**//Permutation Recursion**

void permutation(int idx,int length){

if(idx == length) {

for(int i=1;i<=length;i++){

cout<<arr[i]<<" ";

}

cout<<endl;

return;

}

for(int j=idx;j<=length;j++){

int temp=arr[idx];

arr[idx]=arr[j];

arr[j]=temp;

permutation(idx+1,length);

temp=arr[idx];

arr[idx]=arr[j];

arr[j]=temp;

}

return;

}

**//Basic Queue Implementation**

struct Object{

int key;

};

struct Queue{

Object object;

Object array[MAX\_SIZE+5];

int start\_ptr=0,end\_ptr=0,total\_size=0;

bool empty(){

if(total\_size == 0) return true;

return false;

}

Object front(){

if(total\_size != 0) {

return array[start\_ptr];

}

}

bool pop(){

if(total\_size > 0) {

start\_ptr=(start\_ptr+1)%MAX\_SIZE;

total\_size--;

return true;

}

return false;

}

bool push(Object object){

if(total\_size<MAX\_SIZE) {

array[end\_ptr]=object;

end\_ptr = (end\_ptr+1)%MAX\_SIZE;

total\_size=total\_size+1;

return true;

}

return false;

}

};

**// Backtrack**

int arr[MAX+1];

int list[MAX+1];

void backtrack(int idx,int pos,int total,int n){

if(pos>n) {

for(int i=1;i<=n;i++){

if(i==1) cout<<list[i];

else if(i==n) cout<<" "<<list[i]<<endl;

else{

cout<<" "<<list[i];

}

}

return;

}

for(int i=idx;i<=total;i++){

list[pos]=arr[i];

backtrack(i+1,pos+1,total,n);

}

return;

}

**// Basic Stack Implementation**

struct Object{

int key;

};

struct Stack{

Object object;

Object array[MAX\_SIZE+5];

int start\_ptr=0,end\_ptr=0,total\_size=0;

bool empty(){

if(total\_size == 0) return true;

return false;

}

Object top(){

if(total\_size != 0) {

return array[start\_ptr-1];

}

}

bool pop(){

if(total\_size > 0) {

start\_ptr=(start\_ptr-1+MAX\_SIZE)%MAX\_SIZE;

total\_size--;

return true;

}

return false;

}

bool push(Object object){

if(total\_size<MAX\_SIZE) {

array[start\_ptr]=object;

start\_ptr = (start\_ptr+1)%MAX\_SIZE;

total\_size=total\_size+1;

return true;

}

return false;

}

};

**//merge sort implementation**

void merge\_sort(int arr[],int n){

if(n==1) return;

int left\_arr[MAX+1];

int right\_arr[MAX+1];

int left\_n=0,right\_n=0;

for(int i=1;i<=n/2;i++){

left\_n++;

left\_arr[left\_n]=arr[i];

}

for(int i=n/2+1;i<=n;i++){

right\_n++;

right\_arr[right\_n]=arr[i];

}

merge\_sort(left\_arr,left\_n);

merge\_sort(right\_arr,right\_n);

int i=0,j=1,k=1;

while(j<=left\_n && k<=right\_n){

if(left\_arr[j] <= right\_arr[k]) {

i++;

arr[i]=left\_arr[j];

j++;

}

else{

i++;

arr[i]=right\_arr[k];

k++;

}

}

while(j<=left\_n){

i++;

arr[i]=left\_arr[j];

j++;

}

while(k<=right\_n){

i++;

arr[i]=right\_arr[k];

k++;

}

return;

}

**//binary search**

int bs(struct Friend arr[], int st, int en, int d){

int mid;

int save=st;

int beg=st;

while(st<=en){

mid=(st+en)/2;

if((arr[mid].money-arr[beg].money)<d) {

save = max(save,mid);

st=mid+1;

}

else{

en=mid-1;

}

}

return save;

}

**//Meet in the middle Technique**

int collection[MAX\_POSS+5][MAX+5];

int sum\_collection[MAX\_POSS+5];

int cnt;

int counting\_sort[

void generate\_all\_possible(int main\_arr[],int temp\_arr[],int idx\_main,int idx\_temp,int tot){

if(idx\_main>tot) {

cnt++;

sum\_collection[cnt] = idx\_temp;

for(int i=1;i<=idx\_temp;i++){

collection[cnt][i]=temp\_arr[i];

}

return;

}

//choice 1

//not taking

generate\_all\_possible(main\_arr,temp\_arr,idx\_main+1,idx\_temp,tot);

temp\_arr[idx\_temp+1]=main\_arr[idx\_main];

generate\_all\_possible(main\_arr,temp\_arr,idx\_main+1,idx\_temp+1,tot);

temp\_arr[idx\_temp+1] = -1;

return;

}

void meet\_in\_the\_middle(int arr[],int n){

//generate all possible

int left[MAX+1];

for(int i=1;i<=n/2;i++){

left[i]=arr[i];

}

int temp\_arr[MAX+1];

cnt=0;

for(int i=1;i<=MAX\_POSS;i++){

for(int j=1;j<=MAX;j++){

collection[i][j]=-1;

}

sum\_collection[i]=0;

}

generate\_all\_possible(left,temp\_arr,1,0,n/2);

/\*for(int i=1;i<=cnt;i++){

for(int j=1;j<=sum\_collection[i];j++){

cout<<collection[i][j]<<" ";

}

cout<<endl;

}\*/

int right\_arr[MAX+1];

int rt=0;

for(int i=n/2+1;i<=n;i++){

rt++;

right\_arr[rt]=arr[i];

}

int counting\_sort[MAX+500];

for(int i=0;i<MAX\_NUMBR;i++) {

counting\_sort[i]=0;

}

for(int i=1;i<=rt;i++){

counting\_sort[right\_arr[i]]++;

}

//Now merge two systems

for(int i=1;i<=cnt;i++){

}

}