAT STEP 11 UNTIL j<length
- **STEP 11:** if(arr[i]>arr[j]) then
- temp = arr[i]
- arr[i]=arr[j]
- arr[j]=temp
- **STEP 12:** j=j+1.
- **STEP 13:** i=i+1.
- **STEP 14:** PRINT new line
- **STEP 15:** PRINT "Elements of array sorted in ascending order"
- **STEP 16:** SET i=0. REPEAT STEP 17 and STEP 18 UNTIL i<length
- **STEP 17:** PRINT arr[i]
- **STEP 18:** i=i+1.
- **STEP 19:** RETURN 0.
- **STEP 20:** END.

PROGRAM:

```c
#include <stdio.h>

int main()
{
    //Initialize array
    int arr[] = {5, 2, 8, 7, 1};
    int temp = 0;

    //Calculate length of array arr
    int length = sizeof(arr)/sizeof(arr[0]);

    //Displaying elements of original array
    printf("Elements of original array: \n");
    for (int i = 0; i < length; i++) {
        printf("%d ", arr[i]);
    }

    //Sort the array in ascending order
    for (int i = 0; i < length; i++) {
        for (int j = i+1; j < length; j++) {
            if(arr[i] > arr[j]) {
                temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }

    printf("\n");

    //Displaying elements of array after sorting
    printf("Elements of array sorted in ascending order: \n");
    for (int i = 0; i < length; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

Output:

```
Elements of original array:
5 2 8 7 1
Elements of array sorted in ascending order:
1 2 5 7 8
```