

AISHWARYA.M

1BM20CS401

CSE-4A

Program 1

- a. To solve Tower of Hanoi problem
- b. To solve GCD

Tower of Hanoi

```
#include<stdio.h>
void tower(int n,int source,int temp,int dest)
{
    if(n==1)
    {
        printf("move disc %d from %c to %c\n",n,source,dest);
        return;
    }
    tower(n-1,source,dest,temp);
    printf("move disc %d from %c to %c\n",n,source,dest);
    tower(n-1,temp,source,dest);
}
void main()
{
    int n;
    printf("enter n\n");
    scanf("%d",&n);
```

```
    tower(n,'s','t','d');  
}
```

Output:

```
enter n  
5  
move disc 1 from s to d  
move disc 2 from s to t  
move disc 1 from d to t  
move disc 3 from s to d  
move disc 1 from t to s  
move disc 2 from t to d  
move disc 1 from s to d  
move disc 4 from s to t  
move disc 1 from d to t  
move disc 2 from d to s  
move disc 1 from t to s  
move disc 3 from d to t  
move disc 1 from s to d  
move disc 2 from s to t  
move disc 1 from d to t  
move disc 5 from s to d  
move disc 1 from t to s  
move disc 2 from t to d  
move disc 1 from s to d  
move disc 3 from t to s  
move disc 1 from d to t  
move disc 2 from d to s  
move disc 1 from t to s  
move disc 4 from t to d  
move disc 1 from s to d  
move disc 2 from s to t  
move disc 1 from d to t  
move disc 3 from s to d  
move disc 1 from t to s  
move disc 2 from t to d  
move disc 1 from s to d  
...Program finished with exit code 0
```

GCD

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

```
#include<math.h>
```

```
int gcd(int m,int n)
```

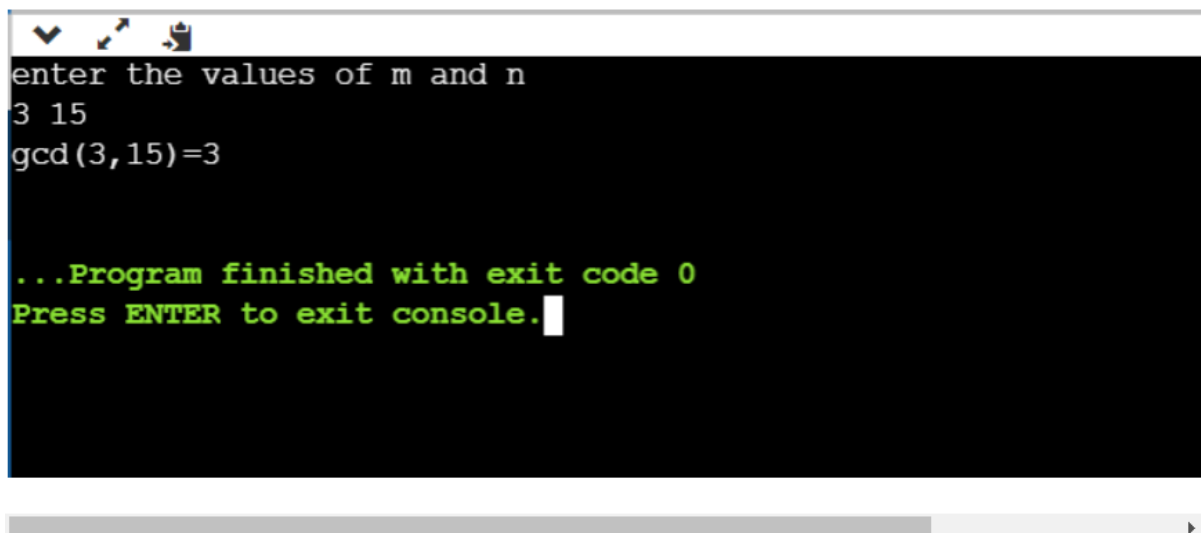
```
{
```

```
if(n==0) return m;
```

```
if(m<n) return gcd(n,m);
```

```
return gcd(n,m%n);  
}  
  
void main()  
{  
    int res ,m,n;  
    printf("enter the values of m and n\n");  
    scanf("%d %d",&m,&n);  
    res = gcd(m,n);  
    printf("gcd(%d,%d)=%d\n",m,n,res);  
}
```

Output:



```
enter the values of m and n  
3 15  
gcd(3,15)=3  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

Program 2 :

Implement Recursive Binary search and Linear search and determine the time required to search an element. Repeat the experiment for different values of N and plot a graph of the time taken versus N

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

```
#include <stdlib.h>
```

```
#include <time.h>
```

```
int binarySearch(int arr[], int l, int r, int x)
```

```
{
```

```
    if (r >= l)
```

```
    {
```

```
        int mid = l + (r - l) / 2;
```

```
        if (arr[mid] == x)
```

```
        {
```

```
            return mid;
```

```
        }
```

```
        if (arr[mid] > x)
```

```
            return binarySearch(arr, l, mid - 1, x);
```

```
        return binarySearch(arr, mid + 1, r, x);
```

```
}
```

```

return -1;
}
int linearSearch(int arr[], int l,int n, int x)
{
    if (l>=n)
        return -1;
    if (arr[l] == x)
    {
        return l;
    }
    return linearSearch(arr, l+1,n, x);
}
int main()
{
    int x, j = 0,flag, n = 10000, sort = 0, i, ch;
    clock_t start, end;
    int arr[n];
    for (i = 0; i < n; i++)
    {
        int no = rand() % n + 1;
        arr[i] = no;
    }
}

```

```

}
for(int m = 0; m<n; m++)
{
    printf(" %d ",arr[m]);
}
printf("\n\n");
for(;;)
{
    printf("1.Linear Search \n2.Binary Search \n");
    scanf("%d",&ch);
    switch(ch)
    {
    case 1:
        printf("Enter the element to be Searched : ");
        scanf("%d", &x);
        start = clock();
        int y = linearSearch(arr, 0, n, x);
        end = clock();
        float ti = ((double)(end - start)/CLOCKS_PER_SEC);
        // printf("Element found at index :%d \n",y);
        if(y != 1)

```

```

{
    printf("Element Found at position %d.\n", y + 1);
}
else
{
    printf("Element not found.\n");
}
printf("Time taken: %f\n\n", ti);
break;
case 2:
    j = 0;
    flag = 0;
    int temp;
    while(j < n-1 && sort == 0 )
    {
        if(arr[j] > arr[j+1])
        {
            flag=1;
        }
        j++;
    }

```

```
if(flag == 1)
{
    for (int i = 0; i < n-1; i++)
    {
        for (j = 0; j < n-i-1; j++)
        {
            if (arr[j] > arr[j+1])
            {
                temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }
    for(int i=0; i<n; i++)
    {
        printf("%d ",arr[i]);
    }
    sort = 1;
    printf("\n\n");
}
```



```
printf("Enter the element to be Searched : ");

scanf("%d",&x);

start = clock();

int result = binarySearch(arr, 0, n - 1, x);

end = clock();

if(result == -1)
{
    printf("Element Not Found.\n");
}
else
{
    printf("Element found at position %d.\n", result + 1);
}

float tm = ((double)(end - start)/CLOCKS_PER_SEC);

printf("Time taken: %f\n\n",tm);


break;

default:

    exit(0);

}

}
```

```
    return 0;  
}
```

Output :

```
1.Linear Search  
2.Binary Search  
2  
Enter the element to be Searched : 10000  
Element found at position 9999.  
Time taken: 0.000004
```

```
1.Linear Search  
2.Binary Search  
█
```

```
1.Linear Search  
2.Binary Search  
1  
Enter the element to be Searched : 9431  
Element Found at position 3594.  
Time taken: 0.000184
```

Program 3 :

Sort a given set of N integer elements using Selection Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.

```
#include <stdio.h>

#include <time.h>

#include <stdlib.h>

int minIndex(int arr[], int i, int j)
{
    if (i == j)
        return i;

    int k = minIndex(arr, i + 1, j);
    return (arr[i] < arr[k])? i : k;
}

void recurSelectionSort(int arr[], int n, int index) {
    int temp;

    if (index == n)
        return;

    int k = minIndex(arr, index, n-1);

        if (k != index)
    {
```

```
temp = arr[k];
arr[k] = arr[index];
arr[index] = temp;
}
recurSelectionSort(arr, n, index + 1);
}

int main()
{
int size,i,n;
clock_t start, end;
printf("Enter the size of the list: ");
scanf("%d", &n);
int arr[n];
for (i = 0; i < n; i++)
{
int no = rand() % n + 1;
arr[i] = no;
}
start = clock();
recurSelectionSort(arr, n,0);
end = clock();
```

```
float ti = ((double)(end - start)/CLOCKS_PER_SEC);  
for (int i = 0; i<n ; i++)  
    printf("%d ",arr[i]);  
printf("\nTime taken: %f", ti);  
return 0;  
}
```

Output:

```
Enter the size of the list: 100  
3 4 6 6 9 12 12 13 14 14 15 16 16 20 22 22 23 24 25 25 26 27 27  
1 52 55 57 57 58 59 60 61 63 63 63 64 65 68 68 68 68 69 69 70 7  
94 94 95 96 97 99 100  
Time taken: 0.000031  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

Program 4 :

Write program to do the following:

a. Print all the nodes reachable from a given starting node in a digraph using BFS method.

b. Check whether a given graph is connected or not using DFS method

Print all the nodes reachable from a given starting node in a digraph using BFS method.

```
#include<stdio.h>

#include <time.h>

int a[20][20],q[20],visited[20],n,i,j,f=0,r=-1;

void bfs(int v)
{
    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++]);
    }
}
```

```

    }
}

void main()
{
    int v;

    clock_t start, end;

    printf("\n Enter the number of vertices:");

    scanf("%d",&n);

    for (i=1;i<=n;i++)
    {
        q[i]=0;

        visited[i]=0;

    }

    printf("\nEnter the adjacency matrix form:\n");

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

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

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

    printf("\n Enter the starting vertex:");

    scanf("%d",&v);

    start = clock();

    bfs(v);

```

```

end = clock();

float time_taken = ((double)(end - start)/CLOCKS_PER_SEC);

    for (i=1;i<=n;i++)
    {
        if(visited[i])

            printf("the node %d is reachable\n",i);

        else

            printf("The node %d is not reachable\n",i);

    }

    printf("\nTime taken: %f", time_taken);
}

```

Output:

```

Enter the number of vertices:4
Enter the adjacency matrix form:
0 1 0 1
1 1 0 1
1 0 1 0
1 1 1 0

Enter the starting vertex:2
the node 1 is reachable
the node 2 is reachable
the node 3 is reachable
the node 4 is reachable

Time taken: 0.000003

...Program finished with exit code 0
Press ENTER to exit console.

```


Check whether a given graph is connected or not using DFS method

```
#include<stdio.h>

#include <time.h>

int a[20][20], reach[20],n;

void dfs(int v)
{
    int i;

    reach[v]=1;

    for(i=1;i<=n;i++)
        if(a[v][i] && !reach[i])
        {
            printf("\n%d->%d",v,i);

            dfs(i);
        }
}

int main()
{
    int i,j,count=0;

    clock_t start, end;

    start = clock();

    printf("\nEnter number of vertices: ");
```

```
scanf("%d",&n);  
for(i=1;i<=n;i++)  
{  
    reach[i]=0;  
    for(j=1;j<=n;j++)  
        a[i][j]=0;  
}  
printf("Enter the adjacency matrix:\n");  
for(i=1;i<=n;i++)  
for(j=1;j<=n;j++)  
scanf("%d",&a[i][j]);  
dfs(1);  
printf("\n");  
for(i=1;i<=n;i++)  
{  
    if(reach[i])  
        count++;  
}  
if(count==n)  
printf("Graph is connected\n");  
else
```

```
printf("Graph is not connected\n");

end = clock();

float time_taken = ((double)(end - start)/CLOCKS_PER_SEC);

printf("\nTime taken: %f", time_taken);

return 0;

}
```

Output:

```
Enter number of vertices: 4
Enter the adjacency matrix:
0 1 1 0
1 1 0 1
1 1 1 0
0 1 1 0

1->2
2->4
4->3
Graph is connected

Time taken: 0.000293

...Program finished with exit code 0
Press ENTER to exit console.□
```

Program 5 :

Sort a given set of N integer elements using Insertion Sort technique and compute its time taken.

```
#include<stdio.h>

#include<stdlib.h>

#include <time.h>

void recursiveInsertionSort(int arr[], int n){

    if (n <= 1)

        return;

    recursiveInsertionSort( arr, n-1 );

    int nth = arr[n-1];

    int j = n-2;

    while (j >= 0 && arr[j] > nth){

        arr[j+1] = arr[j];

        j--;

    }

    arr[j+1] = nth;

}

int main(){

    int size,i,n;

    clock_t start, end;

    printf("Enter the size of the list: ");
```

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

int arr[n];

for (i = 0; i < n; i++)
{
    int no = rand() % n + 1;

    arr[i] = no;
}

printf("Unsorted Array:\t");

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

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

start = clock();

recursiveInsertionSort(arr, n);

end = clock();

float ti = ((double)(end - start)/CLOCKS_PER_SEC);

printf("\nSorted Array:\t");

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

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

printf("\nTime taken: %f", ti);

return 0;

}
```

Output:

```
Enter the size of the list: 30
Unsorted Array: 14 17 28 26 24 26 17 13 10 2 3 8 21 20 24 17 1
Sorted Array:   1 2 3 3 3 7 8 8 9 10 10 12 13 14 14 17 17 17 17
Time taken: 0.000004

...Program finished with exit code 0
Press ENTER to exit console.
```

Program 6 :

Write program to obtain the Topological ordering of vertices in a given digraph.

```
#include<stdio.h>

#include <time.h>

int a[20][20],visited[20],n,stack[20],top=-1;

void dfs_helper(int v)
{
    int i;
    visited[v]=1;
    for(i=1;i<=n;i++)
    {
        if(a[v][i] && !visited[i])
        {
            dfs_helper(i);
        }
    }
    stack[++top]=v;
}

void dfs()
{
    for(int i=1;i<=n;i++)
```

```

    {
        if(!visited[i])
        {
            dfs_helper(i);
        }
    }
    while(top>=0)
    {
        printf("%d ",stack[top--]);
    }
}

void main()
{
    int i,j,count=0;
    clock_t start, end;
    double cpu_time_used;
    printf("\n Enter number of vertices:");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        visited[i]=0;
    }
}

```



```

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

a[i][j]=0;

}

printf("\n Enter the adjacency matrix:\n");

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

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

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

start = clock();

dfs();

end = clock();

printf("\n");

cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;

printf("TIME FOR FUNCTION EXECUTION is %f\n", cpu_time_used);

}

```

Output:

```

Enter number of vertices:5

Enter the adjacency matrix:
0 1 0 0 1
1 0 1 0 1
0 1 0 1 0
1 1 1 0 0
0 1 1 1 0
1 2 5 3 4
TIME FOR FUNCTION EXECUTION is 0.000007

...Program finished with exit code 0
Press ENTER to exit console.

```

Program 7 :

Implement Johnson Trotter algorithm to generate permutations

```
#include <stdlib.h>
```

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

```
int lr=1;
```

```
int rl=0;
```

```
int fact(int n)
```

```
{
```

```
    int res = 1;
```

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

```
        res = res * i;
```

```
    return res;
```

```
}
```

```
int mobile_pos(int a[],int n,int mobile)
```

```
{
```

```
    for(int i=0;i<n;i++)
```

```
    {
```

```
        if(a[i]==mobile)
```

```
        {
```

```
            return i;
```

```
        }
```

```

    }
}

int largest_mobile(int a[],int dir[],int n)
{
    int mobile=0;
    for(int i=0;i<n;i++)
    {
        if(dir[a[i]-1]==rl && i!=0)
        {
            if(a[i]>a[i-1] && a[i]>mobile)
            {
                mobile=a[i];
            }
        }
        if(dir[a[i]-1]==lr && i!=n-1)
        {
            if(a[i]>a[i+1] && a[i]>mobile)
            {
                mobile=a[i];
            }
        }
    }
}

```

```

    }

    if (mobile == 0)
    {
        return 0;
    }
    else
    {
        return mobile;
    }
}

void perm(int a[],int dir[],int n)
{
    int mobile=largest_mobile(a,dir,n);
    int pos=mobile_pos(a,n,mobile);
    if(dir[a[pos]-1]==rl)
    {
        int temp=a[pos];
        a[pos]=a[pos-1];
        a[pos-1]=temp;
    }
    else if(dir[a[pos]-1]==lr)

```

```

{
    int temp=a[pos];
    a[pos]=a[pos+1];
    a[pos+1]=temp;
}
for(int i=0;i<n;i++)
{
    if(a[i]>mobile)
    {
        if(dir[a[i]-1]==lr)
        {
            dir[a[i]-1]=rl;
        }
        else if(dir[a[i]-1]==rl)
        {
            dir[a[i]-1]=lr;
        }
    }
}
for(int i=0;i<n;i++)
{

```

```

        printf("%d ",a[i]);
    }
    printf("\n");
}
int main()
{
    int n;
    printf("Enter the number of elements:");
    scanf("%d",&n);
    int a[n];
    int dir[n];
    for(int i=0;i<n;i++)
    {
        a[i]=i+1;
        dir[i]=0;
        printf("%d ",a[i]);
    }
    printf("\n");
    for(int i=1;i<fact(n);i++)
    {
        perm(a,dir,n);
    }
}

```

```
}  
  
return 0;  
  
}
```

Output :

```
Enter the number of elements:4  
1 2 3 4  
1 2 4 3  
1 4 2 3  
4 1 2 3  
4 1 3 2  
1 4 3 2  
1 3 4 2  
1 3 2 4  
3 1 2 4  
3 1 4 2  
3 4 1 2  
4 3 1 2  
4 3 2 1  
3 4 2 1  
3 2 4 1  
3 2 1 4  
2 3 1 4  
2 3 4 1  
2 4 3 1  
4 2 3 1  
4 2 1 3  
2 4 1 3  
2 1 4 3  
2 1 3 4  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

Program 8 :

Sort a given set of N integer elements using Merge Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.

```
#include <stdio.h>

#include <time.h>

#include <stdlib.h>

void merge(int arr[], int p, int q, int r)
{
    int n1 = q - p + 1;
    int n2 = r - q;
    int L[n1], M[n2];
    for (int i = 0; i < n1; i++)
        L[i] = arr[p + i];
    for (int j = 0; j < n2; j++)
        M[j] = arr[q + 1 + j];
    int i, j, k;
    i = 0;
    j = 0;
    k = p;
    while (i < n1 && j < n2) {
```



```

    if (L[i] <= M[j]) {
        arr[k] = L[i];
        i++;
    } else {
        arr[k] = M[j];
        j++;
    }
    k++;
}

while (i < n1) {
    arr[k] = L[i];
    i++;
    k++;
}

while (j < n2) {
    arr[k] = M[j];
    j++;
    k++;
}
}

void mergeSort(int arr[], int l, int r) {

```

```

if (l < r) {
    int m = l + (r - l) / 2;
    mergeSort(arr, l, m);
    mergeSort(arr, m + 1, r);
    merge(arr, l, m, r);
}
}

void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

int main()
{
    int n;

    clock_t start, end;
    double cpu_time_used;

    printf("Enter the size of the array\n");
    scanf("%d",&n);

    int arr[n];

```

```

printf("The elements of the array:\n");

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

arr[i]= rand()%1000;

printArray(arr, n);

printf("\n");

start = clock();

mergeSort(arr, 0, n-1);

end = clock();

cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;

printf("Sorted array: \n");

printArray(arr, n);

printf("\n");

printf("TIME FOR FUNCTION EXECUTION is %f", cpu_time_used);

return 0;

}

```

Output:

```

Enter the size of the array
50
The elements of the array:
383 886 777 915 793 335 386 492 649 421 362 27 690 59 763 926 540 426 172 736 211 368 567 429 782 530 862 123 67 135 929 802 22 58 69 167
393 456 11 42 229 373 421 919 784 537 198 324 315 370

Sorted array:
11 22 27 42 58 59 67 69 123 135 167 172 198 211 229 315 324 335 362 368 370 373 383 386 393 421 421 426 429 456 492 530 537 540 567 649 6
90 736 763 777 782 784 793 802 862 886 915 919 926 929

TIME FOR FUNCTION EXECUTION is 0.000011

...Program finished with exit code 0
Press ENTER to exit console.

```

Program 9 :

Sort a given set of N integer elements using Quick Sort technique and compute its time taken

```
#include <stdio.h>

#include <time.h>

#include <stdlib.h>

int partition (int arr[], int low, int high)
{
    int pivot = arr[high];
    int i = (low - 1);
    for (int j = low; j <= high - 1; j++)
    {
        if (arr[j] < pivot)
        {
            i++;

            int temp=arr[i];
            arr[i]=arr[j];
            arr[j]=temp;
        }
    }

    int temp=arr[i+1];
    arr[i+1]=arr[high];
```

```

    arr[high]=temp;
    return (i + 1);
}

void quickSort(int arr[], int low, int high)
{
    if (low < high)
    {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

int main()
{
    int n;

    clock_t start, end;

```

```
double cpu_time_used;

    printf("Enter the size of the array\n");

    scanf("%d",&n);

int arr[n];

printf("The elements of the array:\n");

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

arr[i]=rand()%1000;

printArray(arr, n);

printf("\n");

start = clock();

quickSort(arr, 0, n - 1);

end = clock();

cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;

printf("Sorted array: \n");

printArray(arr, n);

printf("\n");

printf("TIME FOR FUNCTION EXECUTION is %f", cpu_time_used);

return 0;

}
```

Output :

```
Enter the size of the array
50
The elements of the array:
383 886 777 915 793 335 386 492 649 421 362 27 690 59 763 926 540 426 172 736 211 368 567 429 782 530 862 123 67 135 929 802 22 58 69 167
393 456 11 42 229 373 421 919 784 537 198 324 315 370

Sorted array:
11 22 27 42 58 59 67 69 123 135 167 172 198 211 229 315 324 335 362 368 370 373 383 386 393 421 421 426 429 456 492 530 537 540 567 649 6
90 736 763 777 782 784 793 802 862 886 915 919 926 929

TIME FOR FUNCTION EXECUTION is 0.000006

...Program finished with exit code 0
Press ENTER to exit console.
```