15/4/21

Page No: 1
Date: / /

① Recursive program for GCD.

```
#include <stdio.h>
int gcd (int n1, int n2) {
    if (n2 != 0) {
        return gcd (n2, n1 % n2); }
    else
        return n1;
}
int main () {
    int n1, n2;
    printf ("Enter two for integers : \n");
    scanf ("%d %d", &n1, &n2);
    printf ("GCD of %d & %d is %d",
            n1, n2, gcd (n1, n2));
    return 0;
}
```

② Iterative program for GCD

```
#include <stdio.h>
int main () {
    int n2, n1, K;
    printf ("Enter two values :- \n");
    scanf ("%d %d", &n1, &n2);
    while (n2 != 0) {
        K = n2;
```

```
        m2 = m1 % m2;
        m1 = k;
    }
    printf("GCD = %d", m1);
    return 0;
}
```

(3) Linear & binary search using Recursion

```
#include <stdio.h>
#include <stdlib.h>
int linear(int a[], int l, int r, int Key){
    if (r < l) {
        return -1;
    if (a[l] == Key)
        return l;
    if (a[r] == Key)
        return r;
    return linear(a, l+1, r-1, Key);
}
int binary(int a[], int first, int
                last, int key) {
    if (last >= first) {
        int m = first + (last - first)/2;
        if (a[m] == key) {
            return m;}
```

```
    if (a[m] > key) {
        return binary(a, first, m-1, key);
    }
    return binary(a, m+1, last, key);
}
return -1;
}
int main() {
    int a[200], i, choice, key, m, res;
    printf("Enter size of array : ");
    scanf("%d", &m);
    printf("Enter values : ");
    for(i=0; i<m; i++){
        scanf("%d", &a[i]);
    }
    for(;;){
        printf("Enter value to find \n");
        scanf("%d", &key);
        printf("1. Linear 2 Binary ");
        scanf("%d", &choice);
        switch(choice){
            case 1: res = linear(a, 0, m-1, key);
                if(res != -1) { printf("%d is
                    at %d", key, res+1);
                else { printf("%d Not found", key);
                break;
```

```
case 2 : res = binary(a, 0, n-1, key);
    if (res == -1) {
        printf("Element Not found"); }
    else {
        printf("Element found at %d",
                (res+1));
    }
    break;
default : exit(0);
}
}
return 0;
}
```

22-4-21

Q-> Selection and Bubble Sort

```
#include <stdio.h>
#include <stdlib.h>
void swap(int *x, int *y){
    int temp;
    temp = (*x);
    (*x) = (*y);
    (*y) = temp;
}
void bubble(int a[], int n){
    int i=0, j=0;
    int swapped;
    while(i != n-1){
        swapped = 0;
        for(j=0; j<n-i-1; j++){
            if(a[j] > a[j+1]){
                swap(&a[j], &a[j+1]);
                swapped = 1;
            }
        } i++;
        if(swapped == 0){ return; }
    }
}
void selection(int a[], int n){
    int i, j, min;
```

```
for (i = 0; i < n-1; i++) {
    min = i;
    for (j = i+1; j < n; j++) {
        if (a[j] < a[min]) {
            min = j;
        }
    } swap (&a[min], &a[i]);
}
}
void display (int a[], int n) {
    int i;
    for (i = 0; i < n; i++) {
        printf ("%d", a[i]);
    }
}
int main() {
    int a[10], i, choice, n;
    for (;;) {
        printf ("Size of array");
        scanf ("%d", &n);
        printf ("Enter array");
        for (i = 0; i < n; i++) {
            scanf ("%d", &a[i]);
        }
        printf ("1 Bubble 2. Selection");
```

```
scanf("%d", &choice);
switch (choice) {
    case 1: printf("Bubble sort");
            bubble(a, n);
            display(a, n);
            break;
    case 2: selection(a, n);
            display(a, n);
            break;
    default: exit(0);
  }
} return 0;
}
```

29/4/21

Q> Bubble sort and selection sort time complexity programs.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void swap (int *x, int *y) {
    int temp;
    temp = (*x);
    (*x) = (*y);
    (*y) = temp;
}
void bubble (int a[], int n) {
    int i = 0, j = 0;
    int swapped;
    while (i != n - 1) {
        swapped = 0;
        for (j = 0; j < n - i - 1; j++) {
            if (a[j] > a[j + 1]) {
                swap (&a[j], &a[j + 1]);
                swapped = 1;
            }
        }
        i++;
        if (swapped == 0) { return; }
    }
}
```

```
void selection(int a[], int n) {
    int i, j, min;
    for(i = 0; i < n-1; i++) {
        min = i;
        for(j = i+1; j < n; j++) {
            if(a[j] < a[min]) {
                min = j;
            }
        } swap(&a[min], &a[i]);
    }
}
void display(int a[], int n) {
    int i;
    for(i = 0; i < n; i++) {
        printf("%d ", a[i]);
    }
}
int main() {
    int a[5000], i, c, n = 0, j, k;
    printf("1. Selection sort \n2. Bubble sort");
    scanf("%d", &i);
    printf("Enter no. of times u want to
        sort : ");
    scanf("%d", &j);
    for(n; n < j; n+1) {
```

```
printf("Enter size : ");
scanf("%d", &K);
for(c = 0; c < K; c++) {
    a[c] = rand();
}

double ss_time = 0.0;
clock_t begin = clock();
if(i == 1) { selection(a, K); }
else if(i == 2) { bubble(a, K); }
else { exit(0); }
display(a, K);
clock_t end = clock();
ss_time += (double)(end - begin) / CLOCKS_PER_SEC;
printf("n = %d : %f", K, ss_time);
}

return 0;
}
```

6|5|21

6 -> Tower of Hanoi

```
#include <stdio.h>
#include <time.h>
#include <math.h>
void towers(int n, char s, char t, char d){
    if(n == 1){
        printf("Move disk from %c to %c\n", s, d);
        return;
    }
    towers(n-1, s, d, t);
    printf("Move disk %d from %c to %c\n", n, s, d);
    towers(n-1, t, s, d);
}
int main(){
    int n;
    printf("Enter no of disks : ");
    scanf("%d", &n);
    double toh_time = 0.0;
    clock_t begin = clock();
    towers(n, 'S', 'T', 'D');
    printf("\n Total Steps : %lf", (pow(2,n)-1));
    clock_t end = clock();
    toh_time += (double)(end - begin)/CLOCKS_PER_SEC;
    printf("\n\n n = %d\t Time = %lf\n\n", n, toh_time);
    return 0;
}
```

Q→ DFS Traversal

```
#include<stdio.h>
#include<time.h>
int G[10][10], v[10], n, a[1][10];
void dfs(int i){
    int j;
    printf("\n%d", i);
    v[i] = 1;
    for(j=0; j<n; j++){
        if(!v[j] && G[i][j]==1)
            dfs(j);
    }
}
void dfs_c(int n, int G[10][10], int m, int s[]){
    int y;
    s[m] = 1;
    for(y=0; y<n; y++){
        if((G[m][y]==1) && (s[y]==0))
            dfs_c(n, G, y, s);
    }
}
int main(){
    int i, j, con, s[10], flag;
    printf("Vertices: ");
    scanf("%d", &n);
```

```
printf(" Enter matrix : ");
for(i = 0; i<m; i++){
    printf(" Enter row %d :- \n", i+1);
    for(j=0; j<m; j++)
        scanf("%d", &a[i][j]);
}
for(i = 0; i<m; i++)
        w[i] = 0;
printf(" DFS Order :- ");
double tot_time = 0.0;
clock_t begin = clock();
dfs(0);
con = 0;
for(j=0; j<m; j++){
    for(i=0; i<m; i++)
            s[i] = 0;
    dfs_c(m, a, j, s);
    flag = 0;
    for(i=0; i<m; i++){
        if(s[i] == 0)
            flag = 1;
    }
    if(flag == 0)
        con = 1;
}
```

```
if (con == 1)
    printf("Graph is Connected");
else { printf("Graph isnt connected"); }
clock_t end = clock();
tot_time += (double)(end - begin)/(clocks_per_sec);
printf("m = %d \t Time = %lf", m, tot_time);
return 0;
}
```

13-5-21

Q=> BFS Traversal

```
#include <stdio.h>
#include <time.h>
int a[10][10], q[10], visited[10], n,
f = 0, r = -1;
void bfs(int v){
    int i;
    for (i = 1; i <= n; i++)
        if (a[v][i] && !visited[i])
            q[++r] = i;
    if (f <= r) {
        visited[q[f]] = 1;
        bfs (q[f++]);
    }
}
int main () {
    int v, i, j;
    printf ("\n Enter the no. of vertices: ");
    scanf ("%d", &n);
    printf ("\n Enter matrix : \n");
    for (i = 1; i <= n; j++)
        printf ("Enter row %d : \n", i);
        for (j = 1; j <= n; j++)
            scanf ("%d", &a[i][j]);
}
```

```
for (i=1; i<=n; i++) {
    q[i] = 0;
    visited[i] = 0;
}
printf("Enter beginning vertex : ");
scanf("%d", &v);
double bfs_time = 0.0;
clock_t begin = clock();
bfs(v);
clock_t end = clock();
bfs_time += (double)(end - begin) / CLOCKS_
                                PER_SEC;
printf("Visitable Nodes are : \n");
for(i=1; i<=n; i++) {
  if(visited[i])
    printf("%d", i);
  else
    printf("\n BFS isnt possible");
printf("\n\n n= %d \n Time : %f \n", n,
                bfs_time);
return 0;
}
```

d) Insertion Sort

```
#include<stdio.h>
```

```
#include<time.h>
void insertion(int a[], int n){
    for(int i=1; i<n; i++){
        int j, key;
        key = a[i];
        j = i-1;
        while(key<a[j] && j>=0){
            a[j+1] = a[j];
            --j;
        }
        a[j+1] = key;
    }
}
void display(int a[], int n){
    for(int i=0; i<n; i++){
        printf("%d ", a[i]);
    }
}
int main(){
    int n, i, a[10];
    printf("Enter size of array: ");
    scanf("%d", &n);
    printf("Enter values of array\n");
    for(i=0; i<n; i++){
        scanf("%d", &a[i]);
    }
```

```
double is_time = 0.0;
clock_t begin = clock();
insertion(a, n);
clock_t end = clock();
is_time += (double)(end - begin) /
                    CLOCKS_PER_SEC;
display(a, n);
printf("\n\n n = %d \t Time : %f\n",
                    n, is_time);
return n;
}
```

20/5/21

Page No: 99
Date:

Q → Topological Sorting

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int a[10][10], indegree[10], front = -1, rear = -1,
stack[10], t[10], k, n;
void get(int n) {
    int i,j;
    printf("Enter adjacency matrix : \n");
    for(i = 0; i < n; i++) {
        printf("Enter row %d \n", (i+1));
        for(j = 0; j < n; j++)
            scanf("%d", &a[i][j]);
    }
}
void cal_indegree(int n) {
    int i, j;
    for(i = 0; i < n; i++) {
        for(j = 0; j < n; j++) {
            indegree[i] = indegree[i] + a[j][i];
}
void push(int x) {
    if(front == -1 && rear == -1)
        front = rear = 0;
    else if(rear == n-1) { return; }
```

```
    else { rear++; }
    stack[rear] = x;
}
int pop() {
    int x;
    if(front == -1 || front > rear) { return -1; }
    x = stack[front];
    if(front == rear || front > rear) {
        front = rear = -1; }
    else { front++; }
    return x;
}
int main() {
    int i, n;
    printf("Enter no. of vertices : ");
    scanf("%d", &n);
    get(n);
    for(i=0; i<n; i++) {
        indegree[i] = 0; }
    double t = time = 0.0;
    clock_t begin = clock();
    cal_indegree(n);
    for(int i=0; i<n; i++)
        if(indegree[i] == 0) {
            push(i);
```

```
while (front != -1) {
    int u = pop();
    if (u == -1) { pop(t) break; }
    t[k] = u;
    k++;
    for (v = 0; v < n; v++) {
        if (a[v][u] == 1) {
            ((indegree[v])--);
            if (indegree[v] == 0) {
                push(v); }
        }
    }
}
clock_t end = clock();
t_time += (double)(end - begin) / (clocks_PER_SEC);
printf("Order :");
for (i = 0; i < k; i++) {
    printf("%d\t", t[i]); }
printf("\n for matrix on = %d \t
    Time = %f, n, t_time);
return 0;
}
```

27/5/21

Q → Merge Sort

```
#include <stdio.h>
#include <time.h>
void merge (int a[], int l, int m, int r){
    int i, j, k, c[10000];
    i = k = l;
    j = mid + 1;
    while (i <= mid && j <= r) {
        if (a[i] < a[j]) {
            c[k] = a[i];
            ++k;
            ++i; }
        else { c[k] = a[j];
            ++k, ++j; }
    }
    if (i > mid) {
        while (j <= r) {
            c[k] = a[j];
            ++k, ++j; }
    }
    if (j > r) {
        while (i <= mid) {
            c[k] = a[i];
            ++k, ++i; }
    }
```

```
    for (i=l; i<=r; i++)
        a[i] = c[i];
}
void mergeSort (int a[], int l, int r){
    int mid;
    if (l<r){
        mid = (l+r)/2;
        mergeSort (a, l, mid);
        mergeSort (a, mid+1, r);
        merge (a, l, mid, r);
    }
}
int main () {
    int a[10000], n, i;
    printf ("Enter no. of elements : ");
    scanf ("%d", &n);
    printf ("Enter array elements");
    for (i=0; i<n; i++)
        a[i] = rand ();
    clock_t begin = clock ();
    mergeSort (a, 0, n-1);
    clock_t end = clock ();
    double time = 0.0;
    time = (double) (end - begin) / CLOCKS_PER_SEC;
    printf ("\n Sorted Array :- ");
```

```
for (i = 0 ; i < n ; i++)
    printf("%d ", a[i]);
printf("Time : %f", n - time);
return 0;
}
```

3|6|21

Q → Quick sort

```
#include <stdio.h>
#include <time.h>
void swap (int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
int partition(int a[], int l, int r){
    int pivot = a[r], pindex = l;
    for(int i = l; i < r; i++) {
        if (a[i] <= pivot) {
            swap(&a[pindex], &a[i]);
            pindex++; }
    }
    swap(&a[pindex], &a[r]);
    return pindex;
}
void quicksort (int a[], int l, int r){
    if (l < r) {
        int p = partition (a, l, r);
        quicksort (a, l, p-1);
        quicksort (a, p+1, r); }
}
int main() {
```

```
int n, a[100];
printf("Enter no of elements :- ");
scanf("%d", &n);
printf("Enter elements");
for(int i=0; i<n; i++){
    scanf("%d", &a[i]); }
double q = 0.0;
clock_t begin = clock();
quicksort(a, 0, n-1);
clock_t end = clock();
q = ((double) end - begin) / CLOCKS_PER_SEC;
printf("Elements :- ");
for(int i = 0; i<n; i++){
    printf("%d ", a[i]);
}
printf("Time taken :- %f", q);
return 0;
}
```

24/6/21

Page No: 27
Date: / /

Q → Floyd algorithm

```
# include <stdio.h>
# include <limits.h>
# define Max 10000
int n;
void display ( int d[][n]) {
    for (int i = 0 ; i < n ; i++ ) {
        for (int j = 0 ; j < n ; j++ ) {
            if ( d[i][j] == Max)
                printf (" %6s ", "INF");
            else
                printf (" %6d ", d[i][j]);
        } printf ("\n"); }
}
void floyd ( int g[][n] ) {
    int d[n][n], i, j, k;
    for ( i = 0; i < n ; i++ ) {
        for ( j = 0; j < n ; j++ ) {
            if ( graph[i][j] == -1 )
                d[i][j] = Max;
            else { d[i][j] = g[i][j]; }}}
    printf (" Matrix D(0) : \n");
    display ( d );
    for ( k = 0; k < n ; k++ ) {
        for ( i = 0; i < n ; i++ ) {
```

```
        for(j=0; j<n; j++){
            if (d[i][j] > d[i][k] + d[k][j])
                d[i][j] = d[i][k] + d[k][j];
        } }
        printf("\n D(%d) Matrix : \n", k+1);
        display (d); }
}
int main () {
    printf(" No of vertices :- ");
    scanf("%d", &n);
    int D[n][n];
    printf(" Enter matrix (-1 for no
            direct path) :- \n");
    for(int i=0; i<n; i++)
        for(int j=0; j<n; j++)
            scanf("%d", &D[i][j]);
    floyd (D);
    return 0;
}
```

d=> Warshall algorithm

```
#include <stdio.h>
const int Max = 100;
void Warshall (int graph[Max][Max], int n)
{  int i, j, k;
   for (k=0 ; k<n ; k++){
      for (i=0; i<n; i++){
         for (j=0; j<n; j++){
            if (graph[i][j] || (graph[i][k] &&
                  graph[k][j]))
               graph[i][j] = 1; } } }
}
int main(){
   int i, j, n;
   int g[Max][Max];
   printf(" No. of vertices :: ");
   scanf("%d", &n);
   printf(" Enter adjacency matrix ");
      for (i=0; i<n; i++)
         for (j=0; j<n; j++)
            scanf("%d", &g[i][j]);
   Warshall (g, n);
   printf(" Transitive Closure :: ");
   for (i=0; i<n; i++) {
      for (j=0; j<n; j++) {
```

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

Q-> Knapsack algorithm

```
#include <stdio.h>
int max(int a, int b) {
    return (a > b) ? a : b;
}
int Knapsack(int W, int wt[], int
                    val[], int n) {
    int i, w; K[n+1][W+1];
    for(i = 0; i <= n; i++) {
        for(w = 0; w <= W; w++) {
            if(i == 0 || w == 0)
                K[i][w] = 0;
            else if(wt[i-1] <= w)
                K[i][w] = max(K[i-1][w],
                (val[i-1] + K[i-1][w-wt[i-1]]));
            else
                K[i][w] = K[i-1][w];
        }
    }
    return K[n][W];
}
int main() {
    int n, max;
    printf("Enter no. of Items : ");
    scanf("%d", &n);
    int val[n], wt[n];
```

```
printf(" Enter wt & profit :- ");
for(int i=0; i<n; i++){
    printf(" wt & Value of %d : ", i+1);
    scanf(" %d %d ", &wt[i], &val[i]);
}
printf(" Enter capacity :- ");
scanf(" %d ", &max);
printf(" Max profit :- %d ",
            KnapSack(max, wt, val, n));
return 0;
}
```

1/7/21

Q → Prim's algorithm

```
#include <stdio.h>
int a, b, u, t, m, n, i, j;
int v[10], c = 1, min = 0; cost[10][10];
int main() {
    printf("Enter no. of vertices : ");
    scanf("%d", &n);
    printf("Enter matrix : ");
    for(i = 1; i <= n; i++) {
        for(j = 1; j <= n; j++) {
            scanf("%d", &cost[i][j]);
            if(cost[i][j] == 0)
                cost[i][j] = 999;
    } }
    v[1] = 1;
    while(c < n) {
        for(i = 1, m = 999; i <= n; i++) {
            for(j = 1; j <= n; j++) {
                if(cost[i][j] < m) { if(v[i] != 0)
                    {m = cost[i][j];
                    a = u = i;
                    b = t = j; } }
    } }
    if(v[u] == 0 || v[t] == 0) {
        printf("Edge (%d %d) : (%d %d %d)
            cost:(%d)", c++, a, b, m);
```

```
        min += cost[a][b];
        v[b] = 1;
    }
    cost[a][b] = cost[b][a] = 999;
}
printf("Min cost = %d", min);
return 0;
}
```

Q-) Kruskal algorithm

```
#include <stdio.h>
int i, j, k, a, b, u, v, n, ne = 1;
int min, mincost = 0, cost[10][10],
parent[10];
int main(){
    printf("Enter vertices :- ");
    scanf("%d", &n);
    printf("Enter adjacency matrix: ");
    for(i=1; i<=n; i++){
        for(j=1; j<=n; j++){
            scanf("%d", &cost[i][j]);
            if(cost[i][j]==0)
                cost[i][j]=999;
    }}
    while(ne<n){
        for(i=1, min=999; i<=n; i++){
            for(j=1; j<=n; j++){
                if(cost[i][j]<min){
                    min=cost[i][j];
                    a=u=i;
                    b=v=j;}}}
        while(parent[u])
            u=parent[u];
        while(parent[v])
            v=parent[v];
```

```
if (u != v) {
    printf ("Edge (%d) : (%d %d)
    Cost : (%d)", ne++, a, b, min);
    mincost += min;
    parent[v] = u; }
cost[a][b] = cost[b][a] = 999;
}
printf ("Minimum Cost = %d", mincost);
return 0;
}
```