1. Linear Search

```
void linearSearch(int a[],int key,int n)
{
	int flag=0;
	for(int i=0;i<n;i++)
	{
		if(a[i]==key)
		flag=1;
		cout<<"Key element "<<key<<" is found at "<<i+1<<" index position";
	}
	if(flag!=1)
		cout<<"Element is not found in the array";
}
```

2. Binary search

```
int binarySearch(int a[],int key,int n)
{
	int low=0;
	int high=n-1;
	int mid;

	while(low<=high)
	{
		mid=(low+high)/2;
		if(a[mid]==key)
			return mid;

		else if(a[mid]<high)
			low=mid+1;
		else
			high=mid-1;
	}

	return 1;
}
```

3. Fractional Knapsack:

```
class item
{
	public:
	int weight,profit;
};
```

```
bool pwRatioComp(item a, item b)
{
	double r1=(double)a.profit/(double)a.weight;
	double r2=(double)b.profit/(double)b.weight;
	return r1>r2;
}

void fractionalKnapsack(item a[],int w,int n)
{
	int currentW=0;
	double maxProfit=0.0;

	sort(a,a+n,pwRatioComp);

	for(int i=0;i<n;i++)
	{
		//full object
		if(currentW+a[i].weight<=w)
		{
			currentW=currentW+a[i].weight;
			maxProfit=maxProfit+a[i].profit;
			cout<<"Item having weight "<<a[i].weight<<" is in the knapsack";
		}

		//fractional item
		else
		{
			int remain=w-currentW;
			maxProfit=maxProfit+((double)remain/(double)a[i].weight)*a[i].profit;
			cout<<"Item having "<<((double)remain/(double)a[i].weight)<<" fractional
of item of weight "<<a[i].weight<<" in the knapsack";
			break;
		}
	}
	cout<<"The maximum profit is:"<<maxProfit;
}
```

4. Job scheduling

```
class job
{
	public:
	char jobID;
	int ded;
	int profit;
```

```
};

bool profitComp(job a, job b)
{
	return(a.profit>b.profit);
}

void jobScheduling(job a[],int n)
{
	int result[n];
	bool slot[n];
	int profit=0;
	for(int i=0;i<n;i++)
		slot[i]=false;

	sort(a,a+n,profitComp);

	for(int i=0;i<n;i++)
	{
		for(int j=min(n,a[i].ded)-1;j>=0;j--)
		{
			if(slot[j]==false)
			{
				result[j]=i;
				slot[j]=true;
				profit=profit+a[i].profit;
				break;
			}
		}
	}

	for(int i=0;i<n;i++)
	{
		if(slot[i])
			cout<<a[result[i]].id<<" ";
	}
	cout<<"Max profit is:"<<profit;
}
```

5. Dynamic Coin Change

```
int coin;
int w;

int minComp(int a,int b)
```

```
{
	if(a>b)
		return b;
	else
		return a;
}

void coinChange(int coin,int w)
{
	int a[coin+1][w+1];
	for(int i=0;i<=coin;i++)
	{
		for(int j=0;j<=w;j++)
		{
			if(i==0||j==0)
				a[i][j]=0;
			else if(i==1)
				a[i][j]=j;
			else if(i>j)
				a[i][j]=a[i-1][j];
			else
				a[i][j]=minComp(a[i-1][j],1+a[i][j-i]);
		cout<<a[i];
		}
	cout<<endl;
	}

}
```

5. TSP

```
int tsp(int graph[][10],int s,int n)
{
	vector<int>vertex;
	for(int i=0;i<n;i++)
	{
		if(i!=s)
			vertex.push_back(i);
	}

	int minpath=INT_MAX;

	do
	{
		int currentW=0;
```

```
                int k=s;
                for(int i=0;i<vertex.size();i++)
                {
                        currentW=currentW+graph[k][vertex[i]];
                        k=vertex[i];
                }
                currentW=currentW+graph[k][s];
        minpath=min(currentW,minpath);

        }while(next_permutation(vertex.begin(),vertex.close());

        return minpath;
}
```

6. Coin change greedy

```
int minCoins(int c[],int value,int n)
{
        int  max=999;
        int res=max;

        if(value==0)
                return 0;

        else
        {
                for(int i=0;i<n-1;i++)
                {
                        if(value>0 && c[i]<value)
                        {
                                int temp=minCoins(c,value-c[i],n)
                                if(res>temp)
                                        res=temp+1;
                        }
                }
        }
        return res;

}
```

7. N queen

```
int board[10][10];
int possSol=0;

void print(n)
```

```
{
	for(int i=0;i<=n-1;i++)
	{
		for(int j=0;j<=n-1;j++)
		{
			cout<<board[i][j]<<" ";
			cout<<endl;
		}
		cout<<endl;

	}
	cout<<endl;
}

//row-> queen no col->possible safe position n-> no of queens
bool isSafe(int col,int row,int n)
{
	for(int i=0;i<row;i++)
	{
		if(board[i][col])
			return false;
	}

	//left diahonal check
	for(int i=row,j=col;i>=0&&j>=0;i--,j--)
	{
		if(board[i][j])
			return false;
	}

	//right diagonal
	for(int i=row,j=col;i>=0&&j<n;j++,i--)
	{
		if(borad[i][j])
			return false;
	}

	return true;
}

//to find positions where which queen can be placed
bool solve(int row,int n)
{
	if(row==n)
```

```
	{
		possSol++;
		cout<<"Solution no:"<<possSol;
		print(n);
		return true;
	}

	//res is for backtracking, occur if a position queen is placed but the child node can't be
where queen is placed
	bool res=false;
	for(int i=0;i<=n-1;i++)
	{
		if(isSafe(i,row,n))
		{
			board[row][i]=1;
			res=solve(row+1,n)||res
			//if res is false backtrack and make the new board [][] values as false
			board[row][i]=0;
		}
	}
	return false;
}

int main()
{
	res=solve(0,n)
	if(res==-1)
		cout<<-1;
	else
		endl;
cout<< toal sol<<possSol;
}
```

8. Cab Activity sol when time is given in HH:MM

```
int a[1000];
int MAX=1440;
void minCabs(int n)
{
	int h1,h2,m1,m2,t1,t2,ans;
	for(int i=0;i<n;i++)
	{
		cin>>h1>>m1>>h2>>m2;
		t1=h1*60+m1;
		t2=h2*60+m2;
```

```
                for(int i=t1;i<=t2;i++)
                {
                        a[i]++;
                }
        }

        int r=0;
        for(int i=0;i<MAX;i++)
        {
                r=max(r,a[i]);
        }

        cout<<min cabs:<<r;

}
```

9. Cab activity sol when S=[] and F=[] are given

```
int count=0, minFT=F[0],k=0;

for(int i=0;i<n;i++)
{
        if(minFT>S[i])
                count++;
        if(minFT<=S[i])
        {
                k++;
                minFT=F[k];
        }
}
cout<<min cabs<<count;
```

10. Activity selection

```
class activity
{
        public:
        int start,finish;
};

bool sortFt(activity a, activity b)
{
        return (a.finish<b.finish);
}
```

```
void scheduleActivity(activity a[],int n)
{
	sort(a,a+n,sortFt);
	cout<<Included activitites are:<<
	int i=0;
	cout<<(<<a[i].start<<,<<,a[i].finish<<);
	for(int j=0;j<n;j++)
	{
		if(a[j].start>=a[i].finish)
		{
			cout<<(<<a[j].start<<,<<,a[j].finish<<);
			i=j;
		}
	}
}
```

11. min max problem

```
class pair
{
	public:
	int min,max;
};

pair minMax(int a[],int low,int high)  /// minmax(a,0,n-1)
{
	pair mm,mml,mmr;
	int mid;

	if(low==high) //only one element
	{
		mm.max=a[low];
		mm.min=a[low];
		return mm;
	}

	if (high-low==1) //2 elements
	{
		if(a[low]>a[high])
		{
			mm.min=a[high];
			mm.max=a[low];
```

```
		}
		else
		{
			mm.max=a[high];
			mm.min=a[low];
		}
	return mm;
	}

	else //more than 2
	{
		mid=(low+high)/2;
		mml=minMax(a,low,mid-1);
		mmr=minMax(a,mid+1,high);

		if(mml.max>mmr.max)
			mm.max=mml.max;
		else
			mm.max=mmr.max;

		if(mml.min>mmr.min)
			mm.min=mmr.min;
		else
			mm.min=mml.min;

		return mm;
	}
}
```

12. Naive Bayes Algo

```
void search(string text,string pattern)
{
	int n=text.size();
	int m=pattern.size();

	for(int i=0;i<n-m;i++)
	{
		for(int j=0;j<m;j++)
		{
			if(text[i+j]!=pattern[j])
				break;

			if(j==m)
				cout<<"Pattern matched at <<i<< "index position";
```

```
                }
        }
}
```

13. KMP

```
void lps_cal(string pat,int lps[])
{
        lps[0]=0;
        int i=1;
        int len=0;
        while(i<pat.length())
        {
                if(pat[i]==len[i])
                {
                        len++;
                        lps[i]=len;
                        i++;
                }

                else
                {
                        if(len==0)
                        {
                                lps[i]=0;
                                i++;
                        }
                        else
                        {
                                len=lps[len-1];
                        }
                }
        }
}

void KMP(string pattern,string text)
{
        int n=text.size();
        int m=pattern.size();

        int lps[m];
        lps_cal(pattern,lps);
        int i=0,j=0;
        while(i<n)
        {
```

```
            if(pattern[j]==text[i])
            {
                  i++;
                  j++;
            }
            if (j==m)
            {
                  cout<<"pattern matched at index"<<i-j;
                  j=lps[j-1];
            }

            else if(i<n&& pattern[j]!=text[i])
            {
                  if(j!=0)
                        j=lps[j-1];
                  else
                        i=i+1;
            }
      }
}
```

14. Huffman Code

```
//right->1 left ->0
class node
{
      public:
            int freq;
            char data;
            const node *l,*r;

      node(char d, int f=-1)
      {
            data=d;
            freq=f;
            l=NULL;
            r=NULL;
      }

      node (const node *p, const node *q)
      {
            data=0;
            freq=p->freq+q->freq;
            l=p;
            r=q;
```

```
	}

	bool operator<(const node &a)const{
		return freq>a.freq;
	}

	void traverse(string code="")const
	{
		if(l!=NULL)
		{
			l->traverse(code+'0');
			r->traverse(code+'1');
		}

		else
		{
			cout<< "Data:"<<data<<" "<<"Frequency:"<<freq<<"
"<<"Code:"<<code<<endl;
		}

	}

};

void huffmanCode(string s)
{
	priority_queue <node>pq;
	int frequency[1000];
	for(int i=0;i<256;i++)
		frequency[i]=0;

	for(int i=0;i<s.size();i++)
		frequency[int(s[i])]++;

	for(int i=0;i<256;i++)
	{
		if(frequency[i])
		{
			pq.push(node(i,frequency[i]));
		}
	}

	while(pq.size()>1)
	{
```

```
                node *t=new node(pq.top());
                pq.pop();
                node *q=new node(pq.top());
                pq.pop();
                pq.push(node(t,q));
        }
        cout<<"huffman encoding";
        pq.top().traverse();
}
```

15. Quick Sort