Practical 1: Finding maximum from an array without recursion

```
#include <iostream>
using namespace std;
int findMax(int arr[], int n) {
   int maxVal = arr[0];
   for (int i = 1; i < n; i++)
      if (arr[i] > maxVal)
         maxVal = arr[i];
   return maxVal;
}
int main() {
   int arr[] = {5, 12, 3, 19, 8};
   int n = sizeof(arr) / sizeof(arr[0]);

   cout << "Maximum element: " << findMax(arr, n) << endl;
   return 0;
}
```

Practical 2: Calculating Binomial Coefficient without recursion (Using formula: B(n, m) = B(n−1, m−1) + B(n−1, m), B(n,0)=B(n,n)=1)

```
#include<iostream>
using namespace std;
class Bino
{
        int k,S[30],add,top;
        public: int Binomial(int,int);
};
int Bino :: Binomial(int i,int j)
{
        top=-1;
```

```
        k=0;
        L1:if ((i!=j)&&(j!=0))
        {
                S[++top]=i-1;
                S[++top]=j-1;
                S[++top]=3;
    S[++top]=i-1;
                S[++top]=j;
                S[++top]=3;
        }
        else
           k++;
         if(top==-1)
        return(k);
  else
  {
        add=S[top--];
        j=S[top--];
        i=S[top--];
        if(add==3)
        {
                goto L1;
                }
        }
        return 0;
}
int main()
{
        Bino B;
        int a,b,val;
        cout<<"\n Enter two values:";
```

```
	cin>>a>>b;
	if (a>b)
	{
		val=B.Binomial(a,b);
		cout<<"\n Binomial coefficient of"<<a<<"&"<<b<<"is:"<<val;
	}
	else
	{
		cout<<"\n invalid input";
	}
}
```

Practical 3: Searching an element in an array without recursion

```
#include<iostream>
using namespace std;
class SearchEle
{
	int S[50],addr,top,A[50],n,i,no,j,k;
	public:
		SearchEle()
		{
			i=1;
		}
		void Get();
		void Search();
};
void SearchEle :: Get()
{
	cout<<"\n Enter the size of elements:";
	cin>>n;
```

```
	cout<<"\n Enter the elements:";
	for(int m=1;m<=n;m++)
	{
		cin>>A[m];
	}
	cout<<"\n Enter the element to be searched:";
	cin>>no;
}
void SearchEle :: Search()
{
	int j,k,top=0;
	L1:if(i<n)
	{
		S[++top]=i;
		S[++top]=2;
		i++;
		goto L1;

		L2:j=S[top--];
		if(A[j]==no)
		{
			k=j;
			cout<<"\n Element is found at position:"<<k;
			return;
		}
		else
		{
			k=0;
		}
	}
```

```
        if(top==0 && k==0)
        {
                cout<<"\n Element is not found:";
        }
        else
        {
                addr=S[top--];
                if(addr==2)
                        goto L2;
        }
}
int main()
{
        SearchEle S;
        int val;

        S.Get();
        S.Search();
}
```

Practical 4: Create Max/Min Heap using INSERT operation

```
#include<iostream>
using namespace std;
class InsertMaxHeap
{
        int a[20],n;
        public:
                void Insert(int);
                void Get();
                void Show();
```

```
};
void InsertMaxHeap::Get()
{
	cout<<"\n Enter the size of heap:";
	cin>>n;

	cout<<"Enter the element:\n";
	for(int i=1;i<=n;i++)
	{
		cin>>a[i];
	}
	cout<<"\n Before building heap:\n";
	Show();

	for(int i=2;i<=n;i++)
	{
		Insert(i);
	}
}
void InsertMaxHeap::Insert(int index)
{
	int i=index,item=a[i];
	while(i>1 && a[i/2]<item)
	{
		a[i]=a[i/2];
		i=i/2;
	}
	a[i]=item;
}
void InsertMaxHeap::Show()
{
```

```
	for(int i=1;i<=n;i++)
	{
		cout<<a[i]<<"\t";
	}
	cout<<endl;
}
int main()
{
	InsertMaxHeap heap;
	heap.Get();

	cout<<"\n After building max heap:\n";
	heap.Show();
	return 0;
}
```

Practical 5: Create Max/Min Heap using ADJUST/HEAPIFY operation

```
#include<iostream>
using namespace std;
class AdjustMinHeap
{
	int a[10],n;
	public:
		void Adjust(int,int);
		void Heapify(int);
		void Get();
		void Show();
};
void AdjustMinHeap :: Get()
{
```

```
	cout<<"\n Enter the size of heap:";
	cin>>n;

	cout<<"\n Enter the elements:";
	for(int i=1;i<=n;i++)
	{
		cin>>a[i];
	}
	Heapify(n);
}
void AdjustMinHeap :: Adjust(int i,int n)
{
	int j=2*i,item=a[i];
	while(j<=n)
	{
		if((j<n) && (a[j]>a[j+1]))
		{
			j++;
		}
		if(item<=a[j])
			break;
		a[j/2]=a[j];
		j=2*j;
	}
	a[j/2]=item;
}
void AdjustMinHeap :: Heapify(int n)
{
	for(int i=n/2;i>=1;i--)
		Adjust(i,n);
}
```

```
void AdjustMinHeap :: Show()
{
	cout<<"\n Min Heap is:";
	for(int i=1;i<=n;i++)
		cout<<a[i]<<"\t";
}
int main()
{
	AdjustMinHeap a;
	a.Get();
	a.Show();
}
```

```
#include<iostream>
using namespace std;
class AdjustMaxHeap
{
	int a[10],n;
	public:
		void Adjust(int,int);
		void Heapify(int);
		void Get();
		void Show();
};
void AdjustMaxHeap :: Get()
{
	cout<<"\n Enter the size of heap:";
```

```
        cin>>n;


        cout<<"\n Enter the elements:";
        for(int i=1;i<=n;i++)
        {
                cin>>a[i];
        }
        Heapify(n);
}
void AdjustMaxHeap :: Adjust(int i,int n)
{
        int j=2*i,item=a[i];
        while(j<=n)
        {
                if((j<n) && (a[j]>a[j+1]))
                {
                        j++;
                }
                if(item<=a[j])
                        break;
                a[j/2]=a[j];
                j=2*j;
        }
        a[j/2]=item;
}
void AdjustMaxHeap :: Heapify(int n)
{
        for(int i=n/2;i>=1;i--)
                Adjust(i,n);
}
void AdjustMaxHeap :: Show()
```

```
{
        cout<<"\n Max Heap is:";
        for(int i=1;i<=n;i++)
                cout<<a[i]<<"\t";
}
int main()
{
        AdjustMaxHeap a;
        a.Get();
        a.Show();
}
```

Practical 6: Sort a given array in Ascending/Descending order using Heap Sort with n = 1000, 2000, 3000 and measure exact execution time

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <chrono>
#include <cstdlib>
using namespace std;
using namespace chrono;

// Heapify for ascending order
void heapifyAsc(vector<int> &arr, int n, int i) {
   int largest = i; // root
   int l = 2*i + 1;
   int r = 2*i + 2;

   if (l < n && arr[l] > arr[largest]) largest = l;
   if (r < n && arr[r] > arr[largest]) largest = r;
```

```
    if (largest != i) {
        swap(arr[i], arr[largest]);
        heapifyAsc(arr, n, largest);
    }
}

// Heapify for descending order
void heapifyDesc(vector<int> &arr, int n, int i) {
    int smallest = i; // root
    int l = 2*i + 1;
    int r = 2*i + 2;

    if (l < n && arr[l] < arr[smallest]) smallest = l;
    if (r < n && arr[r] < arr[smallest]) smallest = r;

    if (smallest != i) {
        swap(arr[i], arr[smallest]);
        heapifyDesc(arr, n, smallest);
    }
}

// Heap Sort for ascending
void heapSortAsc(vector<int> &arr) {
    int n = arr.size();

    // Build max heap
    for (int i = n/2 - 1; i >= 0; i--)
        heapifyAsc(arr, n, i);

    // Extract elements
```

```
    for (int i = n - 1; i > 0; i--) {
        swap(arr[0], arr[i]);
        heapifyAsc(arr, i, 0);
    }
}

// Heap Sort for descending
void heapSortDesc(vector<int> &arr) {
    int n = arr.size();

    // Build min heap
    for (int i = n/2 - 1; i >= 0; i--)
        heapifyDesc(arr, n, i);

    // Extract elements
    for (int i = n - 1; i > 0; i--) {
        swap(arr[0], arr[i]);
        heapifyDesc(arr, i, 0);
    }
}

// Function to generate random array
vector<int> generateArray(int n) {
    vector<int> arr(n);
    for (int i = 0; i < n; i++)
        arr[i] = rand() % 10000; // Random values from 0 to 9999
    return arr;
}

// Function to measure execution time
void measureHeapSort(int n) {
```

```
    vector<int> arr = generateArray(n);
    vector<int> arrAsc = arr;
    vector<int> arrDesc = arr;

    cout << "\nArray Size: " << n;

    auto startAsc = high_resolution_clock::now();
    heapSortAsc(arrAsc);
    auto stopAsc = high_resolution_clock::now();
    auto durationAsc = duration_cast<microseconds>(stopAsc - startAsc);
    cout << "\n→ Ascending Sort Time: " << durationAsc.count() << " microseconds";

    auto startDesc = high_resolution_clock::now();
    heapSortDesc(arrDesc);
    auto stopDesc = high_resolution_clock::now();
    auto durationDesc = duration_cast<microseconds>(stopDesc - startDesc);
    cout << "\n→ Descending Sort Time: " << durationDesc.count() << " microseconds\n";
}

int main() {
    srand(time(0)); // Seed for random number generation
    measureHeapSort(1000);
    measureHeapSort(2000);
    measureHeapSort(3000);
    return 0;
}
```

Practical 7: Implement Weighted UNION and Collapsing FIND operations (Disjoint Set)

```
#include <iostream>
using namespace std;

int parent[10], size[10];

int find(int x) {
   return parent[x] = (parent[x] == x) ? x : find(parent[x]);
}

void unionSet(int x, int y) {
   int rx = find(x), ry = find(y);
   if (rx == ry) return;
   if (size[rx] < size[ry]) swap(rx, ry);
   parent[ry] = rx;
   size[rx] += size[ry];
}

int main() {
   for (int i = 0; i < 10; i++) parent[i] = i, size[i] = 1;

   unionSet(1, 2);
   unionSet(2, 3);
   unionSet(4, 5);
   unionSet(5, 6);
   unionSet(3, 6);

   cout << "Find(6): " << find(6) << "\n";
   cout << "Parent: ";
   for (int i = 0; i < 10; i++) cout << parent[i] << " ";
```

```
}
```

Practical 8: Search an element from a given array using Binary Search

```
#include<iostream>
#include<conio.h>
#include<time.h>
#include<stdlib.h>
using namespace std;

class BSearch
{
        int A[100],Size;
public:
   int Get();
        void sort();
        int Search(int,int,int);
        void show(int);
};
int BSearch :: Get()
{
        cout<<"\n Enter the Size of List:";
        cin>>Size;

        cout<<"\n The elements of List are:\n";
        //randomize();
        for(int i=1;i<=Size;i++)
        {
                A[i]=rand()%100;
                cout<<A[i]<<endl;
        }
        sort();
```

```
	cout<<"\n After sorting:\n";
	for(int i=1;i<=Size;i++)
	{
		cout<<A[i]<<endl;
	}
	return 0;
}
void BSearch :: sort()
{
	for(int i=1;i<=Size;i++)
	{
		for(int j=1;j<=Size;j++)
		{
			if(A[i]<A[j])
			{
				int temp=A[i];
				A[i]=A[j];
				A[j]=temp;
			}
		}
	}
}
int BSearch :: Search(int i,int j,int x)
{
	int mid;
	if(j==i)
	{
		if(x==A[i])
		  return i;
		else
		  return 0;
```

```
        }
        else
           mid=(i+j)/2;
        if(x==A[mid])
           return mid;
   else if(x<A[mid])
           return Search(i,mid-1,x);
   else
           return Search(mid+1,j,x);
}
void BSearch :: show(int x)
{
        int t=Search(1,Size,x);
        if(t==0)
            cout<<"\n Element is not found";
        else
            cout<<"\n Elements is found at location"<<t;
}
int main()
{
        int start,end;
        BSearch b;
        int No;
        //clrscr();

        start=clock();

        b.Get();
        cout<<"\n Enter element to search:";
        cin>>No;
        b.show(No);
```

```
	end=clock();


	cout<<"\n The execution time is:"<<(end-start)/CLK_TCK;
	//getch();
}
```

Practical 9: Write a program to find minimum and maximum from a given array using MAXMIN

```
#include<iostream>
using namespace std;
int main()
{
	int no;
	cout<<"Enter the size of array:";
	cin>>no;
	int a[no];
	for(int i=0;i<no;i++)
	{
		cin>>a[i];
	}
	int max=a[0],min=a[0];
	for(int i=1;i<no;i++)
	{
		if(a[i]>max)
			max=a[i];
		if(a[i]<min)
			min=a[i];
	}
	cout<<"Maximum Element"<<max;
	cout<<"Minimum Element"<<min;
	return 0;
}
```

Practical 10: Sort a given array in using Merge Sort

```
#include<iostream>
#include<conio.h>
#include<stdlib.h>
#include<math.h>
#include<time.h>
using namespace std;

class number
{
        int a[50],n;
public:
        void getdata();
        void mergesort(int low,int high);
        void merge(int low,int mid,int high);
};
void number :: getdata()
{
        int i;
        cout<<"\n Number of Element:";
        cin>>n;
        cout<<"\n Enter the Element:";
        for(i=1;i<=n;i++)
        {
          cin>>a[i];
        }
        cout<<"\n your Array is:";
        for(i=1;i<=n;i++)
           cout<<a[i]<<"\t";
        mergesort(1,n);
```

```
        cout<<"\n The array after sorting:";
        for(i=1;i<=n;i++)
           cout<<a[i]<<"\t";
}
void number :: mergesort(int low, int high)
{
        int mid;
        if (low<high)
        {
                mid=floor((low+high)/2);
                mergesort(low,mid);
                mergesort(mid+1,high);
                merge(low,mid,high);
        }
}
void number :: merge(int low,int mid,int high)
{
        int h,i,j,k,b[5000];
        h=low;
        i=low;
        j=mid+1;
        while((h<=mid)&&(j<=high))
        {
                if(a[h]<=a[j])
                {
                        b[i]=a[h];
                        h=h+1;
                }
                else
                {
                        b[i]=a[j];
```

```
                j=j+1;
            }
            i=i+1;
        }
        if(h>mid)
        {
            for(k=j;k<=high;k++)
            {
                b[i]=a[k];
                i=i+1;
            }
        }
        else
        {
            for(k=h;k<=mid;k++)
            {
                b[i]=a[k];
                i=i+1;
            }
        }
        for(k=low; k<=high; k++)
            a[k]=b[k];
}
int main()
{
        number a;
        int start,end;
        start=clock();
        a.getdata();
        end=clock();
```

```
        cout<<"\n The execution time is:"<<(end - start)/CLK_TCK;
}
```