MST

```
struct edge
{
   int u,v,w;
   bool operator < ( const edge& p ) const
   {
      return w < p.w;
   }
};
int pr[MAXN];
vector<edge>e;
int find(int r)
{
  return (pr[r]==r) ? r:  pr[r]=find(pr[r]);
}
int mst(int n)
{
   sort(e.begin(),e.end());
   for(int i=1;i<=n;i++)pr[i]=i;

   int count=0,s=0;
   for(int i=0;i<(int)e.size();i++)
   {
      int u=find(e[i].u);
      int v=find(e[i].v);
      if(u!=v)
      {
         pr[u]=v;
         count++;
         s+=e[i].w;
         if(count=K=n-1) break;
      }
   }
   return s;
}
```

scc

```
#define graph_size 100001

vector<int> G[graph_size],RG[graph_size];
vector<int> components[graph_size];
stack<int> st;
bool vis[graph_size];
int mark;

void dfs(int u)
{
   vis[u]=1;
   for(int i=0;i<G[u].size();i++){
      int v=G[u][i];
      if(!vis[v]) dfs(v);
   }
   st.push(u);
   return ;
}
void dfs2(int u,int mark){
   components[mark].push_back(u);
   vis[u]=1;
   for(int i=0;i<RG[u].size();i++){
      int v=RG[u][i];
      if(!vis[v]) dfs2(v,mark);
   }
   return ;
}
void SCC(int n){
   while(!st.empty()) st.pop();
   for(int i=0;i<=n;i++) components[i].clear();
   mark=0;
   memset(vis,0,sizeof vis);

   for(int i=1;i<=n;i++){
      if(!vis[i]) dfs(i);
   }

   memset(vis,0,sizeof vis);
   while(!st.empty()){
      int u=st.top();
      st.pop();
      if(!vis[u]){
         dfs2(u,mark);
         mark++;
      }
   }
}
```

KMP

```
int f[1000007];

void failure_function(char *pattern)
{
   f[0] = 0;
   int k = 1, len = 0, len_p = strlen(pattern);

   while (k < len_p)
   {
```

```
      if (pattern[k] == pattern[len])f[k++] =
++len;
      else
      {
         if (len)len = f[len - 1];
         else f[k++] = 0;
      }

   }
   return;
}

void KMP_match(char *txt, char*pattern)
{
   int i = 0, j = 0, ret = -1;
   int len_t = strlen(txt), len_p =
strlen(pattern);

   while (i < len_t)
   {
      if (txt[i] == pattern[j])
      {
         i++; j++;
         if (j == len_p)
         {
            ret = i - len_p;
            printf("A match found from index
%d\n", ret);
            j = f[j - 1];
         }
      }
      else
      {
         if (j)j = f[j - 1];
         else i++;
      }
   }
}

int main()
{
   int tc, t = 0;
   scanf("%d\n", &tc);
   while (tc--)
   {
      char TXT[ 1000007], PT[1000007];
      gets(TXT);
      gets(PT);
      failure_function(PT);

      KMP_match(TXT, PT);
   }
}
```

MATRIX:

```
template<int N> class matrix {
public:
   int  arr[N][N];
   matrix() {
      for( int i = 0 ; i < N ; i ++ ) {
         for( int j = 0 ; j < N ; j++ ) {
            arr[i][j] = 0 ;
         }
      }
   }
   matrix<N> operator *(const matrix<N> &in) {
      matrix<N> ret ;
      for( int i = 0 ; i < N ; i++ ) {
         for( int j = 0 ; j < N ; j++ )
            for( int k = 0 ; k < N ; k++ ) {
               ret.arr[i][j]+=(arr[i][k])*(in.arr[k][j])
;
               ret.arr[i][j]%=10000 ;
            }
      }
      return ret ;
   }
   matrix<N> operator ^( int POW ) {
      matrix<N> ret ;
      for( int i = 0 ; i < N ; i++ ) {
         ret.arr[i][i] = 1 ;
      }
      matrix<N> ME = *this ;
      while( POW ) {
         if( POW&1 ) {
            ret = ret * ME ;
         }
         ME = ME * ME ;
         POW >>= 1 ;
      }
      return ret ;
   }
};
```

BIGMOD

```
int bigmod(long long B,long long P,long long
MOD)
{
   long long R=1;
   while(P>0){
      if(P%2==1){
         R=(R*B)%MOD;
      }
      P/=2;
      B=(B*B)%MOD;
   }
   return R;
}

GENERATE DIVISORS
#define SIZE_N 100
#define SIZE_P 100

bool flag[SIZE_N+5];
int primes[SIZE_P+5];

int seive()
{
   int i,j,total=0,val;
   for(i=2; i<=SIZE_N; i++) flag[i]=1;
   val=sqrt(SIZE_N)+1;
   for(i=2; i<val; i++){
      if(flag[i]){
         for(j=i; j*i<=SIZE_N; j++) flag[i*j]=0;
      }
   }

   for(i=2; i<=SIZE_N; i++){
      if(flag[i]) primes[total++]=i;
   }

   return total;
}

int store_primes[100],freq_primes[100],
store_divisor[10000], Total_Prime, ans;

void divisor(int N)
{
   int i,val,ct;

   val=sqrt(N)+1;
   Total_Prime=0;
   for(i=0; primes[i]<val; i++){
      if(N%primes[i]==0){
         ct=0;
         while(N%primes[i]==0){
            N/=primes[i];
            ct++;
         }
         store_primes[Total_Prime]=primes[i];
         freq_primes[Total_Prime]=ct;
         Total_Prime++;
         val=sqrt(N)+1;
      }
   }
   if(N>1){
      store_primes[Total_Prime]=N;
      freq_primes[Total_Prime]=1;
      Total_Prime++;
   }
}
void Generate(int cur,int num)
{
   int i,val;

   if(cur==Total_Prime){
      store_divisor[ans++]=num;
   }
   else
   {
      val=1;
      for(i=0; i<=freq_primes[cur]; i++){
         Generate(cur+1,num*val);
         val=val*store_primes[cur];
      }
   }
}
int main()
{
   int total=seive();
   int n,i;

   while(scanf("%d",&n)==1){
      divisor(n);
      ans=0;
      Generate(0,1);

sort(&store_divisor[0],&store_divisor[ans]);
      printf("Total No of Divisors: %d\n",ans);
      for(i=0; i<ans; i++){
```

```
        printf("%d ",store_divisor[i]);
      }
      printf("\n");
   }
   return 0;
}
```

NCR

```
#define size_nCr 1001
ll int combination[size_nCr][size_nCr];
void precal_nCr()
{
   combination[0][0]=1;
   for(int i=0;i<size_nCr;i++){
      for(int j=0;j<=i;j++){
         if(j==i || j==0) combination[i][j]=1;
         else if(j==1)combination[i][j]=i;
         else combination[i][j]=combination[i-
1][j]+combination[i-1][j-1];
      }
   }
}
ll int nCr(int n, int r)
{
   if(n>=size_nCr || r>=size_nCr) return 0;
   if(n<0 || r<0) return 0;
   else return Kcombination[n][r];
}
```

PHI FUNCTION

```
int phi[1000105];
void PHI(){
   phi[1]=0;
   for(int i=2;i<=1000100;i++)phi[i]=i;
   for(int i=2;i<=1000100;i++){
      if(phi[i]==i){
         phi[i]--;
         for(int j=i+i;j<=1000100;j+=i){
            phi[j]=(phi[j]/i)*(i-1);
         }
      }
   }
}
```

SUM OF DIVISROS

SEIVE FIRST

```
int SOD(int N)
{
   int i,val,sum,p,s;
   val=sqrt(N)+1;
   sum=1;
   for(i=0; primes[i]<val; i++){
      if(N%primes[i]==0){
         p=1;
         while(N%primes[i]==0){
            N/=primes[i];
            p=p*primes[i];
         }
         p=p*primes[i];
         s=(p-1)/(primes[i]-1);
         sum=sum*s;
      }
   }

   if(N>1){
      p=N*N;
      s=(p-1)/(N-1);
      sum=sum*s;
   }
   return sum;
}
```