Dr. B.R Ambedkar National Institute of Technology, Jalandhar

Introduction to Design and analysis of algorithms Lab
(ITPC - 222)

Submitted to	***Submitted By***
Dr. Mohit Kumar	Aditya Anand
Department of IT	20124009 IT (G1)

Table of contents

8.	Fibonacci Heap a) Make Heap b) Insert c) Find Min d) Extract Min e) Union f) Decrease a key g) Delete	April 7, 2022		
9.	Red Black Tree	April 14, 2022		
10.	Greedy Algorithm a) Deadline based job scheduling b) Activity Selection Problem c) Huffman Code d) Kruskals Algorithm(MST) e) Fractional Knapsack	April 21, 2022		

Lab – 1

S No	Program Title	Date of Implementation	Remarks
i.	Linear Search	Jan 6, 2022	
ii.	Binary Search	Jan 6, 2022	
iii.	Selection Sort	Jan 6, 2022	

a) Linear Search

Input: Array of integer

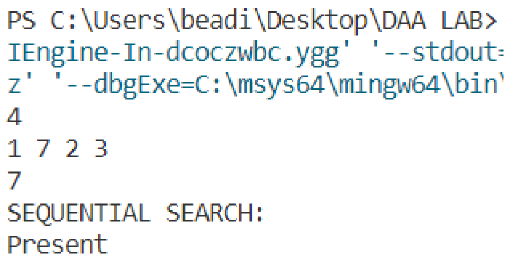
Output: Index of element to be search in input array

Time Complexity: O(n)

Code:

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

// O(n)
bool seq_search(vector<int> v, int target){
    for(int i=0; i<v.size(); i++){
        if(target==v[i]){
            return true;
        }
    }
    return false;
}

int main(){
    int n = 0;
    cin>>n;
    vector<int> v(n, 0);
    for(int i=0; i<n; i++){
        cin>>v[i];
    }

    int target = 0;
    cin>>target;
    cout<<"SEQUENTIAL SEARCH:\n";
    if(seq_search(v, target)){
        cout<<"Present\n";
    }
    else{
        cout<<"Not Present\n";
    }

    return 0;
}
```

Output:



```
PS C:\Users\beadi\Desktop\DAA LAB>
IEngine-In-dcoczwbc.ygg' '--stdout=
z' '--dbgExe=C:\msys64\mingw64\bin'
4
1 7 2 3
7
SEQUENTIAL SEARCH:
Present
```

b) Binary Search

Input: Array of integer

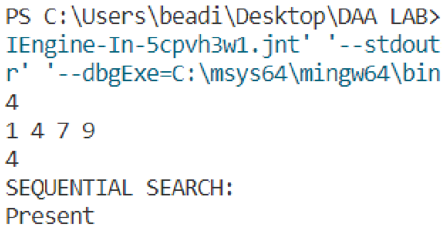
Output: Index of element to be search in sorted input array

Time Complexity: O(log(n))

Code:

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

// O(log(n))
bool binarySearch(vector<int> v, int target){
    int i=0, j=v.size()-1;
    while(j>=i){
        int mid = i + (j-i)/2;
        if(v[mid]==target){
            return true;
        }
        else if(v[mid]>target){
            j=mid-1;
        }
        else{
            i=mid+1;
        }
    }
    return false;
}

int main(){
    int n = 0;
    cin>>n;
    vector<int> v(n, 0);
    for(int i=0; i<n; i++){
        cin>>v[i];
    }

    int target = 0;
    cin>>target;

    cout<<"BINARY SEARCH:\n";
    if(binarySearch(v, target)){
        cout<<"Present\n";
    }
    else{
        cout<<"Not Present\n";
    }

    return 0;
}
```

Output:



```
PS C:\Users\beadi\Desktop\DAA LAB>
IEngine-In-5cpvh3w1.jnt' '--stdout
r' '--dbgExe=C:\msys64\mingw64\bin
4
1 4 7 9
4
SEQUENTIAL SEARCH:
Present
```

c) Selection Sort

Input: Array of integer

Output: Index of element to be search in input array

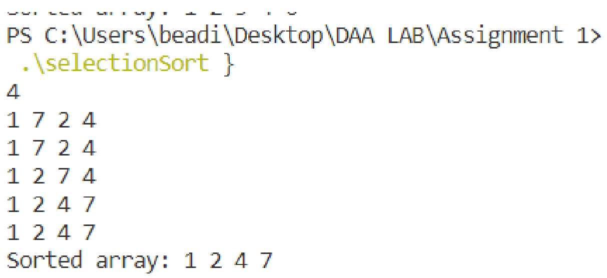
Time Complexity: O(n^2)

Code:

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

```cpp
// O(n^2)
void selection_sort(vector<int> &v){
    // Array from 0 to i-1 is sorted, i to n is unsorted
    int i=0;
    while(i<v.size()){
        int mn = INT_MAX;
        int id = -1;
        for(int j=i; j<v.size(); j++){
            if(mn>v[j]){
                mn = v[j];
                id = j;
            }
        }
        swap(v[i], v[id]);
        i++;
        for(auto ele:v){
            cout<<ele<<" ";
        }
        cout<<"\n";
    }
}

int main(){
    int n=0;
    cin>>n;
    vector<int> v(n, 0);
    for(int i=0; i<n; i++){
        cin>>v[i];
    }
    selection_sort(v);
    cout<<"Sorted array: ";
    for(auto i:v){
        cout<<i<<" ";
    }
    return 0;
}
```

Output:



```
PS C:\Users\beadi\Desktop\DAA LAB\Assignment 1>
 .\selectionSort }
4
1 7 2 4
1 7 2 4
1 2 7 4
1 2 4 7
1 2 4 7
Sorted array: 1 2 4 7
```

Lab – 2

S No	Program Title	Date of Implementation	Remarks
i.	Insertion Sort	Jan 31, 2022	
ii.	Quick sort	Jan 31, 2022	

a) Insertion Sort

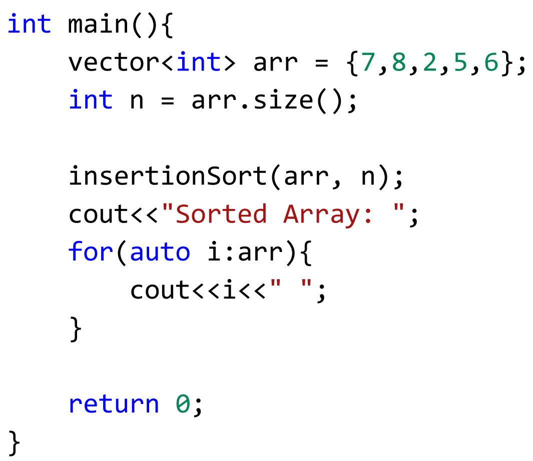
Input: Array of integer

Output: Sorted array of integer

Time Complexity: O(n^2)

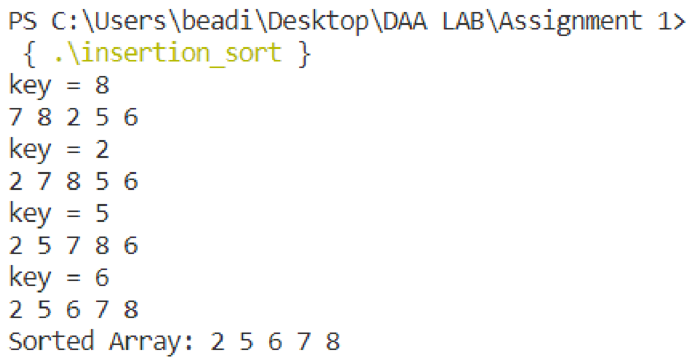
Code:

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

void insertionSort(vector<int> &arr, int n){
    int i, key, j;
    for (i = 1; i < n; i++){
        key = arr[i];
        cout<<"key = "<<key<<"\n";
        j = i - 1;

        while (j >= 0 && arr[j] > key){
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;

        for(auto i: arr){
            cout<<i<<" ";
        }
        cout<<"\n";
    }
    return;
}
```



```
int main(){
    vector<int> arr = {7,8,2,5,6};
    int n = arr.size();

    insertionSort(arr, n);
    cout<<"Sorted Array: ";
    for(auto i:arr){
        cout<<i<<" ";
    }

    return 0;
}
```

Output:



```
PS C:\Users\beadi\Desktop\DAA LAB\Assignment 1>
 { .\insertion_sort }
key = 8
7 8 2 5 6
key = 2
2 7 8 5 6
key = 5
2 5 7 8 6
key = 6
2 5 6 7 8
Sorted Array: 2 5 6 7 8
```

b) Quick Sort

Input: Array of integer

Output: Sorted array of integer

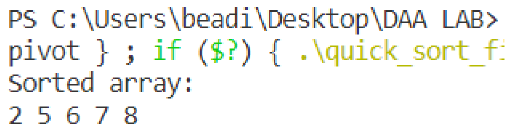
Time Complexity:

Best case: Ω(n*log(n))   Average case: θ(n*log(n))   Worst case: O(n^2)

Code:

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

int N;

void printArray(int arr[], int size)
{
    int i;
    for (i = 0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}

int partition1(int a[], int start, int end)
{

    int pivot = a[start], p1 = start + 1, i, temp;

    for (i = start + 1; i <= end; i++)
    {

        if (a[i] < pivot)
        {
            if (i != p1)
            {
                temp = a[p1];
                a[p1] = a[i];
                a[i] = temp;
            }
            p1++;
        }
    }

    a[start] = a[p1 - 1];
    a[p1 - 1] = pivot;

    return p1 - 1;
}

void quicksort(int *a, int start, int end)
{
    int p1;
    if (start < end)
    {
        p1 = partition1(a, start, end);
        quicksort(a, start, p1 - 1);
        quicksort(a, p1 + 1, end);
    }
}

int main()
{
    int arr[] = {7, 8, 2, 5, 6};
    int n = sizeof(arr) / sizeof(arr[0]);
    N = n;
    quicksort(arr, 0, n - 1);
    cout << "Sorted array: \n";
    printArray(arr, n);
    return 0;
}
```

Output:



```
PS C:\Users\beadi\Desktop\DAA LAB>
pivot } ; if ($?) { .\quick_sort_f
Sorted array:
2 5 6 7 8
```

Lab – 3

S No	Program Title	Date of Implementation	Remarks
i.	Merge Sort	Feb 07, 2022	

Merge Sort

Input: Array of integer

Output: Sorted array of integer

Time Complexity: O(n*log(n))

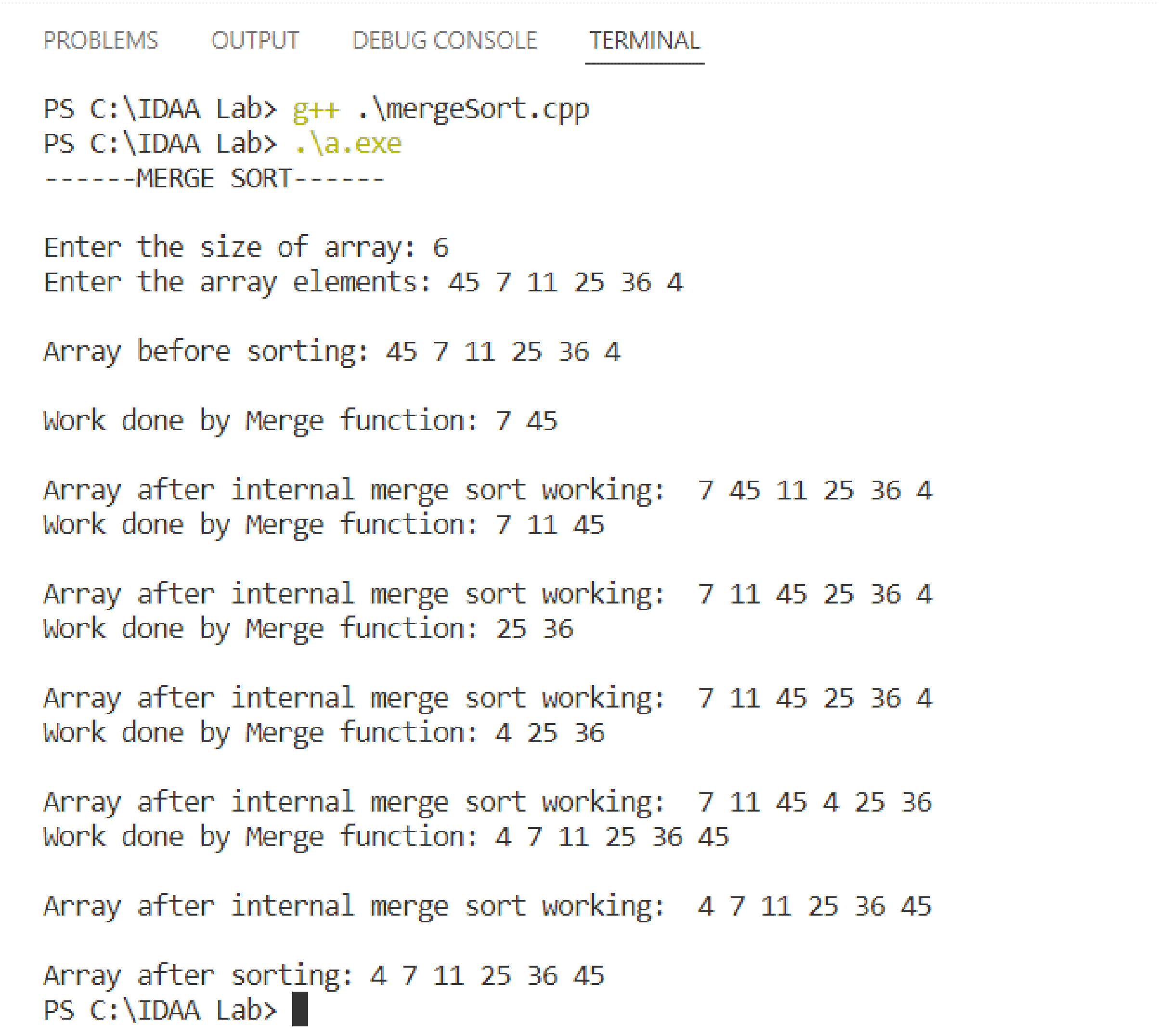
Code:

```
#include <iostream>
using namespace std;
void merge(int *arr, int low, int mid, int high)
{
    int n1 = mid - low + 1, n2 = high - mid;
    int left[n1], right[n2];
    for (int i = 0; i < n1; i++) // making auxiliary arrays left and right
        left[i] = arr[i + low];
    for (int i = 0; i < n2; i++)
        right[i] = arr[mid + 1 + i];

    int i = 0, j = 0, k = low;
    for (; i < n1 && j < n2;)
    {
        if (left[i] <= right[j])
        {
            arr[k] = left[i];
            i++;
            k++;
        }
        else
        {
            arr[k] = right[j];
            k++;
            j++;
        }
    }

    while (i < n1)
```

```cpp
    { // if i<n1 this loop run
        arr[k] = left[i];
        i++, k++;
    }
    while (j < n2)
    { // else if j<n2 this loop run
        arr[k] = right[j];
        k++, j++;
    }

    cout << "Work done by Merge function: ";
    for (int i = low; i <= high; i++)
    {
        cout << arr[i] << " ";
    }
    cout << endl
         << endl;
}

void mergeSort(int *arr, int left, int right)
{
    static int size = right + 1;
    if (left < right)
    {
        int mid = (left + right) / 2;
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
        cout << "Array after internal merge sort working:  ";
        for (int i = 0; i < size; i++)
        {
            cout << arr[i] << " ";
        }
        cout << endl;
    }
}

int main()
{

    cout << "------MERGE SORT------" << endl
         << endl;
    int n = 0;
    cout << "Enter the size of array: ";
    cin >> n;
    int arr[n];
    cout << "Enter the array elements: ";
    for (int i = 0; i < n; i++)
```

```
        cin >> arr[i];

    cout << "\nArray before sorting: ";
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl
         << endl;
    mergeSort(arr, 0, n - 1);
    cout << "\nArray after sorting: ";
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";

    return 0;
}
```

Output:



```
PS C:\IDAA Lab> g++ .\mergeSort.cpp
PS C:\IDAA Lab> .\a.exe
------MERGE SORT------

Enter the size of array: 6
Enter the array elements: 45 7 11 25 36 4

Array before sorting: 45 7 11 25 36 4

Work done by Merge function: 7 45

Array after internal merge sort working:  7 45 11 25 36 4
Work done by Merge function: 7 11 45

Array after internal merge sort working:  7 11 45 25 36 4
Work done by Merge function: 25 36

Array after internal merge sort working:  7 11 45 25 36 4
Work done by Merge function: 4 25 36

Array after internal merge sort working:  7 11 45 4 25 36
Work done by Merge function: 4 7 11 25 36 45

Array after internal merge sort working:  4 7 11 25 36 45

Array after sorting: 4 7 11 25 36 45
PS C:\IDAA Lab> █
```

Lab – 4

S No	Program Title	Date of Implementation	Remarks
i.	Binary Search using Recusion	Feb 14, 2022	
ii.	Factorial using Recusion	Feb 14, 2022	
iii.	Fibonacci series using Recursion	Feb 14, 2022	
iv.	Sum of n Natural numbers using Recusion	Feb 14, 2022	

a) Binary Search

Input: Array of integer

Output: Index of element to be search in sorted input array

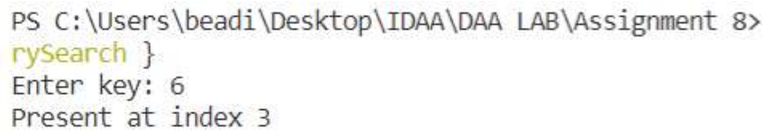
Time Complexity: O(log(n))

Code:

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

int binarySearch(vector<int> v, int s, int e, int key){
    if(s>e){
        return -1;
    }

    int mid = s+(e-s)/2;
    if(v[mid]==key){
        return mid;
    }
    else if(v[mid]>key){
        return binarySearch(v, s, mid-1, key);
    }
    else{
        return binarySearch(v, mid+1, e, key);
    }
}
```

```
int main(){
    vector<int> v = {1,2,3,6,8,9};
    int n = v.size();

    cout<<"Enter key: ";
    int key=0;
    cin>>key;
    int idx = binarySearch(v, 0, n-1, key);
    if(idx!=-1){
        cout<<"Present at index "<<idx;
    }
    else{
        cout<<"Element is not present";
    }
    return 0;
}
```

Output:



```
PS C:\Users\beadi\Desktop\IDAA\DAA LAB\Assignment 8>
rySearch }
Enter key: 6
Present at index 3
```

b) Factorial

Input: Integer

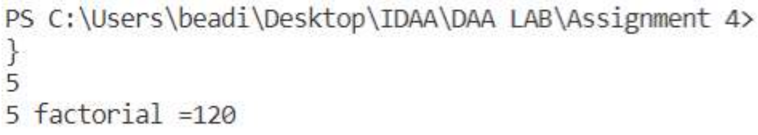
Output: Factorial of input integer

Time Complexity: O(n)

Code:

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

int factorial(int n){
    if(n==0){
        return 1;
    }

    return n*factorial(n-1);
}

int main(){
    int n=0;
    cin>>n;
    cout<<n<<" factorial ="<<factorial(n)<<"\n";
    return 0;
}
```

Output:



```
PS C:\Users\beadi\Desktop\IDAA\DAA LAB\Assignment 4>
}
5
5 factorial =120
```

c) Fibonacci

Input: Integer

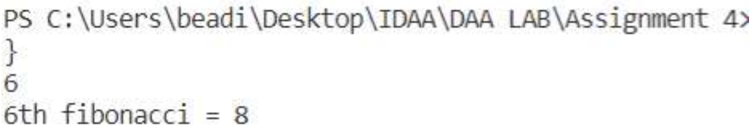
Output: Fibonacci series having length equal to Input integer

Time Complexity: O(n)

Code:

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

int fibonacci(int n){
    if(n==0 || n==1){
        return n;
    }

    return fibonacci(n-1)+fibonacci(n-2);
}

int main(){
    int n = 0;
    cin>>n;
    cout<<n<<"th fibonacci = "<<fibonacci(n);
    return 0;
}
```

Output:



```
PS C:\Users\beadi\Desktop\IDAA\DAA LAB\Assignment 4>
}
6
6th fibonacci = 8
```

d) Sum of n Natural numbers

Input: Integer

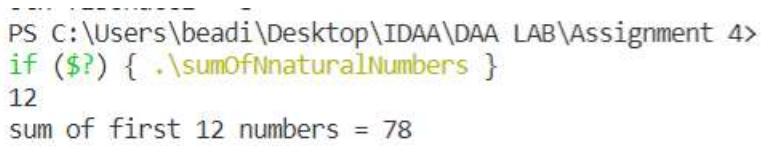
Output: Sum from 1 to input integer value

Time Complexity: O(n)

Code:

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

int sum(int n){
    if(n==1){
        return 1;
    }

    return n+sum(n-1);
}

int main(){
    int n=0;
    cin>>n;
    cout<<"sum of first "<<n<< " numbers = "<<sum(n);
    return 0;
}
```

Output:



```
PS C:\Users\beadi\Desktop\IDAA\DAA LAB\Assignment 4>
if ($?) { .\sumOfNnaturalNumbers }
12
sum of first 12 numbers = 78
```

Lab – 5

S No	Program Title	Date of Implementation	Remarks
i.	Matrix Multiplication using recursion	March 24, 2022	

Matrix Multiplication

Input: 2 Matrix of integers

Output: Product of matrices in matrix form

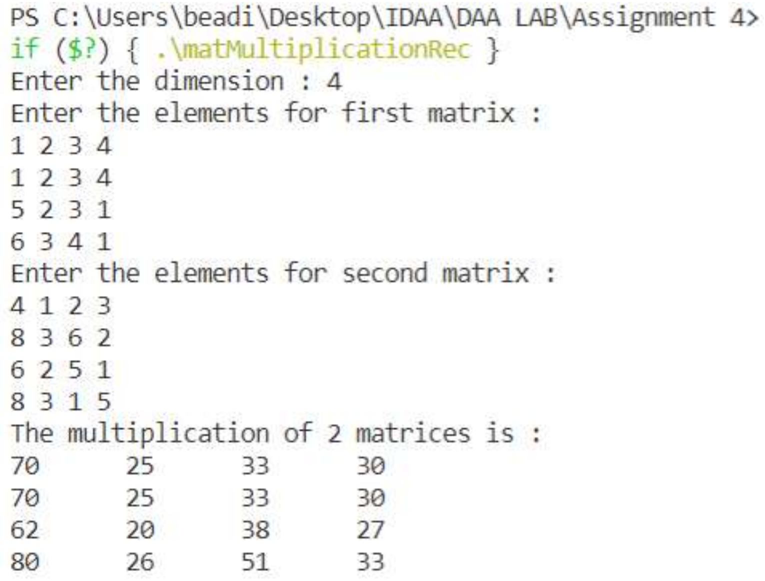
Time Complexity: O(n^2.81)

Code:

```
#include <bits/stdc++.h>
using namespace std;
int **setAllZero(int n)
{
    int **res = new int *[n];
    for (int i = 0; i < n; i++)
    {
        res[i] = new int[n];
        for (int j = 0; j < n; j++)
        {
            res[i][j] = 0;
        }
    }
    return res;
}
int **matrixAddition(int **arr, int **arr1, int n)
{
    int **res = setAllZero(n);
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            res[i][j] = arr[i][j] + arr1[i][j];
        }
    }
    return res;
}
int **matrixSubtraction(int **arr, int **arr1, int n)
{
    int **res = setAllZero(n);
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            res[i][j] = arr[i][j] - arr1[i][j];
        }
    }
    return res;
}
int **matrixMultiplication(int **arr, int **arr1, int n)
{
    int **res = setAllZero(n);
    if (n == 1)
    {
        res[0][0] = arr[0][0] * arr1[0][0];
        return res;
    }
    int **a11 = setAllZero(n / 2);
    int **a12 = setAllZero(n / 2);
    int **a21 = setAllZero(n / 2);
    int **a22 = setAllZero(n / 2);
    int **b11 = setAllZero(n / 2);
    int **b12 = setAllZero(n / 2);
    int **b21 = setAllZero(n / 2);
    int **b22 = setAllZero(n / 2);
    for (int i = 0; i < n / 2; i++)
    {
        for (int j = 0; j < n / 2; j++)
        {
            a11[i][j] = arr[i][j];
            a12[i][j] = arr[i][n / 2 + j];
            a21[i][j] = arr[i + n / 2][j];
            a22[i][j] = arr[i + n / 2][j + n / 2];
            b11[i][j] = arr1[i][j];
            b12[i][j] = arr1[i][n / 2 + j];
            b21[i][j] = arr1[i + n / 2][j];
            b22[i][j] = arr1[i + n / 2][j + n / 2];
        }
    }
    int **p = matrixMultiplication(matrixAddition(a11, a22, n / 2),
matrixAddition(b11, b22, n / 2), n / 2);
    int **q = matrixMultiplication(matrixAddition(a21, a22, n / 2), b11, n /
2);
    int **r = matrixMultiplication(a11, matrixSubtraction(b12, b22, n / 2), n
/ 2);
    int **s = matrixMultiplication(a22, matrixSubtraction(b21, b11, n / 2), n
/ 2);
```

```
    int **t = matrixMultiplication(matrixAddition(a11, a12, n / 2), b22, n /
2);
    int **u = matrixMultiplication(matrixSubtraction(a21, a11, n / 2),
matrixAddition(b11, b12, n / 2), n / 2);
    int **v = matrixMultiplication(matrixSubtraction(a12, a22, n / 2),
matrixAddition(b21, b22, n / 2), n / 2);
    int **c11 = matrixAddition(p, matrixAddition(v, matrixSubtraction(s, t, n
/ 2), n / 2), n / 2);
    int **c12 = matrixAddition(r, t, n / 2);
    int **c21 = matrixAddition(q, s, n / 2);
    int **c22 = matrixAddition(p, matrixAddition(u, matrixSubtraction(r, q, n
/ 2), n / 2), n / 2);
    for (int i = 0; i < n / 2; i++)
    {
        for (int j = 0; j < n / 2; j++)
        {
            res[i][j] = c11[i][j];
            res[i][j + n / 2] = c12[i][j];
            res[i + n / 2][j] = c21[i][j];
            res[i + n / 2][j + n / 2] = c22[i][j];
        }
    }
    return res;
}
int main()
{
    int n;
    cout << "Enter the dimension : ";
    cin >> n;
    int **arr = setAllZero(n);
    int **arr1 = setAllZero(n);
    cout << "Enter the elements for first matrix : " << endl;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cin >> arr[i][j];
        }
    }
    cout << "Enter the elements for second matrix : " << endl;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cin >> arr1[i][j];
        }
    }
    int **res;
    res = matrixMultiplication(arr, arr1, n);
    cout << "The multiplication of 2 matrices is : " << endl;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cout << res[i][j] << "\t";
        }
        cout << endl;
    }
}
```

Output:



```
PS C:\Users\beadi\Desktop\IDAA\DAA LAB\Assignment 4>
if ($?) { .\matMultiplicationRec }
Enter the dimension : 4
Enter the elements for first matrix :
1 2 3 4
1 2 3 4
5 2 3 1
6 3 4 1
Enter the elements for second matrix :
4 1 2 3
8 3 6 2
6 2 5 1
8 3 1 5
The multiplication of 2 matrices is :
70      25      33      30
70      25      33      30
62      20      38      27
80      26      51      33
```

Lab – 6

S No	Program Title	Date of Implementation	Remarks
i.	Heap Sort	Mar 23, 2022	

Heap Sort

Input: Array of integer

Output: Sorted array of integer

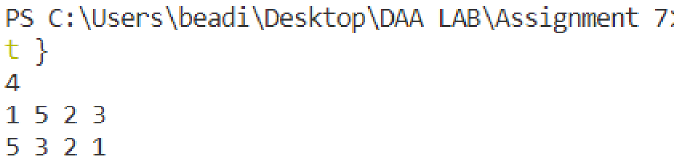
Time Complexity: O(n*log(n))

Code:

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

void heapSort(vector<int> &v){
    int n=v.size();

    // Create the heap
    for(int i=1; i<n; i++){
        int t=i;
        while((t/2)>0 && v[t/2]>v[t]){
            swap(v[t/2], v[t]);
            t/=2;
        }
    }

    // Remove root from the heap and store it at the end
    for(int i=n-1; i>1; i--){
        swap(v[i], v[1]);
        int t=1;
        while((2*t+1)<i){
            int l=2*t;
            int r=2*t+1;

            if(l<i && r<i){
                if(v[l]<v[r]){
                   if(v[2*t]<v[t]){
                        swap(v[2*t], v[t]);
                        t=2*t;
                        continue;
```

```
                    }
                }
                else{
                    if(v[2*t+1]<v[t]){
                        swap(v[2*t+1], v[t]);
                        t=2*t+1;
                        continue;
                    }
                }
            }
            else{
                break;
            }
        }
        if((2*t+1)==i){
            if(v[2*t]<v[t]){
                swap(v[2*t], v[t]);
                t=2*t;
            }
        }
    }
}

int main(){
    int n=0;
    cin>>n;
    vector<int> v(n+1, 0);   // because 1 based indexing
    for(int i=1; i<n+1; i++){
        cin>>v[i];
    }
    heapSort(v);

    for(int i=1; i<n+1; i++){
        cout<<v[i]<<" ";
    }
    return 0;
}
```

Output:



```
PS C:\Users\beadi\Desktop\DAA LAB\Assignment 7:
t }
4
1 5 2 3
5 3 2 1
```

Lab – 7

S No	Program Title	Date of Implementation	Remarks
i.	Binomial Heap	Mar 30, 2022	

Binomial Heap

Input: Integers

Output: Minimum element in heap and full heap traversal print

Time Complexity: O(log(n))

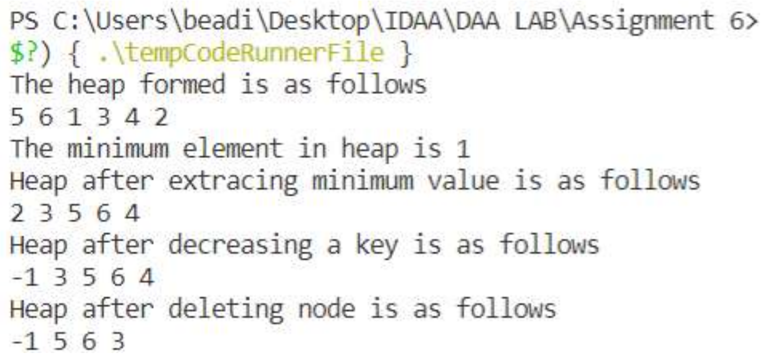
Code:

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

// A binomial heap node structure
struct node{
    int degree, data;
    struct node *parent, *sibling, *child;
};

// This function creates a new node with given key
struct node *newNode(int key){
    struct node *temp = new node;
    temp->data = key;
    temp->degree = 0;
    temp->child = temp->parent = temp->sibling = NULL;
    return temp;
    }

    // Merging two binomial trees
    struct node *mergeBinomialTrees(struct node *b1, struct node *b2){

    if (b1->data > b2->data)
        swap(b1, b2);

    b2->parent = b1;
    b2->sibling = b1->child;
    b1->child = b2;
    b1->degree++;
    return b1;
    }

    // This function performs union of two Binomial heaps
    list<node *> unionBinomialHeap(list<node *> l1, list<node *> l2){
    list<node *> res;

    auto i = l1.begin();
    auto j = l2.begin();

    while (i != l1.end() && j != l2.end())
    {
        if ((*i)->degree <= (*j)->degree)
        {
        res.push_back((*i));
        i++;
        }
        else
        {
        res.push_back((*j));
        j++;
        }
    }

    while (i != l1.end())
    {
        res.push_back((*i));
        i++;
    }

    while (j != l2.end())
    {
        res.push_back((*j));
        j++;
    }

    return res;
}

// Adjust function ensures that the root nodes in list are in increasing order
and no two binomial trees
// have the same degree.
list<node *> adjust(list<node *> heap){

    if (heap.size() <= 1)
        return heap;

    list<node *> new_heap;
    list<node *>::iterator it1, it2, it3;
    it1 = it2 = it3 = heap.begin();

    if (heap.size() == 2)
    {
        it2 = it1;
        it2++;
        it3 = heap.end();
    }
    else
    {
        it2++;
        it3 = it2;
        it3++;
    }

    while (it1 != heap.end())
    {
        if (it2 == heap.end())
        it1++;
        else if ((*it1)->degree < (*it2)->degree)
        {
        it1++;
        it2++;
        if (it3 != heap.end())
            it3++;
        }
        else if (it3 != heap.end() && (*it1)->degree == (*it2)->degree &&
(*it1)->degree == (*it3)->degree)
        {
        it1++;
        it2++;
        it3++;
        }
        else if ((*it1)->degree == (*it2)->degree)
        {
        struct node *temp;
        *it1 = mergeBinomialTrees(*it1, *it2);
        it2 = heap.erase(it2);
        if (it3 != heap.end())
            it3++;
        }
    }

    return heap;
}

// This function adds a Binomial tree in heap and then performs union on it
```

```
list<node *> insertTreeInHeap(list<node *> heap, struct node *tree){
    list<node *> temp;
    temp.push_back(tree);

    temp = unionBinomialHeap(temp, heap);

    return adjust(temp);
}

// This function inserts a new node in Binomial heap
list<node *> insert(int key, list<node *> heap){
    struct node *temp;
    temp = newNode(key);
    return insertTreeInHeap(heap, temp);
}

// This function returns the pointer to the minimum element of entire heap
struct node *getMin(list<node *> heap){
    struct node *minimum = NULL;
    int mini = INT_MAX;
    auto it = heap.begin();
    while (it != heap.end())
    {
        if ((*it)->data < mini)
        {
        mini = (*it)->data;
        minimum = (*it);
        }

        it++;
    }

    return minimum;
}

// This function is a helper function and it includes all the children of
minimum node in the main root list and then performs union and adjust to
ensure binomial trees of unique degree
list<node *> removeMinimum(struct node *tree){
    list<node *> heap;
    struct node *temp = tree->child;
    struct node *helper;

    while (temp)
    {
        helper = temp;
        temp = temp->sibling;
        helper->sibling = NULL;
        helper->parent = NULL;
        heap.push_front(helper);
    }
    return heap;
}

// This function extracts the minimum node from the Binomial heap and returns
the modified heap
list<node *> extractMin(list<node *> heap){
    list<node *> new_heap, helper;
    struct node *temp;

    temp = getMin(heap);
    auto it = heap.begin();

    while (it != heap.end())
    {
        if ((*it) != temp)
        {
        new_heap.push_back((*it));
        }
        it++;
    }

    helper = removeMinimum(temp);
    helper = unionBinomialHeap(new_heap, helper);
    helper = adjust(helper);
    return helper;
}

// This function searches a given Binomial tree for a node with a given value
and returns the pointer to that node
struct node *findNode(struct node *h, int val){
  if (h == NULL)
    return NULL;

  if (h->data == val)
    return h;

  struct node *res = findNode(h->child, val);
  if (res != NULL)
    return res;

  return findNode(h->sibling, val);
}

// This function takes input an old and a new key and replaces old key with
new key and performs necessary swapping to ensure min-heap property
```

```
list<node *> decreaseKey(list<node *> heap, int old_val, int new_val){
  struct node *temp = NULL;
  auto it = heap.begin();

  while (it != heap.end())
  {
    temp = findNode(*it, old_val);

    if (temp != NULL)
      break;
    // (*it) = (*it)->sibling;
    it++;
  }

  if (temp == NULL)
    return heap;

  temp->data = new_val;

  struct node *parent = temp->parent;

  while (parent != NULL && temp->data < parent->data)
  {
    swap(temp->data, parent->data);
    temp = parent;
    parent = parent->parent;
  }

  return heap;
}

// This function takes input a value and deletes the node with corresponding
value from the Binary heap
list<node *> deleteNode(list<node *> heap, int val){
  struct node *temp = NULL;
  auto it = heap.begin();

  while (it != heap.end())
  {
    temp = findNode(*it, val);

    if (temp != NULL)
      break;

    it++;
  }

  if (temp == NULL)
  {
    cout << "Value to be deleted not found in heap " << endl;
    return heap;
  }

  temp->data = INT_MIN;
  struct node *parent = temp->parent;

  while (parent != NULL && temp->data < parent->data)
  {
    swap(temp->data, parent->data);
    temp = parent;
    parent = parent->parent;
  }
  heap = extractMin(heap);

  return heap;
}

// This function take input a root of a Binomial tree and prints all the
values in that tree using DFS approach
void printTree(struct node *root){
  while (root)
  {
    cout << root->data << " ";
    printTree(root->child);
    root = root->sibling;
  }
}

// This function takes input of a Binomial heap and prints all its key values
void printHeap(list<node *> heap){
  auto it = heap.begin();

  while (it != heap.end())
  {
    printTree(*it);
    it++;
  }

  cout << endl;
}

// Main function
int main(){
    // 1. Creating a Binomial heap
    list<node *> heap;
    // 2. Inserting values in Binomial heap
    heap = insert(1, heap);
    heap = insert(2, heap);
    heap = insert(3, heap);
    heap = insert(4, heap);
    heap = insert(5, heap);
    heap = insert(6, heap);
    cout << "The heap formed is as follows\n";
    printHeap(heap);

    // 3. Getting minimum element from heap
    cout << "The minimum element in heap is " << getMin(heap)->data << endl;

    // 4. Removing minimum element from heap
    heap = extractMin(heap);
    cout << "Heap after extracing minimum value is as follows \n";
    printHeap(heap);

    // 5. Decreasing a key
    heap = decreaseKey(heap, 2, -1);
    cout << "Heap after decreasing a key is as follows\n";
    printHeap(heap);

    // 6. Deleting a node
    heap = deleteNode(heap, 4);
    cout << "Heap after deleting node is as follows\n";
    printHeap(heap);

    return 0;
}
```

Output:



```
PS C:\Users\beadi\Desktop\IDAA\DAA LAB\Assignment 6>
$?) { .\tempCodeRunnerFile }
The heap formed is as follows
5 6 1 3 4 2
The minimum element in heap is 1
Heap after extracing minimum value is as follows
2 3 5 6 4
Heap after decreasing a key is as follows
-1 3 5 6 4
Heap after deleting node is as follows
-1 5 6 3
```

Lab – 7

S No	Program Title	Date of Implementation	Remarks
i.	Fibonacci Heap	April 7, 2022	

Binomial Heap

Input: Integers

Output: Minimum element in heap and full heap traversal print

Time Complexity:

a) Make Heap
b) Find Min
c) Union
d) Insert a node
e) Extract min
f) Decrease a key
g) Delete a node

O(log(n))

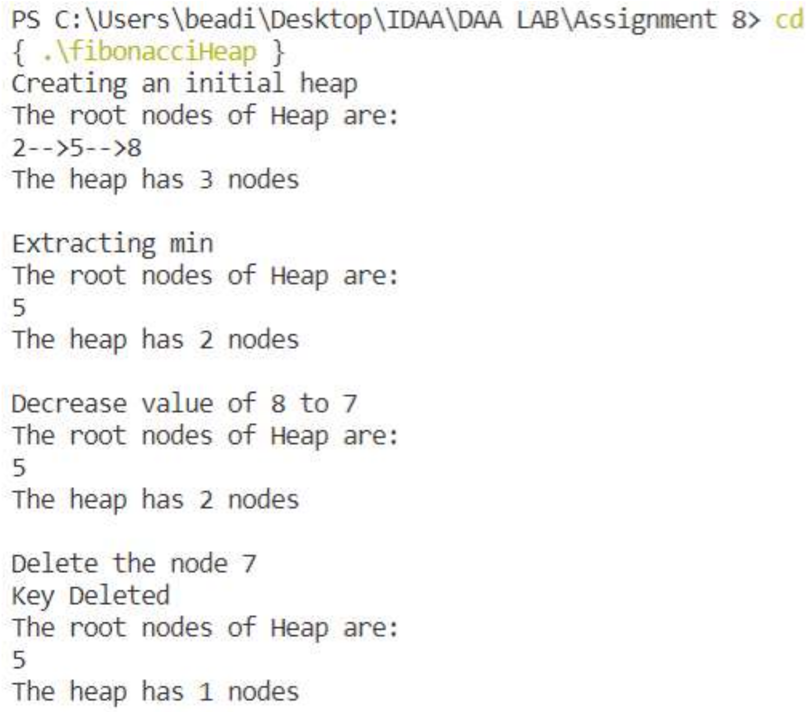
Code:

```
// C++ program to demonstrate various operations of fibonacci heap
#include<bits/stdc++.h>
using namespace std;

// Creating a structure to represent a node in the heap
struct node {
    node* parent; // Parent pointer
    node* child; // Child pointer
    node* left; // Pointer to the node on the left
    node* right; // Pointer to the node on the right
    int key; // Value of the node
    int degree; // Degree of the node
    char mark; // Black or white mark of the node
    char c; // Flag for assisting in the Find node function
};

// Creating min pointer as "mini"
```

```
struct node* mini = NULL;

// Declare an integer for number of nodes in the heap
int no_of_nodes = 0;

// Function to insert a node in heap
void insertion(int val)
{
    struct node* new_node = new node();
    new_node->key = val;
    new_node->degree = 0;
    new_node->mark = 'W';
    new_node->c = 'N';
    new_node->parent = NULL;
    new_node->child = NULL;
    new_node->left = new_node;
    new_node->right = new_node;
    if (mini != NULL) {
        (mini->left)->right = new_node;
        new_node->right = mini;
        new_node->left = mini->left;
        mini->left = new_node;
        if (new_node->key < mini->key)
            mini = new_node;
    }
    else {
        mini = new_node;
    }
    no_of_nodes++;
}
// Linking the heap nodes in parent child relationship
void Fibonnaci_link(struct node* ptr2, struct node* ptr1)
{
    (ptr2->left)->right = ptr2->right;
    (ptr2->right)->left = ptr2->left;
    if (ptr1->right == ptr1)
        mini = ptr1;
    ptr2->left = ptr2;
    ptr2->right = ptr2;
    ptr2->parent = ptr1;
    if (ptr1->child == NULL)
        ptr1->child = ptr2;
    ptr2->right = ptr1->child;
    ptr2->left = (ptr1->child)->left;
    ((ptr1->child)->left)->right = ptr2;
    (ptr1->child)->left = ptr2;
    if (ptr2->key < (ptr1->child)->key)
        ptr1->child = ptr2;
    ptr1->degree++;
}
// Consolidating the heap
void Consolidate()
{
    int temp1;
    float temp2 = (log(no_of_nodes)) / (log(2));
    int temp3 = temp2;
    struct node* arr[temp3+1];
    for (int i = 0; i <= temp3; i++)
        arr[i] = NULL;
    node* ptr1 = mini;
    node* ptr2;
    node* ptr3;
    node* ptr4 = ptr1;
    do {
        ptr4 = ptr4->right;
        temp1 = ptr1->degree;
        while (arr[temp1] != NULL) {
            ptr2 = arr[temp1];
            if (ptr1->key > ptr2->key) {
                ptr3 = ptr1;
                ptr1 = ptr2;
                ptr2 = ptr3;
            }
            if (ptr2 == mini)
                mini = ptr1;
            Fibonnaci_link(ptr2, ptr1);
            if (ptr1->right == ptr1)
                mini = ptr1;
            arr[temp1] = NULL;
            temp1++;
        }
        arr[temp1] = ptr1;
        ptr1 = ptr1->right;
    } while (ptr1 != mini);
    mini = NULL;
    for (int j = 0; j <= temp3; j++) {
        if (arr[j] != NULL) {
            arr[j]->left = arr[j];
            arr[j]->right = arr[j];
            if (mini != NULL) {
                (mini->left)->right = arr[j];
                arr[j]->right = mini;
                arr[j]->left = mini->left;
                mini->left = arr[j];
                if (arr[j]->key < mini->key)
                    mini = arr[j];
            }
            else {
                mini = arr[j];
            }
            if (mini == NULL)
                mini = arr[j];
            else if (arr[j]->key < mini->key)
                mini = arr[j];
        }
    }
}

// Function to extract minimum node in the heap
void Extract_min()
{
    if (mini == NULL)
        cout << "The heap is empty" << endl;
    else {
        node* temp = mini;
        node* pntr;
        pntr = temp;
        node* x = NULL;
        if (temp->child != NULL) {

            x = temp->child;
            do {
                pntr = x->right;
                (mini->left)->right = x;
                x->right = mini;
                x->left = mini->left;
                mini->left = x;
                if (x->key < mini->key)
                    mini = x;
                x->parent = NULL;
                x = pntr;
            } while (pntr != temp->child);
        }
        (temp->left)->right = temp->right;
        (temp->right)->left = temp->left;
        mini = temp->right;
        if (temp == temp->right && temp->child == NULL)
            mini = NULL;
        else {
            mini = temp->right;
            Consolidate();
        }
        no_of_nodes--;
    }
```

```
}

// Cutting a node in the heap to be placed in the root list
void Cut(struct node* found, struct node* temp)
{
    if (found == found->right)
        temp->child = NULL;

    (found->left)->right = found->right;
    (found->right)->left = found->left;
    if (found == temp->child)
        temp->child = found->right;

    temp->degree = temp->degree - 1;
    found->right = found;
    found->left = found;
    (mini->left)->right = found;
    found->right = mini;
    found->left = mini->left;
    mini->left = found;
    found->parent = NULL;
    found->mark = 'B';
}

// Recursive cascade cutting function
void Cascase_cut(struct node* temp)
{
    node* ptr5 = temp->parent;
    if (ptr5 != NULL) {
        if (temp->mark == 'W') {
            temp->mark = 'B';
        }
        else {
            Cut(temp, ptr5);
            Cascase_cut(ptr5);
        }
    }
}

// Function to decrease the value of a node in the heap
void Decrease_key(struct node* found, int val)
{
    if (mini == NULL)
        cout << "The Heap is Empty" << endl;

    if (found == NULL)
        cout << "Node not found in the Heap" << endl;
    found->key = val;

    struct node* temp = found->parent;
    if (temp != NULL && found->key < temp->key) {
        Cut(found, temp);
        Cascase_cut(temp);
    }
    if (found->key < mini->key)
        mini = found;
}

// Function to find the given node
void Find(struct node* mini, int old_val, int val)
{
    struct node* found = NULL;
    node* temp5 = mini;
    temp5->c = 'Y';
    node* found_ptr = NULL;
    if (temp5->key == old_val) {
        found_ptr = temp5;
        temp5->c = 'N';
        found = found_ptr;
        Decrease_key(found, val);
    }
    if (found_ptr == NULL) {
        if (temp5->child != NULL)
            Find(temp5->child, old_val, val);
        if ((temp5->right)->c != 'Y')
            Find(temp5->right, old_val, val);
    }
    temp5->c = 'N';
    found = found_ptr;
}

// Deleting a node from the heap
void Deletion(int val)
{
    if (mini == NULL)
        cout << "The heap is empty" << endl;
    else {

        // Decreasing the value of the node to 0
        Find(mini, val, 0);

        // Calling Extract_min function to
        // delete minimum value node, which is 0
        Extract_min();
        cout << "Key Deleted" << endl;
    }
}

// Function to display the heap
void display()
{
    node* ptr = mini;
    if (ptr == NULL)
        cout << "The Heap is Empty" << endl;

    else {
        cout << "The root nodes of Heap are: " << endl;
        do {
            cout << ptr->key;
            ptr = ptr->right;
            if (ptr != mini) {
                cout << "-->";
            }
        } while (ptr != mini && ptr->right != NULL);
        cout << endl
            << "The heap has " << no_of_nodes << " nodes" << endl
            << endl;
    }
}

// Driver code
int main()
{
    // We will create a heap and insert 3 nodes into it
    cout << "Creating an initial heap" << endl;
    insertion(5);
    insertion(2);
    insertion(8);

    // Now we will display the root list of the heap
    display();

    // Now we will extract the minimum value node from the heap
    cout << "Extracting min" << endl;
    Extract_min();
    display();

    // Now we will decrease the value of node '8' to '7'
    cout << "Decrease value of 8 to 7" << endl;
    Find(mini, 8, 7);
    display();

    // Now we will delete the node '7'
    cout << "Delete the node 7" << endl;
    Deletion(7);
    display();

    return 0;
}
```

Output:



```
PS C:\Users\beadi\Desktop\IDAA\DAA LAB\Assignment 8> cd
{ .\fibonacciHeap }
Creating an initial heap
The root nodes of Heap are:
2-->5-->8
The heap has 3 nodes

Extracting min
The root nodes of Heap are:
5
The heap has 2 nodes

Decrease value of 8 to 7
The root nodes of Heap are:
5
The heap has 2 nodes

Delete the node 7
Key Deleted
The root nodes of Heap are:
5
The heap has 1 nodes
```

------------------------------------- THANK YOU--------------------------------------

---