1. Prime Factorization

prime(int n)

{

    while (n%2 == 0)

    {

        printf("%d ", 2);

        n = n/2;

    }

    for (int i = 3; i <= sqrt(n); i = i+2)

    {

        while (n%i == 0)

        {

            printf("%d ", i);

            n = n/i;

        }

    }

    if (n > 2)

        printf ("%d ", n);

}

2. sieve\_algorithm:

void SieveOfEratosthenes(int n)

{

    bool prime[n+1];

    memset(prime, true, sizeof(prime));

    for (int p=2; p\*p<=n; p++)

    {

        if (prime[p] == true)

        {

            for (int i=p\*2; i<=n; i += p)

                prime[i] = false;

        }

    }

    // Print all prime numbers

    for (int p=2; p<=n; p++)

       if (prime[p])

          cout << p << " ";

}

void search(char\* pat, char\* txt)

{

    int M = strlen(pat);

    int N = strlen(txt);

    for (int i = 0; i <= N - M; i++) {

        int j;

                for (j = 0; j < M; j++)

            if (txt[i + j] != pat[j])

                break;

        if (j == M)            printf("Pattern found at index %d \n", i);

    }

}

3.DFS:

void find\_max(int s)

{

**int v=0,l=0;int level[100005];**

    queue<int>q;

    q.push(s);

    vis[s]=1;

**level[s]=0;**

    while(!q.empty())

    {

       int d=q.front();

        q.pop();

**if(level[d]%2)**

            v++;

            else

                l++;

        for(int i=0;i<store[d].size();++i)

**{**

            if(!vis[store[d][i]])

            {

                level[store[d][i]]=level[d]+1;

                vis[store[d][i]]=1;

**q.push(store[d][i]);**

          }

        }

  }

    result+=max(l,v);

**}**

4. BFS Algo

class Graph

{

int numVertices;

list \*adjLists;

bool\* visited;

public:

Graph(int vertices);

void addEdge(int src, int dest);

void BFS(int startVertex);

};

Graph::Graph(int vertices)

{

numVertices = vertices;

adjLists = new list[vertices];

}

void Graph::addEdge(int src, int dest)

{

adjLists[src].push\_back(dest);

adjLists[dest].push\_back(src);

}

void Graph::BFS(int startVertex)

{

visited = new bool[numVertices];

for(int i = 0; i < numVertices; i++)

visited[i] = false;

list queue;

visited[startVertex] = true;

queue.push\_back(startVertex);

list::iterator i;

while(!queue.empty())

{

int currVertex = queue.front();

cout << "Visited " << currVertex << " ";

queue.pop\_front();

for(i = adjLists[currVertex].begin(); i != adjLists[currVertex].end(); ++i)

{

int adjVertex = \*i;

if(!visited[adjVertex])

{

visited[adjVertex] = true;

queue.push\_back(adjVertex);

}

}

}

}

5.Heaps Algo.

void printArr(int a[],int n)

{

    for (int i=0; i<n; i++)

        cout << a[i] << " ";

    printf("\n");

}

void heapPermutation(int a[], int size, int n)

{

    if (size == 1)

    {

        printArr(a, n);

        return;

    }

    for (int i=0; i<size; i++)

    {

        heapPermutation(a,size-1,n);

        if (size%2==1)

            swap(a[0], a[size-1]);

        else

            swap(a[i], a[size-1]);

    }

}