Study, analyse and implement tower of hannoi where the aim it to more entire stack to another rod for n = 3 and understand the concept of recursion.

Algorithm

```
procedure hannoi(disk,source,aux,destination)
    if disk == 1 then
        move disk from source to destination
    else
        hannoi(n-1,source,destination,aux)
        move disk from source to destination
        hannoi(n-1, aux, source, destination)
    end_if
end_procedure
```

program

```
#include<stdio.h>
int towerOfHannoi(int n, char source, char aux, char destination)
{
    if(n == 1)
    {
        printf("move disk from %c to %c\n", source, destination);
        return 0;
    }
    else
        towerOfHannoi(n-1, source, destination, aux);
    printf("move disk from %c to %c\n", source, destination);
    towerOfHannoi(n-1, aux, source, destination);
}
void main()
{
    hannoi(3,'A','B','C');
}
```

```
O/P:
    move disk from A to C
    move disk from A to B
    move disk from C to B
    move disk from A to C
    move disk from B to A
    move disk from B to C
    move disk from A to C
```

Explain the knapsack algorithm to find an optimal solution of getting maximum profit and implement the program

Algorithm

```
local real cu;            [remaining knapsack capacity]
local int i, n;
x ← 0;                   [initilize solution vector to zero]
cu ← M;
for i ← 0 to n-1 do
    if W[i] > cu then
        [no space remaining]
        break
    end_if
    x[i] ← 1;
    cu ← cu - W[i];
end_for
if i<= n then
    x[i] ← cu / W[i];
end_if

where:
    p[0:n-1]  :   profit ratios
    W[0:n-1]  :   weight ratios
    p[i]/W[i] :   arrange in ascending order
        M     :   knapsack capacity
    X[0:n-1]  :   solution vector
    these are all real numbers

```

Program

```
#include<stdio.h>
void knapsack(int n, float weight[], float profit[], float capacity)
{
    float x[20], tp = 0;
    int i, j, cu;
    cu = capacity;

    for (i = 0; i < n; i++)
        x[i] = 0.0;

    for (i = 0; i < n; i++)
        if(weight[i] > cu)
            break;
        else
        {
            x[i] = 1.0;
            tp = tp + profit[i];
            cu = cu - weight[i];
        }
    if(i <= n)
        x[i] = cu / weight[i];

    tp = tp + x[i] * profit[i];
    printf("\nMaximum profit: %f\n", tp);

}

int main()
{
    float weight[20], profit[20], capacity;
    int num, i, j;
    float ratio[20], temp;

    printf("\nEnter the number of objects: ");
```

```
    scanf("%d", &num);

    printf("Enter the weights of objects: ");
    for(i = 0; i < num; i++)
        scanf("%f", &weight[i]);

    printf("Enter the profit of objects: ");
    for(i = 0; i < num; i++)
        scanf("%f", &profit[i]);

    printf("Enter the capacity of knapsack: ");
    scanf("%f", &capacity);

    for(i = 0; i < num; i++)
        ratio[i] = profit[i] / weight[i];

    for(i = 0; i < num; i++)
        for (int j = i + 1; j < num; j++)
            if(ratio[i] < ratio[j])
            {
                temp = ratio[j];
                ratio[j] = ratio[i];
                ratio[i] = temp;

                temp = weight[j];
                weight[j] = weight[i];
                weight[i] = temp;

                temp = profit[j];
                profit[j] = profit[i];
                profit[i] = temp;
            }
    knapsack(num, weight, profit, capacity);
    return 0;
}
```

```
O/P:
    Enter the number of objects: 3
    Enter the weights of objects: 18 15 10
    Enter the profit of objects: 25 24 15
    Enter the capacity of knapsack: 20

    Maximum profit: 31.500000
```

Develop a program to implement floyd alforithm which will produce the shortest distance between all vertex pair of weighted graph.

```
let G = <N,A> be a directed graph
N -> Set of Node
A -> Set of Edge
Each edge has an associated non-negative length & we calculate shortest path between
each pair of node

function floyd(L[1:n][1:n] : array[1:n][1:n]
               [1:n] arrayD[1:n][1:n])
    arrayD ← L;
    for k ← 1 to n do
        for i ← 1 to n do
```

```
            for j ← 1 to n do
                D[i][j] = min(D[i][j], D[i][k]+[k][j])
                return D;
```

Program

```
#include <stdio.h>
int min(int a, int b){
    if(a<b)
        return(a);
    else
        return(b);
}

void floyd(int d[10][10],int n){
    int i,j,k;
    for (k=1;k<=n;k++)
        for(i=1;i<=n;i++)
            for(j=1;j<=n;j++)
                d[i][j]=min(d[i][j],d[i][k]+d[k][j]);
}

int main()
{
    int i,j,k,n,a[10][10],d[10][10];
    printf("Enter number of virtices: ");
    scanf("%d",&n);

    printf("Enter values or mattrix: \n");
    printf("Enter 0 for self loop nd 999 for node with no value:\n");

    for(i=1;i<=n;i++){
        for(j=1;j<=n;j++){
            scanf("%d",&a[i][j]);
            d[i][j]=a[i][j];
        }
    }

    floyd(d,n);

    printf("All pairs shortest path :\n");
    for(int i=1;i<=n;i++){
        for (j=1;j<=n;j++)
            printf("%d\t",d[i][j]);
        printf("\n");

    }

    return 0;
}
```

```
O/P:
    Enter number of virtices: 3
    Enter values or mattrix:
    Enter 0 for self loop nd 999 for node with no value:
    0 4 11
    6 0 2
    3 999 0
    All pairs shortest path :
    0  4  6
    5  0  2
```

```
    3  7  0
```

Write a program to implement binary search algorithm using recursion

Algorithm

```
binarySearchR(a,low,high,item)
    local int mid;
    mid ← floor((low+high) / 2)
    if low > high then
        return false
    if a[mid] == item then
        return mid
    else if a[mid] > item then
        binarySearchR(a, low, mid-1, item)
    else
        binarySearchR(a, mid+1, high, item)
    end
```

Program

```
#include <stdio.h>

int binarySearch(int array[], int x, int low, int high) {
  if (high >= low) {
    int mid = low + (high - low) / 2;

    // If found at mid, then return it
    if (array[mid] == x)
      return mid;

    // Search the left half
    if (array[mid] > x)
      return binarySearch(array, x, low, mid - 1);

    // Search the right half
    return binarySearch(array, x, mid + 1, high);
  }

  return -1;
}

int main(void) {
  int array[] = {3, 4, 5, 6, 7, 8, 9};
  int n = sizeof(array) / sizeof(array[0]);
  int x = 4;
  int result = binarySearch(array, x, 0, n - 1);
  if (result == -1)
    printf("Not found");
  else
    printf("Element is found at index %d", result);
}
```

```
O/P:
    Element is found at index 1
```

implement longest common subsequesce problem using dynamic programming

Algorithm

```
[c[i,j] maximum length array]

m = length(x)
n = length(y)

for i = 0 to m do
    c[i][0] = 0
end_for

for j = 0 to m do
    c[0][j] = 0
end_for

for i = 1 to m do
    for j = 1 to n do
        if x[i] = y[i] then
            c[i,j] = c[i-1,j-1] + 1
        else if c[i-1,j] >= c[i,j-1] then
            c[i,j] = c[i-1,j]
        else
            c[i,j] = c[i,j-1]
        end_if
    end_for
end_for
return c and b
```

Program

```
#include<stdio.h>
#include<string.h>
int i,j,m,n,c[20][20];
char x[20],y[20],b[20][20];
void lcs()
{
    m=strlen(x);
    n=strlen(y);
    for(i=0;i<=m;i++)
        c[i][0]=0;
    for(i=0;i<=n;i++)
        c[0][i]=0;

    //c, u and l denotes cross, upward and downward directions respectively
    for(i=1;i<=m;i++)
        for(j=1;j<=n;j++)
        {
            if(x[i-1]==y[j-1])
            {
                c[i][j]=c[i-1][j-1]+1;
                b[i][j]='c';
            }
            else if(c[i-1][j]>=c[i][j-1])
            {
```

```
                c[i][j]=c[i-1][j];
                b[i][j]='u';
            }
            else
            {
                c[i][j]=c[i][j-1];
                b[i][j]='l';
            }
        }
}
void print(int i,int j)
{
    if(i==0 || j==0)
        return;
    if(b[i][j]=='c')
    {
        print(i-1,j-1);
        printf("%c",x[i-1]);
    }
    else if(b[i][j]=='u')
        print(i-1,j);
    else
        print(i,j-1);
}
int main()
{
    printf("Enter 1st sequence:");
    scanf("%s",x);
    printf("Enter 2nd sequence:");
    scanf("%s",y);
    printf("\nThe Longest Common Subsequence is ");
    lcs();
    print(m,n);
    return 0;
}
```

```
O/P:
    Enter 1st sequence:aditya
    Enter 2nd sequence:aytida

    The Longest Common Subsequence is ada
```

Develop a program to print all the node reachable a given shorting node in diagram using depth first search

```
procedure dfSearch(G)
    for each V € N do
        mark[v] ← not visited
    for each V € N do
        if mark[v] ≠ visited then
            dfs(v)
        end_if
    end_for
end_procedure


procedure dfs(v)
    {node v has not visited previously}
```

```
    mark[v] ← visited
    for each node w adjacent to v do
        if mark[v] ≠ visited then
            dfs(v)
        end_if
    end_for
end_procedure
```

Program

```
#include<stdio.h>
int reach[20],mat[20][20],n;
void DFS(int v)
{
    int i;
    reach[v] = 1;
    for(i=1;i<=n;i++)
        if(mat[v][i] && !reach[i])
        {
            printf("%d => %d\n",v,i);
            DFS(i);
        }
}
int main()
{
    int count=0;
    printf("Enter the number of vertices: ");
    scanf("%d", &n);

    for (int i = 1; i <= n;i++)
        for (int j = 1; j <= n; j++)
            mat[i][j] = 0;

    for (int i = 1; i <= n;i++)
        for (int j = 1; j <= n; j++)
            scanf("%d", &mat[i][j]);

    DFS(1);
    printf("\n");
    for (int i=1;i<=n;i++) {
        if(reach[i])
            count++;
}
    if(count==n)
        printf("\n Graph is connected");
    else
        printf("\n Graph is not connected");
}
```

```
O/P:
    Enter the number of vertices: 6
    0 1 0 1 0 0
    0 0 1 0 1 0
    0 0 0 0 1 1
    0 1 0 0 0 0
    0 0 0 0 0 0
    0 0 0 0 0 1

    1 => 2
    2 => 3
    3 => 5
```

```
    3 => 6
    1 => 4


    Graph is connected
```

Write a program to implement quicksort using divide and conquer

Algorithm

```
quicksort(int a[], int l, int u)
{
    int j;
    if (l < u)
    {
        j = partition(a,l,u);
        quicksort(a,l,j-1);
        quicksort(a,j+1,u);
    }
}

int partition(int a[], int l, int u)
{
    int v, i, j, temp;
    v = a[i];
    i = 1;
    j = u + 1;
    do{
        do{i++;}while(a[i]<u && i<=u);
        do{j--;}while(a[j]>u);
        if(i<j)
        {
            temp = a[i];
            a[i] = a[j];
            a[j] = temp;
        }
    }while(i<j);
    swap[j] with piavote
}
```

Program

```
#include<stdio.h>

void swap(int *a , int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

int partition(int *arr , int lb , int ub)
{
    int pivot = lb;
    int i = lb , j = ub + 1;
    int temp;
```

```
    while(i<j)
    {
        do{
            i++;
        }while (arr[i]<=arr[pivot] && i<ub);

        do{
            j--;
        }while (arr[j]>arr[pivot] && j>lb);

        if(i<j)
        {
            swap(&arr[i] , &arr[j]);
        }

    }
    swap(&arr[pivot] , &arr[j]);

    return j;

}

void quickSort(int *arr , int lb , int ub)
{
    int j;
    if(lb<ub)
    {
        j = partition(arr,lb,ub);
        quickSort(arr,lb,j-1);
        quickSort(arr,j+1,ub);

    }

}


int main()
{
    int n;
    printf("Enter size of array : ");
    scanf("%d" , &n);
    int arr[n];
    printf("Enter element in unsorted order:");

    for(int i=0 ; i<n ; i++)
    {
        scanf("%d" , &arr[i]);
    }


    quickSort(arr , 0 , n-1);
    printf("The sorted array is:\t");
    for(int i=0 ; i<n ; i++)
    {
        printf("%d\t" , arr[i]);
    }

    return 0;
}
```

```
O/P:
    Enter size of array : 6
```

```
    Enter element in unsorted order:22 1 10 5 60 45
    The sorted array is:    1       5       10      22      45      60
```

Write a program to implement merge sort using divide and conquer

Algorithm

```
Algorithm: mergesort(low, high)

    global array a[low:high]
    if low < high then
        mid ← floor((low+high)/2)
        mergesort(low, mid)
        mergesort(mid+1, high)
        merge(low, mid, high)
    end_if
end




Algorithm: merge(low, mid, high)

    global array a[low:high]
    local int h, i, j, k;
    local array b[low:high]
    h ← low
    i ← low
    j ← mid+1
    while(h <= mid && j <= u) do
        if a[h] < a[j] then
            b[i] ← a[h]
            h ← h + 1
        else
            j ← j + 1
        end_if
    end_while
    if h > mid then
        for k ← j to high do
            b[i] ← a[k]
            i ← i + 1
        end_for
    else
        for k ← h to mid do
            b[i] ← a[k]
            i ← i + 1
        end_for
    end_if
    [copy merged array back to a[] ]
    for k ← low to high do
        a[k] ← b[k];
    end_for
end
```

Program

```c
#include<stdio.h>

void merge(int arr[100], int l, int mid, int u)
{
    int h, i, j, k;
    int b[100];
    h = l;
    j = mid+1;
    i = 0;

    while(h <= mid && j <= u)
    {
        if(arr[h] <= arr[j])
        {
            b[i] = arr[h];
            h++;
        }
        else
        {
            b[i] = arr[j];
            j++;
        }
        i = i + 1;
    }

    while(j <= u)
    {
        b[i] = arr[j];
        j++;
        i++;
    }

    while(h <= mid)
    {
        b[i] = arr[h];
        i++;
        h++;
    }

    int z = 0;
    for(k = l; k <= u; k++)
    {
        arr[k] = b[z];
        z++;
    }
}

void merge_sort(int arr[100], int l, int u)
{
    if(l < u)
    {
        int mid = (l+u)/ 2;
        merge_sort(arr, l, mid);
        merge_sort(arr, mid+1, u);
        merge(arr, l, mid, u);
    }
}

int main()
{
    int n;
    int arr[100];
    printf("Enter the size of array : ");
    scanf("%d",&n);
```

```
    scanf("%d",&n);

    printf("Enter all elements in unsorted order : ");
    for(int i = 0; i < n; i++)
        scanf("%d",&arr[i]);

    merge_sort(arr, 0, n-1);

    printf("Array in sorted order : ");
    for(int i = 0; i < n; i++)
        printf("%d ",arr[i]);
    printf("\n");

    return 0;
}
```

```
O/P:
    Enter the size of array : 6
    Enter all elements in unsorted order : 15 24 65 155 36 78
    Array in sorted order : 15 24 36 65 78 155
```

Write a program to implement Breadth First Search using backtracking

Algorithm

```
procedure BFS(v)
    ∅ ← empty Queue
    mark[v] ← visited
    enqueue v into Queue
    while Queue is not empty do
        u ← first(Queue)
        enqueue u from Queue
        for each node w adjacent to u do
            if mark[w] ≠ visited then
                mark[w] ← visited
                enqueue w into Queue
            end_if
        end_for
    end_while
```

Program

```
#include<stdio.h>

void BFS(int matrix[100][100], int visited[100], int queue[100], int node, int x)
{

    int front = 0;
    int rear = 0;
    queue[front] = x;
    visited[x] = -1;


    while(front <= rear)
    {
        int temp = queue[front];
        front++;
```

```
        printf("%d ",temp);

        x = temp;
        for(int i = 0; i < node; i++)
        {
            if(visited[i+1] == 0 && matrix[x-1][i] == 1)
            {
                rear++;
                queue[rear] = i+1;
                visited[i+1] = 1;
            }
        }
    }

}

int main()
{
    int visited[100];
    int queue[100];
    int node;
    printf("Enter number of node : ");
    scanf("%d",&node);

    for(int i = 0; i <= node; i++)
    {
        visited[i] = 0;
        queue[i] = -1;
    }

    int matrix[100][100];
    printf("Enter Values : \n");
    for(int i = 0; i < node; i++)
        for(int j = 0; j < node; j++)
            scanf("%d",&matrix[i][j]);



    printf("Output : ");
    BFS(matrix, visited, queue, node, 1);
    printf("\n");
    return 0;
}
```

```
O/P:
    Enter number of node : 6
    Enter Values :
    0 1 0 1 0 0
    0 0 1 0 1 0
    0 0 0 0 1 1
    0 1 0 0 0 0
    0 0 0 0 0 0
    0 0 0 0 0 1
    Output : 1 2 4 3 5 6
```

Implement C program for n-queen problem using backtracking method

Algorithm

```
PLACE(k)

global integer X[0:k-1]
local integer i,k;
for i ← 0 to k-1 do
    if x(i) = X(k) or ABS(X(i) - X(k)) == ABS(i - k) then
        [two queens in the same col. or same diag. or same row]
        return false
    end_if
end_for
return true




NQueen(n)

[general n queen problem, n > 3]
integer k;
global X[0:n - 1];
x(1) ← 0;
k ← 1;
while k> 0 do
    X(k) ← X(k)+1 ;
    while X(k) ≤ n and Not PLACE (k) do
        X(k) ← X(k) + 1;
    end_while

    if X(k) ≤ n then
        if k = n then
            [is solution complete?]
            print (X);
        else
            k ← k + 1; [go to the next row]
            X(k) ← 0;
        end_if
    else
        k ← k - 1;
    end_if
end_while
```

Program

```
#include<stdio.h>
#include<math.h>
int a[30],count = 0;
int place(int pos)
{
    int i;
    for(i=1; i<pos; i++)
        if(a[i]==a[pos] || abs(a[i] - a[pos]) == abs(i - pos))
            return 0;
    return 1;
}

void print_solution(int n)
{
    int i,j;
    count++;
    printf("\nSolution %d: \n",count);
    for(i=1; i<=n; i++)
```

```
    for(i=1; i<=n; i++)
    {
        for(j=1; j<=n; j++)
            if(a[i] == j)
                printf("Q\t");
            else
                printf("*\t");
        printf("\n");
    }
}

void queen(int n)
{
    int k = 1;
    a[k] = 0;
    while(k != 0)
    {
        a[k] = a[k] + 1;
        while(a[k] <=n && !place(k))
            a[k]++;
        if(a[k] <= n)
            if(k == n)
                print_solution(n);
            else
            {
                k++;
                a[k] = 0;
            }
        else
            k--;
    }
}
void main()
{
    int i, n = 4;
    queen(n);
    printf("\nTotal solution %d\n",count);
}
```

```
O/P:
    Solution 1:
    *       Q       *       *
    *       *       *       Q
    Q       *       *       *
    *       *       Q       *

    Solution 2:
    *       *       Q       *
    Q       *       *       *
    *       *       *       Q
    *       Q       *       *

    Total solution 2
```