**Binary search->**

**Code->**

#include<iostream>

using namespace std;

int binary(int a[],int m,int h,int l,int f)

{

if(l>h)

return -1;

else if(a[m]==f)

return m;

else if(a[m]>f)

{

int a1,b,c;

a1=l;

b=m-1;

c=(l+m-1)/2;

return binary(a,c,b,a1,f);

}

else{

int a1,b,c;

a1=h;

b=m+1;

c=(h+m+1)/2;

return binary(a,c,a1,b,f);

}

return -1;

}

int main()

{

int a[]={1,2,3,4,5};

cout<<binary(a,2,4,0,5)<<endl;

return 0;

}

**Linear search using recursion->**

**first occurence index->**

**code->**

#include<iostream>

using namespace std;

int index(int a[],int n,int c,int ele)

{

if(c==n-1 && a[c]!=ele)

return -1;

else if(c==n-1 && a[c]==ele)

return c;

if(a[c]==ele)

return c;

else

return index(a,n,c+1,ele);

return -1;

}

int main()

{

int a[]={1,2,2,3,4,5,6};

cout<<index(a,7,0,11);

return 0;

}

**Pow(a,n)->**

**Code->**

#include<iostream>

using namespace std;

int pow(int a,int n)

{

if(n==1)

return a;

else if(n==0)

return 1;

return a\*pow(a,n-1);

}

int main()

{

cout<<pow(3,2);

return 0;

}

**Mul(a,n) without \*->**

**Code->**

#include<iostream>

using namespace std;

int mul(int a,int n)

{

if(n==1)

return a;

else if(n==0)

return 0;

return a+mul(a,n-1);

}

int main()

{

cout<<mul(2,3);

return 0;

}

**Backward linear search->**

**Code-?**

#include<iostream>

using namespace std;

int index(int a[],int n,int c,int ele)

{

if(c==0 && a[c]!=ele)

return -1;

else if(c==0 && a[c]==ele)

return c;

if(a[c]==ele)

return c;

else

return index(a,n,c-1,ele);

return -1;

}

int main()

{

int a[]={1,2,2,3,4,5,6};

cout<<index(a,7,6,2);

return 0;

}

**All occurence of an element in an array->**

**Code->**

#include<iostream>

using namespace std;

void index(int a[],int n,int c,int ele)

{

if(c==n)

return;

if(a[c]==ele)

cout<<c<<" ";

index(a,n,c+1,ele);

}

int main()

{

int a[]={1,2,2,3,4,5,6};

index(a,7,0,2);

return 0;

}

**If we want to store the result->**

**Code->**

#include<iostream>

using namespace std;

int index(int a[],int n,int c,int store[],int j,int ele)

{

if(c==n)

return j;

if(a[c]==ele)

{

store[j]=c;

return index(a,n,c+1,store,j+1,ele);

}

return index(a,n,c+1,store,j,ele);

}

int main()

{

int a[]={1,2,2,3,4,5,6};

int store[100];

int cnt=index(a,7,0,store,0,2);

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

cout<<store[i]<<" ";

return 0;

}

**Fast power->**

**Code->**

int fast\_power(int a,int n)

{

if(n==0)

return 1;

if(n==1)

return a;

int smaller\_power= fast\_power(a,n/2);

smaller\_power \*= smaller\_power;

if(n%2!=0)

return a\*smaller\_power;

else

return smaller\_power;

}

int main()

{

cout<<fast\_power(2,5);

return 0;

}

**Bubble sort recursive->**

**Code->**

#include<iostream>

using namespace std;

void swap(int &a,int &b)

{

int c=a;

a=b;

b=c;

}

void bubble\_sort(int a[],int n,int j)

{

if(n==1)

return;

if(j==n-1)

return bubble\_sort(a,n-1,0);

if(a[j]>a[j+1])

swap(a[j],a[j+1]);

bubble\_sort(a,n,j+1);

}

int main()

{

int a[]={1,6,5,4,3,8,9,0};

bubble\_sort(a,8,0);

for(int i=0;i<8;i++)

{

cout<<a[i]<<" ";

}

return 0;

}

**Merge sort->**

**Code->**

#include<iostream>

using namespace std;

void merges(int a[],int s,int e)

{

int i=s;

int mid=(s+e)/2;

int j=((s+e)/2)+1;

int k=s;

int temp[100];

while(i<=mid && j<=e)

{

if(a[i]<a[j])

{

temp[k++]=a[i++];

}

else

{

temp[k++]=a[j++];

}

}

while(i<=mid)

{

temp[k++]=a[i++];

}

while(j<=e)

{

temp[k++]=a[j++];

}

for(int x=0;x<k;x++)

{

a[x]=temp[x];

}

}

void mergesort(int a[],int s,int e)

{

if(s>=e)

return;

int mid=(s+e)/2;

mergesort(a,s,mid);

mergesort(a,mid+1,e);

merges(a,s,e);

}

int main()

{

int a[]={6,5,9,8,7,0,3,1,2,4};

mergesort(a,0,9);

for(int i=0;i<10;i++)

cout<<a[i]<<" ";

return 0;

}

**Inversion count->**

**Code->**

#include<iostream>

using namespace std;

int merges(int a[],int s,int e)

{

int cnt=0;

int i=s;

int mid=(s+e)/2;

int j=mid+1;

int k=s;

int temp[100];

while(i<=mid && j<=e)

{

if(a[i]<a[j])

{

temp[k++]=a[i++];

}

else

{

cnt+=mid-i+1;

temp[k++]=a[j++];

}

}

while(i<=mid)

{

temp[k++]=a[i++];

}

while(j<=e)

{

temp[k++]=a[j++];

}

for(int x=0;x<k;x++)

{

a[x]=temp[x];

}

return cnt;

}

int inversion\_count(int a[],int s,int e)

{

if(s>=e)

return 0;

int mid=(s+e)/2;

int x=inversion\_count(a,s,mid);

int y=inversion\_count(a,mid+1,e);

int z=merges(a,s,e);

return x+y+z;

}

int main()

{

int a[]={1,5,2,6,3,0};

cout<<inversion\_count(a,0,5);

}

**QUICK SORT**

**CODE->**

#include<iostream>

using namespace std;

void swap(int &a,int &b)

{

int c=a;

a=b;

b=c;

}

int partitions(int a[],int s,int e)

{

int i=s-1;

int j=s;

int pivot=a[e];

for(;j<=e-1;j++)

{

if(a[j]<=pivot)

{

i=i+1;

swap(a[i],a[j]);

}

}

swap(a[e],a[i+1]);

return i+1;

}

void quick\_sort(int a[],int s,int e)

{

if(s>=e)

{

return;

}

int p=partitions(a,s,e);

quick\_sort(a,s,p-1);

quick\_sort(a,p+1,e);

}

int main()

{

int a[]={2,6,1,5,4,8,0};

quick\_sort(a,0,6);

for(int i=0;i<=6;i++)

cout<<a[i]<<" ";

return 0;

}

**2048 problem**

**Code->**

#include<iostream>

#include<vector>

using namespace std;

void print(vector<string> s,int n)

{

if(n==0)

return;

print(s,n/10);

cout<<s[n%10]<<" ";

}

int main()

{

vector<string> s={"zero","one","two","three","four","five","six","seven","eight","nine"};

int n;

cin>>n;

print(s,n);

return 0;

}

**STRING TO INTEGER RECURSION**

**CODE->**

#include<iostream>

#include<string>

using namespace std;

int strtoint(char \*a,int i)

{

if(i>=0)

{

int sum=strtoint(a,i-1);

sum=sum\*10+(a[i]-'0');

return sum;

}

return 0;

}

int main()

{

char \*p={"1234"};

cout<<strtoint(p,3);

return 0;

}

**REPLACE pi with 3.14**

**Code->**

#include<iostream>

using namespace std;

void replace\_arr(char a[],int i)

{

if(a[i]=='\0' || a[i+1]=='\0')

return;

if(a[i]=='p' && a[i+1]=='i')

{

int j=i+2;

while(a[j]!='\0')

{

j++;

}

while(j>=i+2)

{

a[j+2]=a[j];

j--;

}

a[i]='3';

a[i+1]='.';

a[i+2]='1';

a[i+3]='4';

replace\_arr(a,i+4);

}

else

{

replace\_arr(a,i+1);

}

}

int main()

{

char a[1000];

cin>>a;

replace\_arr(a,0);

cout<<a;

return 0;

}

**Tiling problem->**

**Statement-> wall=4\*N**

**Tiles-> (4,1) & (1,4)**

**How many methods to fill wall?**

**Code->**

#include<iostream>

using namespace std;

int valTiles(int n)

{

if(n<=3)

return 1;

return (valTiles(n-4)+valTiles(n-1));

}

int main()

{

int n;

cin>>n;

cout<<valTiles(n)<<endl;

return 0;

}

**Ladder problem->**

**Code->**

#include<iostream>

using namespace std;

int ladder(int n,int k)

{

if(n==0)

return 1;

if(n<0)

return 0;

int ans=0;

for(int i=1;i<=k;i++)

ans+=ladder(n-i,k);

return ans;

}

int main()

{

int n,k;

cin>>n>>k;

cout<<ladder(n,k);

return 0;

}

**Friends problem-> n friends goes to party either single or couple so find total number of ways.**

**Code->**

#include<iostream>

using namespace std;

int ways(int n)

{

if(n==1 || n==0)

return 1;

return ways(n-1)+(n-1)\*ways(n-2);

}

int main()

{

int n;

cin>>n;

cout<<ways(n)<<endl;

return 0;

}

**Tower of Hanoi->**

**Code->**

#include<iostream>

using namespace std;

int TOH(int n,int source,int helper,int destination)

{ static int cnt=0;

if(n==0)

return 0;

int w=TOH(n-1,source,destination,helper);

cout<<"Moving disk "<<n<<" from "<<source<<" to "<<destination<<" using "<<helper<<endl;

cnt++;

int h=TOH(n-1,helper,source,destination);

return cnt;

}

int main()

{

int n;

cin>>n;

cout<<TOH(n,1,2,3);

return 0;

}

**SUBSEQUENCE PRINTING->**

**CODE->**

#include<iostream>

using namespace std;

int subsequence(char \*a,char \*out,int i,int j)

{

static int cnt=0;

if(a[i]=='\0')

{

out[j]=a[i];

cout<<out<<endl;

cnt++;

return cnt;

}

out[j]=a[i];

subsequence(a,out,i+1,j+1);

subsequence(a,out,i+1,j);

return cnt;

}

int main()

{

char \*a={"abc"};

char out[100];

cout<<subsequence(a,out,0,0);

cout<<endl<<"one substring is empty"<<endl;

return 0;

}

**GENERATE BRACKETS**

**CODE->**

#include<iostream>

using namespace std;

void generate\_brackets(int n,int i,char \*out,int open,int close)

{

if(i==2\*n)

{

out[i]='\0';

cout<<out<<endl;

return;

}

if(open<n)

{

out[i]='(';

generate\_brackets(n,i+1,out,open+1,close);

}

if(close<open)

{

out[i]=')';

generate\_brackets(n,i+1,out,open,close+1);

}

}

int main()

{

int n;

cin>>n;

char out[100];

generate\_brackets(n,0,out,0,0);

return 0;

}

**0-1 KNAPSACK**

**CODE->**

#include<iostream>

using namespace std;

int profit(int \*w,int \*p,int n,int c)

{

if(n==0 || c==0)

return 0;

int ans=0;

int inc=0,exc=0;

if(c>=w[n-1])

{

inc=p[n-1]+profit(w,p,n-1,c-w[n-1]);

}

exc=profit(w,p,n-1,c);

ans=max(inc,exc);

return ans;

}

int main()

{

int w[]={1,2,3,5};

int p[]={40,20,30,100};

int n=4;

int c=7;

cout<<profit(w,p,n,c);

return 0;

}

**KEYPAD STRING GENERATION**

**CODE->**

#include<iostream>

using namespace std;

char keypad[][10]={"","","abc","def","ghi","jkl","mno","pqrs","tuv","wxyz"};

void generate\_names(char \*in,char \*out,int i,int j)

{

if(in[i]=='\0')

{

out[i]='\0';

cout<<out<<endl;

return;

}

int digit=in[i]-'0';

if(digit==0 || digit==1)

{

generate\_names(in,out,i+1,j);

}

for(int k=0;keypad[digit][k]!='\0';k++)

{

out[j]=keypad[digit][k];

generate\_names(in,out,i+1,j+1);

}

return ;

}

int main()

{

char in[100];

cin>>in;

char out[100];

generate\_names(in,out,0,0);

return 0;

}

**GENERATE STRING FROM GIVEN NUMBER WHERE 1->26 IS MAPPED WITH A->Z . YOU CAN COMBINE ANY TWO CONTIGUOUS NUMBER**

**CODE->**

#include<iostream>

using namespace std;

void generate\_strings(char \*in,char \*out,int i,int j)

{

if(in[i]=='\0')

{

out[j]='\0';

cout<<out<<endl;

return;

}

int num1=in[i]-'0';

char ch=num1+'A'-1;

if(in[i+1]!='0'){

out[j]=ch;

generate\_strings(in,out,i+1,j+1);

}

if(in[i+1]!='\0')

{

int num2=in[i+1]-'0';

int fin=num1\*10+num2;

if(fin<=26)

{

ch=fin+'A'-1;

out[j]=ch;

generate\_strings(in,out,i+2,j+1);

}

}

}

int main()

{

char in[100];

char out[100];

cin>>in;

generate\_strings(in,out,0,0);

return 0;

}