Bangalore Institute of Technology

# Department of Computer Science and Engineering DESIGN AND ANALYSIS OF ALGORITHMS LAB (BCSL404)

1. **Design and implement C/C++ Program to find a subset of a given set S = {sl , s2,.....,sn} of n positive integers whose sum is equal to a given positive integer d.**

**AIM**: An instance of the Subset Sum problem is a pair (S, t), where S = {x1, x2, ..., xn} is a set of positive integers and t (the target) is a positive integer. The decision problem asks for a subset of S whose sum is as large as possible, but not larger than t.

**Algorithm**:

SumOfSub (s, k, r)

//Values of x[ j ], 1 <= j < k, have been determined

//Node creation at level k taking place: also call for creation at level K+1 if possible

// s = sum of 1 to k-1 elements and r is sum of k to n elements

//generating left child that means including k in solution

Set x[k] = 1

If (s + s[k] = d) then subset found, print solution

If (s + s[k] + s[k+1] <=d)

then SumOfSum (s + s[k], k+1, r – s[k])

//Generate right child i.e. element k absent

If (s + r - s[k] >=d) AND (s + s[k+1] )<=d

THEN { x[k]=0;

SumOfSub(s, k+1, r – s[k])

**Program**

#include<stdio.h>

#define MAX 10

int s[MAX],x[MAX],d;

void sumofsub(int p,int k,int r)

{

int i;

x[k]=1;

if((p+s[k])==d)

{

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

if(x[i]==1)

printf("%d ",s[i]);

printf("\n");

}

else

if(p+s[k]+s[k+1]<=d)

sumofsub(p+s[k],k+1,r-s[k]);

if((p+r-s[k]>=d) && (p+s[k+1]<=d))

{

x[k]=0;

sumofsub(p,k+1,r-s[k]);

}

}

int main()

{

int i,n,sum=0;

printf("\nEnter the n value:");

scanf("%d",&n);

printf("\nEnter the set in increasing order:");

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

scanf("%d",&s[i]);

printf("\nEnter the max subset value:");

scanf("%d",&d);

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

sum=sum+s[i];

if(sum<d || s[1]>d)

printf("\nNo subset possible");

else

sumofsub(0,1,sum);

return 0;

}

**Input/output:**

Enter the n value:9

Enter the set in increasing order:1 2 3 4 5 6 7 8 9

Enter the max subset value:9

1 2 6

1 3 5

1 8

2 3 4

2 7

3 6

4 5

9

1. **Design and implement C/C++ Program for N Queen's problem using Backtracking.**

**Aim:** To implement N Queens Problem using Back Tracking

**Definition:**

The object is to place queens on a chess board in such a way as no queen can capture another one in a single

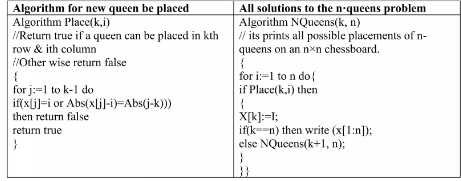
move.

Recall that a queen can move horizontal, vertical, or diagonally an infinite distance

This implies that no two queens can be on the same row, col, or diagonal

We usually want to know how many different placements there are

**Using Backtracking Techniques**



**Program**

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

#define True 1

#define False 0

int x[10]; **// Initial solution vector**

int n;

int count; //**Number of solutions**

void printsolution()

{

char c[10][10];

printf("Solution %d:\n\n",++count);

**// No queen has placed initially**

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

{

for(int j=1;j<=n;j++)

{

c[i][j]='X';

}

}

**// Place the queens on the chess board**

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

{

c[i][x[i]]='Q';

}

**// Place where the queens have been placed on the chess board**

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

{

for(int j=1;j<=n;j++)

{

printf("%c",c[i][j]);

}

printf("\n");

}

}

**// Function to check whether the queens can be placed successfully or not**

int place(int k,int xk)

{

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

{

//check whether two queens attach vertically or diagonally

if((x[i]==xk)||(abs(i-k)==abs(x[i]-xk)))

{

return False; **//Queen cannot be placed in the kth column**

}

}

return True; **//kth queen can be successfully placed**

}

void nqueen(int k)

{

int j;

for(int j=1;j<=n;j++)

{

if(place(k,j)) **//Try to place in various columns**

{

x[k]=j; **//Queen can be placed**

if(k==n)

printsolution(),printf("\n");

else

nqueen(k+1); **//Place the next queen in next row**

}

}

}

void main()

{

printf("enter the number of queens\n");

scanf("%d",&n);

nqueen(1);

}

**Input/Output:**

Enter the no. of queens:4

Solution 1

XQXX

XXXQ

QXXX

XXQX

Solution 2

XXQX

QXXX

XXXQ

XQXX