Data Structures and Algorithms

by

Love Kumar

Website, LinkedIn, github

Contents

Chapter 1

Data Structures and Algorithms

A **data structure** is a way of **collecting and organising data**. And, an algorithm is a collection of steps to solve a particular problem. Learning data structures and algorithms allow us to **write efficient and optimized computer programs.**

1.1 Bit Manipulation Algorithms

1.1.1 Binary Number System

in **decimal**

$$Ex : (125)_{10} = (1 \times 10^2 + 2 \times 10^1 + 5 \times 10^0)_{10} \tag{1.1}$$

in **octal**

$$(125)_{10} = (1 \times 8^2 + 7 \times 8^1 + 5 \times 8^0)_{10} = (175)_8 \tag{1.2}$$

in **hexadecimal**

$$(125)_{10} = (7 \times 16^1 + 13 \times 16^0)_{10} = (7D)_{16} \tag{1.3}$$

in **binary**

$$Ex : (101)_2 = (1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0)_{10} \tag{1.4}$$

1.1.2 Conversion

1. Decimal to Binary

Performing Division by Two with Remainder (For integer part)

Example 1. Converting decimal number 112 into binary number.

Division	Remainder (R)
112 / 2 = 56	0
56 / 2 = 28	0
28 / 2 = 14	0
14 / 2 = 7	0
7 / 2 = 3	1
3 / 2 = 1	1
1 / 2 = 0	1

Now, write remainder from bottom to up (in reverse order), this will be 1110000 which is equivalent binary number of decimal integer 112.

Example 2. Convert **decimal fractional** number 0.8125 into binary number.

Multiplication	Resultant integer part (R)
0.81252 x 2= 1.625	1
0.6252 x 2= 1.25	1
0.252 x 2= 0.50	0
0.52 x 2= 1.0	1
0 x 2 = 0	0

Now, write these resultant integer part, this will be 0.11010 which is equivalent binary fractional number of decimal fractional 0.8125.

1.1.3 Bitwise Operators

```
// C Program to demonstrate use of bitwise operators
#include <stdio.h>
int main()
{
    // a = 5(00000101), b = 9(00001001)
    unsigned char a = 5, b = 9;
    printf("a = %d, b = %d\n", a, b);
// The & (bitwise AND)
    // The result is 00000001
    printf("a&b = %d\n", a & b);
// The | (bitwise OR)
```

```
    // The result is 00001101
    printf("a|b = %d\n", a | b);
// The ^ (bitwise XOR)
    // The result is 00001100
    printf("a^b = %d\n", a ^ b);
// The ~ (bitwise NOT) in C or C++ takes
//one number and inverts all bits of it
    // The result is 11111010
    printf("~a = %d\n", a = ~a);
// The << (left shift)
    // The result is 00010010
    printf("b<<1 = %d\n", b << 1);
// The >> (right shift)
    // The result is 00000100
    printf("b>>1 = %d\n", b >> 1);

    return 0;
}
```

```
output
a = 5, b = 9
a&b = 1
a|b = 13
a^b = 12
~a = 250
b<<1 = 18
```

```
b>>1 = 4
```

1.2 Question

1. Check if a Number is Odd or Even using Bitwise Operators.

```
#include <iostream>
using namespace std;
int main()
{int n=5;
//using &
 ((n&1)==0) ? cout<<"even " : cout<<"odd ";
//using ^ XOR
 ((n ^ 1) == (n + 1)) ? cout<<"even " : cout<<"odd ";
//using || or
 ((n | 1) > n) ? cout<<"even " : cout<<"odd ";
}
```

```
output
odd odd odd
```

1.3 Notes

The bitwise operators should not be used in place of logical operators. The result of logical operators **(&&,|| and !)** is either 0 or 1, but bitwise operators return an integer value. Also, the logical operators consider any non-zero operand as 1.

```c
#include <stdio.h>
int main()
{
        int x = 2, y = 5;
        (x & y) ? printf("True ") : printf("False ");
        (x && y) ? printf("True ") : printf("False ");
        return 0;
}
```

```
output
False True
```

Chapter 2

Notes

2.1 OOPS

2.1.1 Abstraction:

Data abstraction is one of the most essential and important features of object-oriented programming in C++. Abstraction means displaying only essential information and hiding the details. Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.
Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of the car or applying brakes will stop the car but he does not know about how on pressing accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of accelerator, brakes etc in the car. This is what abstraction is.

2.1.2 Polymorphism:

The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.
A person at the same time can have different characteristic. Like a man at the same time is a father, a husband, an employee. So the same person posses different behaviour in different situations. This is called polymorphism.

2.1.3 Inheritance:

The capability of a class to derive properties and characteristics from another class is called Inheritance. Inheritance is one of the most important features of Object-Oriented Programming.

2.1.4 Encapsulation:

In normal terms, Encapsulation is defined as wrapping up of data and information under a single unit. In Object-Oriented Programming, Encapsulation is defined as binding together the data and the functions that manipulate them.
Consider a real-life example of encapsulation, in a company, there are different sections like the accounts section, finance section, sales section etc. The finance section handles all the financial transactions and keeps records of all the data related to finance. Similarly, the sales section handles all the sales-related activities and keeps records of all the sales. Now there may arise a situation when for some reason an official from the finance section

needs all the data about sales in a particular month. In this case, he is not allowed to directly access the data of the sales section. He will first have to contact some other officer in the sales section and then request him to give the particular data. This is what encapsulation is. Here the data of the sales section and the employees that can manipulate them are wrapped under a single name “sales section”.

2.2 Data Structures

A data structure is a particular way of organizing data in a computer so that it can be used effectively.

2.2.1 Array Data Structure

An array is a collection of items stored at contiguous memory locations

2.2.2 Linked List Data Structure

A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations.

2.2.3 Binary Tree Data Structure

A tree whose elements have at most 2 children is called a binary tree. Since each element in a binary tree can have only 2 children, we typically name them the left and right child. A Binary Tree node contains following parts.

- Data

- Pointer to left child
- Pointer to right child

2.2.4 Binary Search Tree

Binary Search Tree is a node-based binary tree data structure which has the following properties:

- The left subtree of a node contains only nodes with keys lesser than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- The left and right subtree each must also be a binary search tree.

2.2.5 Hashing Data Structure

Hashing is an important Data Structure which is designed to use a special function called the Hash function which is used to map a given value with a particular key for faster access of elements. The efficiency of mapping depends of the efficiency of the hash function used.

2.3 sort

2.3.1 C++ program to demonstrate descending order sort using

```cpp
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int arr[] = { 1, 5, 8, 9, 6, 7, 3, 4, 2, 0 };
    int n = sizeof(arr) / sizeof(arr[0]);

    sort(arr, arr + n, greater<int>());

    cout << "Array after sorting : \n";
    for (int i = 0; i < n; ++i)
        cout << arr[i] << " ";

    return 0;
}
```

```
Output:
Array after sorting :
9 8 7 6 5 4 3 2 1 0
```

2.3.2 Sorting a vector in C++

```cpp
#include <bits/stdc++.h>
using namespace std;

int main()
{
    vector<int> v{ 1, 5, 8, 9, 6, 7, 3, 4, 2, 0 };

    sort(v.begin(), v.end());

    cout << "Sorted \n";
    for (auto x : v)
        cout << x << " ";

    return 0;
}
```

```
Example Output
Sorted
0 1 2 3 4 5 6 7 8 9
```

2.3.3 vector sort - print value nearest 0

```cpp
#include<bits/stdc++.h>
using namespace std;

int Solve (int N, vector <int> A) {
   // Write your code
   sort(A.begin(), A.end());
   return A[0];
}

int main() {

    ios::sync_with_stdio(0);
    cin.tie(0);
    int N;
    cin >> N;
    vector <int> A(N);
    for (int a_i = 0; a_i < N; ++a_i) {
        cin >> A[a_i];
    }
    int out = Solve(N, A);
    cout << out << "\n";
}
```

```
Example Input
5
```

```
5 6 7 3 9
Example Output
3
```