Sparse Matrix and its representations | Set 1 (Using Arrays and Linked Lists)

A matrix is a two-dimensional data object made of m rows and n columns, therefore having total m x n values. If most of the elements of the matrix have **0 value**, then it is called a sparse matrix.

Why to use Sparse Matrix instead of simple matrix ?

- **Storage:** There are lesser non-zero elements than zeros and thus lesser memory can be used to store only those elements.
- **Computing time:** Computing time can be saved by logically designing a data structure traversing only non-zero elements..

Example:

```
0 0 3 0 4
0 0 5 7 0
0 0 0 0 0
0 2 6 0 0
```

Representing a sparse matrix by a 2D array leads to wastage of lots of memory as zeroes in the matrix are of no use in most of the cases. So, instead of storing zeroes with non-zero elements, we only store non-zero elements. This means storing non-zero elements with **triples- (Row, Column, value).**

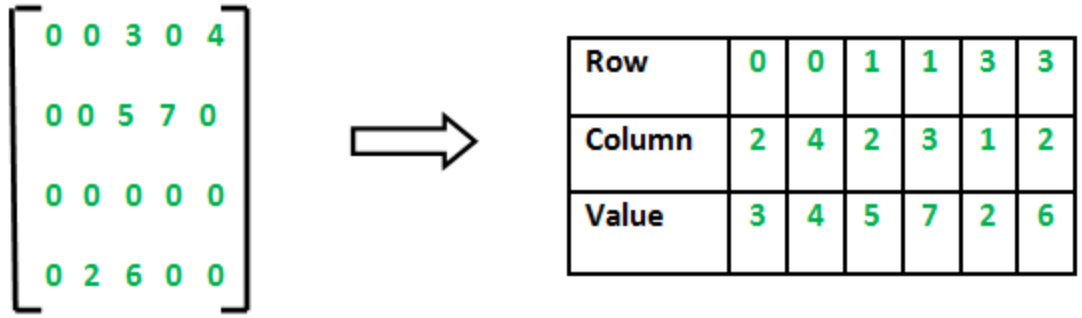
Sparse Matrix Representations can be done in many ways following are two common representations:

1. Array representation
2. Linked list representation

Method 1: Using Arrays

2D array is used to represent a sparse matrix in which there are three rows named as

- **Row:** Index of row, where non-zero element is located
- **Column:** Index of column, where non-zero element is located
- **Value:** Value of the non zero element located at index – (row,column)



- C++
- Java

filter_none

edit
play_arrow
brightness_4

```
// C++ program for Sparse Matrix Representation
// using Array
#include<stdio.h>

int main()
{
    // Assume 4x5 sparse matrix
    int sparseMatrix[4][5] =
    {
        {0 , 0 , 3 , 0 , 4 },
        {0 , 0 , 5 , 7 , 0 },
        {0 , 0 , 0 , 0 , 0 },
        {0 , 2 , 6 , 0 , 0 }
    };

    int size = 0;
    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 5; j++)
            if (sparseMatrix[i][j] != 0)
                size++;

    // number of columns in compactMatrix (size) must be
    // equal to number of non - zero elements in
    // sparseMatrix
    int compactMatrix[3][size];

    // Making of new matrix
    int k = 0;
    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 5; j++)
            if (sparseMatrix[i][j] != 0)
            {
                compactMatrix[0][k] = i;
                compactMatrix[1][k] = j;
                compactMatrix[2][k] = sparseMatrix[i][j];
                k++;
            }

    for (int i=0; i<3; i++)
```

```
    {
        for (int j=0; j<size; j++)
            printf("%d ", compactMatrix[i][j]);

        printf("\n");
    }
    return 0;
}
```

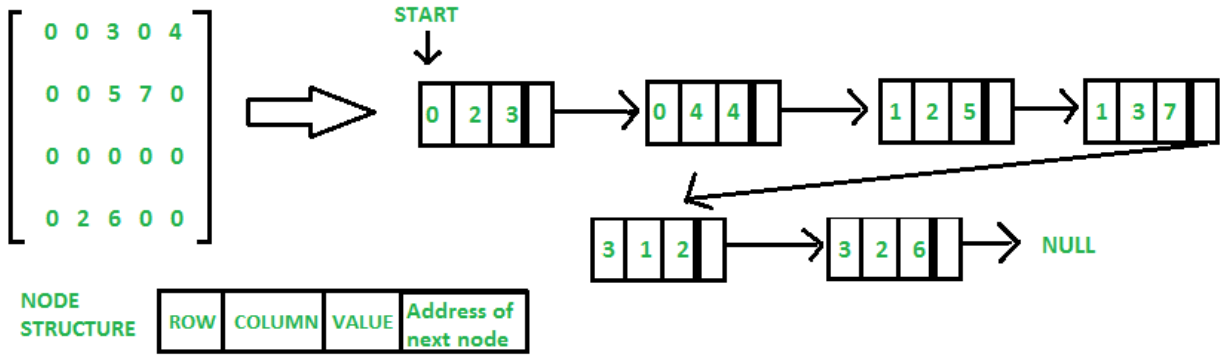
Output:

```
0 0 1 1 3 3
2 4 2 3 1 2
3 4 5 7 2 6
```

Method 2: Using Linked Lists

In linked list, each node has four fields. These four fields are defined as:

- **Row:** Index of row, where non-zero element is located
- **Column:** Index of column, where non-zero element is located
- **Value:** Value of the non zero element located at index – (row,column)
- **Next node:** Address of the next node



filter_none
edit
play_arrow
brightness_4

```
// C program for Sparse Matrix Representation
// using Linked Lists
#include<stdio.h>
#include<stdlib.h>

// Node to represent sparse matrix
struct Node
```

```
{
   int value;
   int row_position;
   int column_postion;
   struct Node *next;
};

// Function to create new node
void create_new_node(struct Node** start, int
non_zero_element,
                  int row_index, int column_index )
{
   struct Node *temp, *r;
   temp = *start;
   if (temp == NULL)
   {
      // Create new node dynamically
      temp = (struct Node *) malloc (sizeof(struct Node));
      temp->value = non_zero_element;
      temp->row_position = row_index;
      temp->column_postion = column_index;
      temp->next = NULL;
      *start = temp;

   }
   else
   {
      while (temp->next != NULL)
         temp = temp->next;

      // Create new node dynamically
      r = (struct Node *) malloc (sizeof(struct Node));
      r->value = non_zero_element;
      r->row_position = row_index;
      r->column_postion = column_index;
      r->next = NULL;
      temp->next = r;

   }
}

// This function prints contents of linked list
// starting from start
```

```
void PrintList(struct Node* start)
{
    struct Node *temp, *r, *s;
    temp = r = s = start;

    printf("row_position: ");
    while(temp != NULL)
    {

        printf("%d ", temp->row_position);
        temp = temp->next;
    }
    printf("\n");

    printf("column_postion: ");
    while(r != NULL)
    {
        printf("%d ", r->column_postion);
        r = r->next;
    }
    printf("\n");
    printf("Value: ");
    while(s != NULL)
    {
        printf("%d ", s->value);
        s = s->next;
    }
    printf("\n");
}


// Driver of the program
int main()
{
   // Assume 4x5 sparse matrix
    int sparseMatric[4][5] =
    {
        {0 , 0 , 3 , 0 , 4 },
        {0 , 0 , 5 , 7 , 0 },
        {0 , 0 , 0 , 0 , 0 },
        {0 , 2 , 6 , 0 , 0 }
    };
```

```
    /* Start with the empty list */
    struct Node* start = NULL;

    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 5; j++)

            // Pass only those values which are non - zero
            if (sparseMatric[i][j] != 0)
                create_new_node(&start, sparseMatric[i][j], i, j);

    PrintList(start);

    return 0;
}
```

Output:

```
row_position: 0 0 1 1 3 3

column_postion: 2 4 2 3 1 2

Value: 3 4 5 7 2 6
```

Other representations:

As a **Dictionary** where row and column numbers are used as keys and values are matrix entries. This method saves space but sequential access of items is costly.
As a **list of list**. The idea is to make a list of rows and every item of list contains values. We can keep list items sorted by column numbers.
Sparse Matrix and its representations | Set 2 (Using List of Lists and Dictionary of keys)

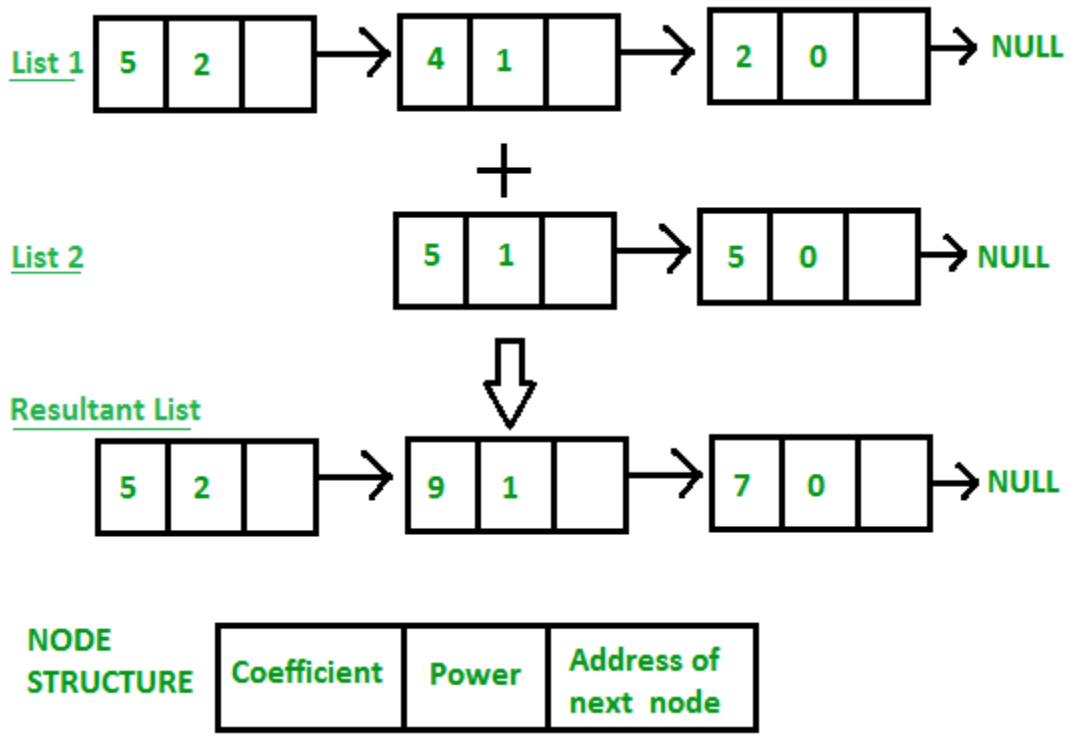
Adding two polynomials using Linked List

Given two polynomial numbers represented by a linked list. Write a function that add these lists means add the coefficients who have same variable powers.

Example:

```
Input:
     1st number = 5x^2 + 4x^1 + 2x^0
     2nd number = 5x^1 + 5x^0
Output:
        5x^2 + 9x^1 + 7x^0
Input:
     1st number = 5x^3 + 4x^2 + 2x^0
```

```
     2nd number = 5x^1 + 5x^0
Output:
        5x^3 + 4x^2 + 5x^1 + 7x^0
```



Program to add two polynomials

Given two polynomials represented by two arrays, write a function that adds given two polynomials.

Example:

```
Input:  A[] = {5, 0, 10, 6}
        B[] = {1, 2, 4}
Output: sum[] = {6, 2, 14, 6}

The first input array represents "5 + 0x^1 + 10x^2 + 6x^3"
The second array represents "1 + 2x^1 + 4x^2"
And Output is "6 + 2x^1 + 14x^2 + 6x^3"
```

Addition is simpler than multiplication of polynomials. We initialize result as one of the two polynomials, then we traverse the other polynomial and add all terms to the result.

```
add(A[0..m-1], B[0..n01])
1) Create a sum array sum[] of size equal to maximum of 'm' and 'n'
2) Copy A[] to sum[].
3) Travers array B[] and do following for every element B[i]
        sum[i] = sum[i] + B[i]
4) Return sum[].
```

The following is implementation of above algorithm.

- C++

```
// Simple C++ program to add two polynomials
#include <iostream>
using namespace std;
```

```
// A utility function to return maximum of two integers
int max(int m, int n) {  return (m > n)? m: n; }

// A[] represents coefficients of first polynomial
// B[] represents coefficients of second polynomial
// m and n are sizes of A[] and B[] respectively
int *add(int A[], int B[], int m, int n)
{
   int size = max(m, n);
   int *sum = new int[size];

   // Initialize the porduct polynomial
   for (int i = 0; i<m; i++)
     sum[i] = A[i];

   // Take ever term of first polynomial
   for (int i=0; i<n; i++)
       sum[i] += B[i];

   return sum;
}

// A utility function to print a polynomial
void printPoly(int poly[], int n)
{
    for (int i=0; i<n; i++)
    {
       cout << poly[i];
       if (i != 0)
        cout << "x^" << i ;
       if (i != n-1)
       cout << " + ";
    }
}

// Driver program to test above functions
int main()
{
    // The following array represents polynomial 5 +
10x^2 + 6x^3
    int A[] = {5, 0, 10, 6};

    // The following array represents polynomial 1 + 2x
+ 4x^2
    int B[] = {1, 2, 4};
    int m = sizeof(A)/sizeof(A[0]);
    int n = sizeof(B)/sizeof(B[0]);

    cout << "First polynomial is \n";
    printPoly(A, m);
    cout << "\nSecond polynomial is \n";
    printPoly(B, n);

    int *sum = add(A, B, m, n);
    int size = max(m, n);

    cout << "\nsum polynomial is \n";
```

```
    printPoly(sum, size);

    return 0;
}
```

Output:

```
First polynomial is
5 + 0x^1 + 10x^2 + 6x^3
Second polynomial is
1 + 2x^1 + 4x^2
Sum polynomial is
6 + 2x^1 + 14x^2 + 6x^3
```

Time complexity of the above algorithm and program is O(m+n) where m and n are orders of two given polynomials.

C PROGRAM FOR POLYNOMIAL ADDITION USING ARRAYS

```
#include<stdio.h>
void main()
{
int poly1[6][2],poly2[6][2],term1,term2,match,proceed,i,j;
printf("Enter the number of terms in first polynomial : ");
scanf("%d",&term1);
printf("Enter the number of terms in second polynomial : ");
scanf("%d",&term2);
printf("Enter the coeff and expo of the first polynomial:\n");
for(i=0;i<term1;i++)
{
scanf("%d %d",&poly1[i][0],&poly1[i][1]);
}
printf("Enter the coeff and expo of the second polynomial:\n");
for(i=0;i<term2;i++)
{
scanf("%d %d",&poly2[i][0],&poly2[i][1]);
}
printf("The resultant polynomial after addition :\n");
for(i=0;i<term1;i++)
{
match=0;
for(j=0;j<term2;j++)
{
if(match==0)
if(poly1[i][1]==poly2[j][1])
{
printf("%d %d\n",(poly1[i][0]+poly2[j][0]), poly1[i][1]);
match=1;
```

```
}
}
}
for(i=0;i<term1;i++)
{
proceed=1;
for(j=0;j<term2;j++)
{
if(proceed==1)
if(poly1[i][1]!=poly2[j][1])
proceed=1;
else
proceed=0;
}
if(proceed==1)
printf("%d %d\n",poly1[i][0],poly1[i][1]);
}
for(i=0;i<term2;i++)
{
proceed=1;
for(j=0;j<term1;j++)
{
if(proceed==1)
if(poly2[i][1]!=poly1[j][1])
proceed=1;
else
proceed=0;
}
if(proceed==1)
printf("%d %d",poly2[i][0],poly2[i][1]);
}
getch();
}
```

SORTINGS:

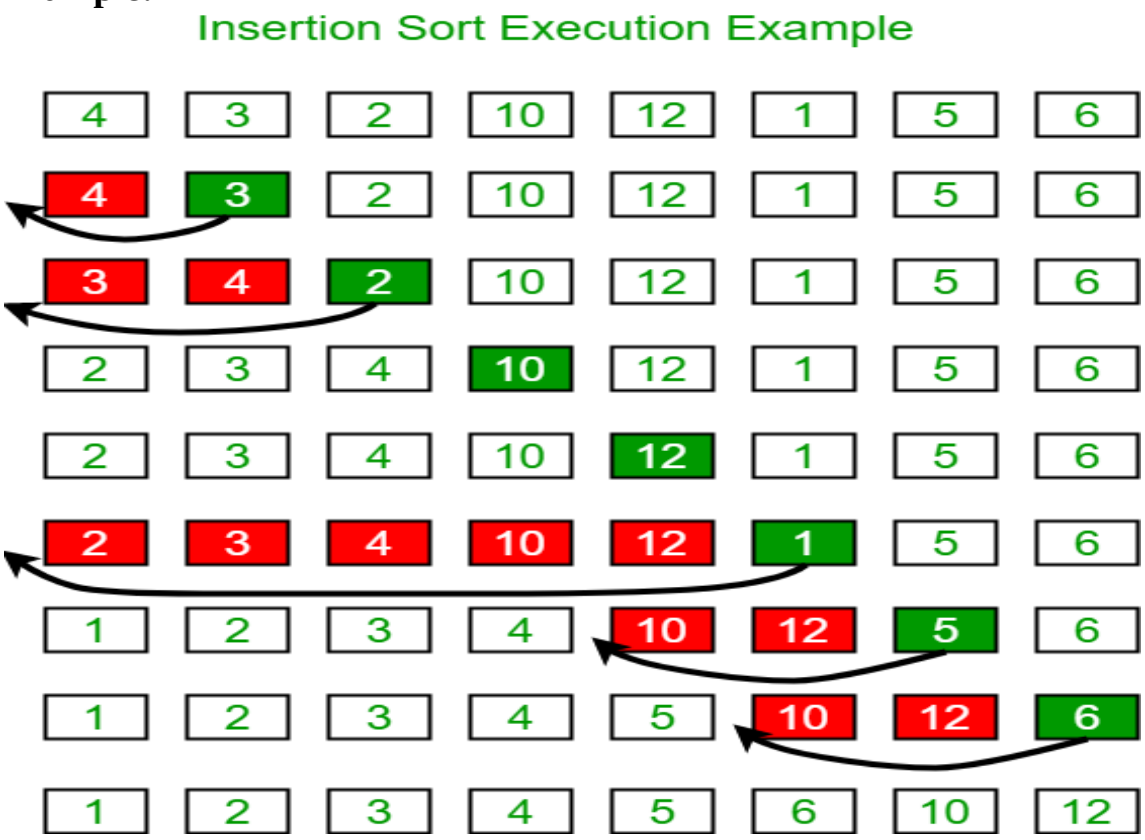
Insertion Sort

Insertion sort is a simple sorting algorithm that works the way we sort playing cards in our hands.

Algorithm

// Sort an arr[] of size n
insertionSort(arr, n)
Loop from i = 1 to n-1.
……a) Pick element arr[i] and insert it into sorted sequence arr[0…i-1]

Example:



Another Example:

12, 11, 13, 5, 6

Let us loop for i = 1 (second element of the array) to 4 (last element of the array)

i = 1. Since 11 is smaller than 12, move 12 and insert 11 before 12

11, 12, 13, 5, 6

i = 2. 13 will remain at its position as all elements in A[0..I-1] are smaller than 13

11, 12, 13, 5, 6

i = 3. 5 will move to the beginning and all other elements from 11 to 13 will move one position ahead of their current position.

5, 11, 12, 13, 6

i = 4. 6 will move to position after 5, and elements from 11 to 13 will move one position ahead of their current position.

5, 6, 11, 12, 13

```
// C program for insertion sort
#include <math.h>
#include <stdio.h>
```

```
/* Function to sort an array using insertion sort*/
void insertionSort(int arr[], int n)
{
   int i, key, j;
   for (i = 1; i < n; i++) {
      key = arr[i];
      j = i - 1;

      /* Move elements of arr[0..i-1], that are
        greater than key, to one position ahead
        of their current position */
      while (j >= 0 && arr[j] > key) {
         arr[j + 1] = arr[j];
         j = j - 1;
      }
      arr[j + 1] = key;
   }
}

// A utility function to print an array of size n
void printArray(int arr[], int n)
{
   int i;
   for (i = 0; i < n; i++)
      printf("%d ", arr[i]);
   printf("\n");
}

/* Driver program to test insertion sort */
int main()
{
   int arr[] = { 12, 11, 13, 5, 6 };
   int n = sizeof(arr) / sizeof(arr[0]);

   insertionSort(arr, n);
   printArray(arr, n);

   return 0;
}
```

Output:

```
5 6 11 12 13
```

Radix Sort

The lower bound for Comparison based sorting algorithm (Merge Sort, Heap Sort, Quick-Sort .. etc) is Ω(nLogn), i.e., they cannot do better than nLogn. Counting sort is a linear time sorting algorithm that sort in O(n+k) time when elements are in range from 1 to k.

What if the elements are in range from 1 to n^2?

We can't use counting sort because counting sort will take $O(n^2)$ which is worse than comparison based sorting algorithms. Can we sort such an array in linear time?

Radix Sort is the answer. The idea of Radix Sort is to do digit by digit sort starting from least significant digit to most significant digit. Radix sort uses counting sort as a subroutine to sort.

The Radix Sort Algorithm

1) Do following for each digit i where i varies from least significant digit to the most significant digit.

…….**a)** Sort input array using counting sort (or any stable sort) according to the i'th digit.

Example:

Original, unsorted list:

170, 45, 75, 90, 802, 24, 2, 66

Sorting by least significant digit (1s place) gives: [*Notice that we keep 802 before 2, because 802 occurred before 2 in the original list, and similarly for pairs 170 & 90 and 45 & 75.]

170, 90, 802, 2, 24, 45, 75, 66

Sorting by next digit (10s place) gives: [*Notice that 802 again comes before 2 as 802 comes before 2 in the previous list.]

802, 2, 24, 45, 66, 170, 75, 90

Sorting by most significant digit (100s place) gives:

2, 24, 45, 66, 75, 90, 170, 802

What is the running time of Radix Sort?

Let there be d digits in input integers. Radix Sort takes O(d*(n+b)) time where b is the base for representing numbers, for example, for decimal system, b is 10. What is the value of d? If k is the maximum possible value, then d would be $O(\log_b(k))$. So overall time complexity is $O((n+b) * \log_b(k))$.

Which looks more than the time complexity of comparison based sorting algorithms for a large k. Let us first limit k. Let $k <= n^c$ where c is a constant. In that case, the complexity becomes $O(nLog_b(n))$. But it still doesn't beat comparison based sorting algorithms.

What if we make value of b larger?. What should be the value of b to make the time complexity linear? If we set b as n, we get the time complexity as O(n). In other words, we can sort an array of integers with range from 1 to n^c if the numbers are represented in base n (or every digit takes $log_2(n)$ bits).

Is Radix Sort preferable to Comparison based sorting algorithms like Quick-Sort?

If we have log_2n bits for every digit, the running time of Radix appears to be better than Quick Sort for a wide range of input numbers. The constant factors hidden in asymptotic notation are higher for Radix Sort and Quick-Sort uses hardware caches more effectively. Also, Radix sort uses counting sort as a subroutine and counting sort takes extra space to sort numbers.

Address Calculation sort

- This technique uses hasing function for sorting the elements. Any hashing function f(x) that can be used for sorting should have this property.

  If X<Y, then f(X)<=f(y)
- These types of functions are called non decreasing functions or order preserving hashing functions.
- This function can be applied to each element, and according to the value of the hashing function each element is placed in a particular set.
- When two elements are to be placed in the same set, then they are placed in the same set, then they are placed in sorted order. Now we will take some numbers and sort them using address calculation sort.
- 194,289,566,432,654,98,232,415,345,276,532,254,165,965,476
- Let us take a function f(x) whose value is equal to the first digit of X, this will obviously be a non decreasing function because if first digit of any number A is less than the first digit of any number B, A will definitely be less then B.

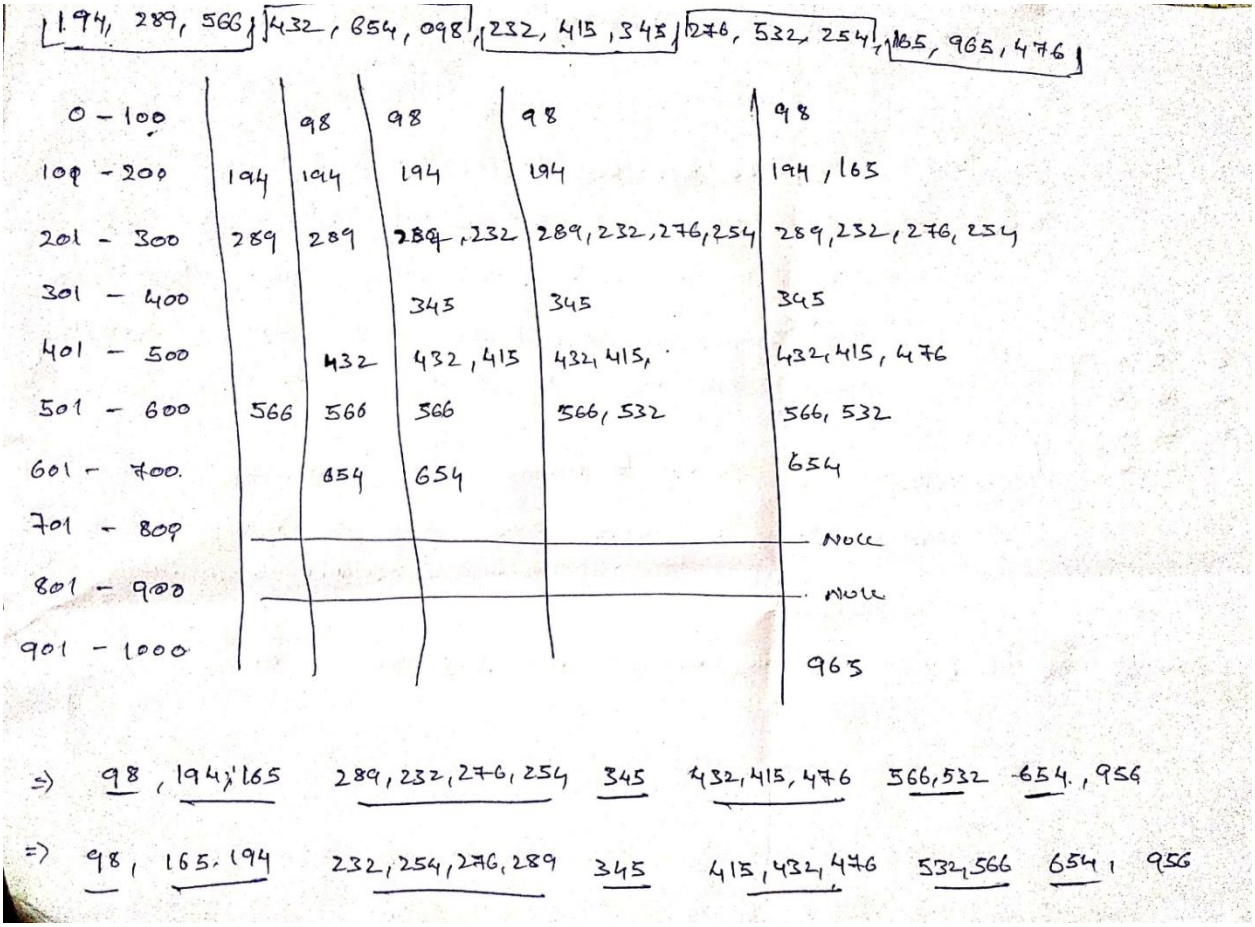
x	194	289	566	432	654	098	232	415	345	276	532	254	165	965	476
f(x)	1	2	5	4	6	0	2	4	3	2	5	2	1	9	4

Now all the elements will be placed in different sets according to the corresponding values of f(x). The value of function f(x) can be 0, 1, 2....., 9 so there will be 10 sets into which these elements are inserted.

0	098
1	165, 194
2	232, 254, 276, 289
3	345
4	415, 432, 476
5	532, 566
6	654
7	
8	
9	965

We can see that the value of f(x) for the elements 289, 232, 276, 254 is same, so they are inserted in the same set but in sorted order i.e. 232, 254, 276, 289. Similarly other elements are also inserted in their particular sets in sorted order. All the elements in a set are less than elements of the next set and elements in a set are in sorted order. So if we concatenate all the sets then we will get the sorted list.

98, 165, 194, 232, 254. 276, 289, 345, 415, 432, 476, 532, 566, 654, 965



TOWERS OF HANOI

Program for Tower of Hanoi

Tower of Hanoi is a mathematical puzzle where we have three rods and n disks. The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:
1) Only one disk can be moved at a time.
2) Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
3) No disk may be placed on top of a smaller disk.

Approach :

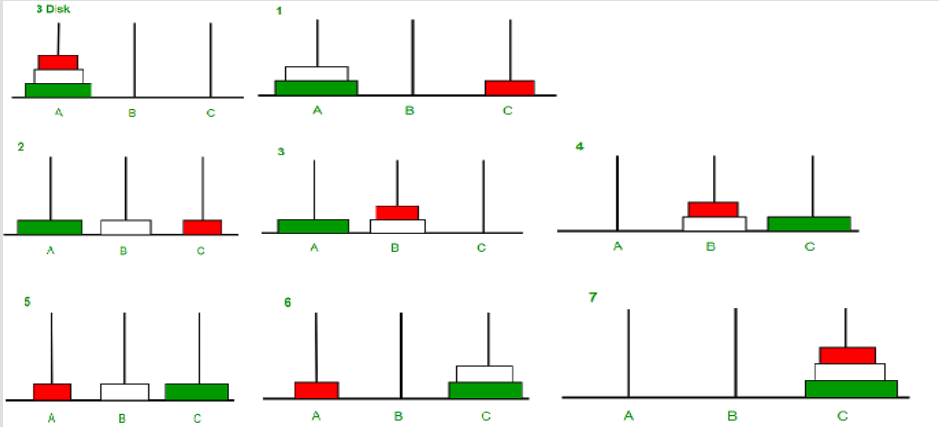
```
Take an example for 2 disks :
Let rod 1 = 'A', rod 2 = 'B', rod 3 = 'C'.

Step 1 : Shift first disk from 'A' to 'B'.
```

```
Step 2 : Shift second disk from 'A' to 'C'.
Step 3 : Shift first disk from 'B' to 'C'.

The pattern here is :
Shift 'n-1' disks from 'A' to 'B'.
Shift last disk from 'A' to 'C'.
Shift 'n-1' disks from 'B' to 'C'.

Image illustration for 3 disks :
```



Examples:

```
Input : 2
Output : Disk 1 moved from A to B
         Disk 2 moved from A to C
         Disk 1 moved from B to C

Input : 3
Output : Disk 1 moved from A to C
         Disk 2 moved from A to B
         Disk 1 moved from C to B
         Disk 3 moved from A to C
         Disk 1 moved from B to A
         Disk 2 moved from B to C
         Disk 1 moved from A to C
```

/*IMPLEMENTATION OF TOWERS OF HANOI USING C*/

#include<stdio.h>

int hanoi(int,char,char,char);

void main()

```
{
int n,c;
printf("Enter number of disks:");
scanf("%d",&n);
c=hanoi(n,'A','B','C');
printf("Number of steps required is:%d",c);

}
int hanoi(int n,char beg,char mid,char end)
{
static int count;
if(n>0)
{
hanoi(n-1,beg,end,mid);
printf("Move disk %d from %c to %c \n",n,beg,end);
count++;
hanoi(n-1,mid,beg,end);
 }
return(count);
}
```