

BHAVANA CK

1BM0CS403

CSE-4A

PROGRAM 1:

Write a recursive program to

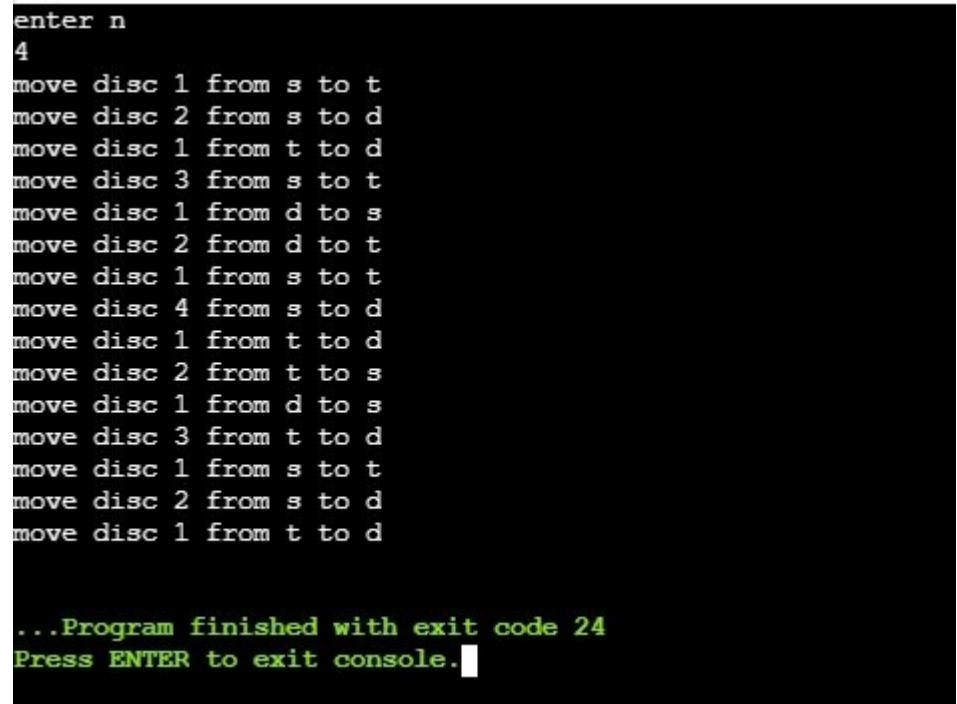
- a. Solve Towers-of-Hanoi problem
- b. To find GCD

```
// 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
4
move disc 1 from s to t
move disc 2 from s to d
move disc 1 from t to d
move disc 3 from s to t
move disc 1 from d to s
move disc 2 from d to t
move disc 1 from s to t
move disc 4 from s to d
move disc 1 from t to d
move disc 2 from t to s
move disc 1 from d to s
move disc 3 from t to d
move disc 1 from s to t
move disc 2 from s to d
move disc 1 from t to d

...Program finished with exit code 24
Press ENTER to exit console.
```

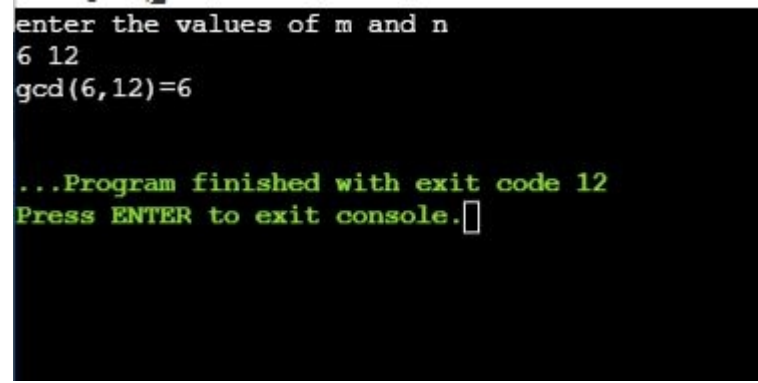
//GCD

```
#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  
6 12  
gcd(6,12)=6  
  
...Program finished with exit code 12  
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 <time.h>
#include <stdlib.h>
```

```
int Binary_search(int arr[], int l,int r,int e)
{
    int p;
    if(r<l)
        return -1;
    p=(l+r)/2;
    if(arr[p]==e)
        return p;
    if(arr[p]<e)
        return Binary_search(arr,p+1,r,e);
    if(arr[p]>e)
        return Binary_search(arr,l,r-1,e);
}
```

```
void Sort(int arr[], int n)
{
```

```

int i,j,a;
for (i = 0; i < n; ++i)
{

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

        if (arr[i] > arr[j])
        {

            a = arr[i];
            arr[i] = arr[j];
            arr[j] = a;

        }

    }

}

int Linear_search(int arr[], int l,int r,int e)
{
    if(l>r)
        return -1;
    if(arr[l]==e)
        return l;
    return Linear_search(arr,l+1,r,e);
}

void printArray(int arr[], int size)
{

```

```

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

```

```

int main(void)
{
    int choice;
    int n,ele,f;
    clock_t start, end;
    double cpu_time_used;
    printf("1.Linear Search\n2.Binary
Search\n3.Exit\n");
    scanf("%d",&choice);
    while(choice<=2)
    {
        switch(choice)
        {
            case 1:
            {
                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();
                printArray(arr, n);
                printf("Enter the element to be
searched\n");

```

```

scanf("%d",&ele);
start = clock();
f=Linear_search(arr,0, n-1,ele);
end = clock();
cpu_time_used = ((double) (end - start)) /
CLOCKS_PER_SEC;
if(f==-1)
printf("Search unsuccessful.\n");
else
printf("Search successful. Element found
at index %d\n",f);
printf("TIME FOR FUNCTION EXECUTION
is %f\n", cpu_time_used);
break;
}
case 2:
{
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();
Sort(arr, n);
printf("Sorted array: \n");
printArray(arr, n);
printf("Enter the element to be
searched\n");
scanf("%d",&ele);
start = clock();
f=Binary_search(arr,0, n-1,ele);

```

```

        end = clock();
        cpu_time_used = ((double) (end - start)) /
CLOCKS_PER_SEC;
        if(f==-1)
        printf("Search unsuccessful.\n");
        else
        printf("Search successful. Element found
at index %d\n",f);
        printf("TIME FOR FUNCTION EXECUTION
is %f\n", cpu_time_used);
        break;
    }
}
printf("1.Linear Search\n2.Binary
Search\n3.Exit\n");
scanf("%d",&choice);
}
return 0;
}

```

OUTPUT:

```

1.Linear Search
2.Binary Search
3.Exit
1
Enter the size of the array
50
The elements of the array:
1804289383 846930886 1681692777 1714636915 1957747793 424238335 719885386 1649760492 596
690 1102520059 2044897763 1967513926 1365180540 1540383426 304089172 1303455736 35005211
21530 278722862 233665123 2145174067 468703135 1101513929 1801979802 1315634022 63572305
628175011 1656478042 1131176229 1653377373 859484421 1914544919 608413784 756898537 1734
Enter the element to be searched
44
Search unsuccessful.
TIME FOR FUNCTION EXECUTION is 0.000003
1.Linear Search
2.Binary Search
3.Exit
2
Enter the size of the array
50
The elements of the array:
Sorted array:
42999170 84353895 135497281 137806862 184803526 356426808 412776091 491705403 511702305
754 760313750 805750846 855636226 939819582 943947739 945117276 982906996 1100661313 112
44043 1411549676 1424268980 1433925857 1469348094 1474612399 1477171087 1548233367 15859
327 1843993368 1889947178 1911759956 1918502651 1937477084 1956297539 1984210012 1998898
Enter the element to be searched
23
Search unsuccessful.
TIME FOR FUNCTION EXECUTION is 0.000003

```


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

    int k = minIndex(a, i + 1, j);

    return (a[i] < a[k])? i : k;
}

void selectionSort(int a[], int n, int index)
{
    if (index == n)
        return;

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

```

        if (k != index)
        {
            int temp=a[k];
            a[k]=a[index];
            a[index]=temp;
        }

        selectionSort(a, n, index + 1);
    }

```

```

void printArray(int arr[], int size)
{
    int i;
    for (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();
    selectionSort(arr, n, 0);
    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
100
The elements of the array:
383 886 777 915 793 335 386 492 649 421 362 27 690 59 763 926 540 426 172 736 211 368 567 429 781
167 393 456 11 42 229 373 421 919 784 537 198 324 315 370 413 526 91 980 956 873 862 170 996 281
7 124 895 582 545 814 367 434 364 43 750 87 808 276 178 788 584 403 651 754 399 932 60 676 368 71
Sorted array:
11 12 22 27 42 43 58 59 60 67 69 84 87 91 94 123 124 135 167 170 172 178 198 211 226 229 276 281
67 368 368 370 373 383 386 393 399 403 413 421 421 426 429 434 456 492 505 526 530 537 539 540 540
736 739 750 754 763 777 782 784 788 793 802 808 814 846 857 862 862 873 886 895 915 919 925 926
TIME FOR FUNCTION EXECUTION is 0.000056
...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

```
// BFS
```

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

```
#include<conio.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;
```

```
double cpu_time_used;
```

```

printf("\n Enter the number of vertices:");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
q[i]=0;
visited[i]=0;
}
printf("\n Enter graph data in 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();
cpu_time_used = ((double) (end - start)) /
CLOCKS_PER_SEC;
printf("\n The node which are reachable are:\n");
for(i=1;i<=n;i++)
if(visited[i])
printf("%d\t",i);
printf("TIME FOR FUNCTION EXECUTION is %f\n",
cpu_time_used);
getch();
}

```

OUTPUT:

```

Enter the number of vertices:4

Enter graph data in matrix form:

0 1 0 0
1 0 1 1
1 1 0 1
1 1 1 0

Enter the starting vertex:1

The node which are reachable are:
1      2      3      4
TIME FOR FUNCTION EXECUTION is 0.000003

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

```

// DFS

```

#include<stdio.h>
#include<conio.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);
    }
}
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++)
{
    reach[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(1);
end = clock();
printf("\n");
for(i=1;i<=n;i++)
{
    if(reach[i])
        count++;
}
if(count==n)
    printf("\n Graph is connected\n");
else
    printf("\n Graph is not connected\n");
cpu_time_used = ((double) (end - start)) /
CLOCKS_PER_SEC;

```

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

OUTPUT:

```
Enter number of vertices:5  
  
Enter the adjacency matrix:  
0 1 1 0 1  
1 0 1 1 0  
1 1 0 1 1  
0 1 1 0 1  
1 0 1 1 0  
  
1->2  
2->3  
3->4  
4->5  
  
Graph is connected  
TIME FOR FUNCTION EXECUTION is 0.000040  
  
...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 <time.h>
#include <stdlib.h>

void insertionSort(int arr[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;

        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}

void printArray(int arr[], int size)
{
    int i;
```

```
        for (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();
    insertionSort(arr, n);
    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
200
The elements of the array:
383 886 777 915 793 335 386 492 649 421 362 27 690 59 763 926 540 426
167 393 456 11 42 229 373 421 919 784 537 198 324 315 370 413 526 91
7 124 895 582 545 814 367 434 364 43 750 87 808 276 178 788 584 403 6
67 601 97 902 317 492 652 756 301 280 286 441 865 689 444 619 440 729
9 624 528 871 732 829 503 19 270 368 708 715 340 149 796 723 618 245
7 856 743 491 227 365 859 936 432 551 437 228 275 407 474 121 858 395

Sorted array:
11 11 12 19 22 27 29 31 34 42 43 58 59 60 67 69 84 87 91 94 97 97 117
227 228 228 229 235 237 245 270 275 276 280 281 286 301 305 306 313 3
0 373 378 379 383 386 393 395 399 403 407 413 421 421 426 428 429 432
500 503 505 526 528 529 530 537 539 540 545 551 555 567 567 570 582 5
9 715 723 729 729 732 736 739 743 750 754 756 763 764 764 771 777 782
857 858 859 862 862 865 871 873 886 895 902 914 915 919 921 925 926 9

TIME FOR FUNCTION EXECUTION is 0.000048

...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<conio.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);  
getch();  
}  
}
```

OUTPUT:

```
Enter number of vertices:5  
  
Enter the adjacency matrix:  
0  1 0 1 0  
1 0 1 1 1  
1 1 0 1 1  
1 0 0 0 1  
0 1 0 1 0  
1 2 3 4 5  
TIME FOR FUNCTION EXECUTION is 0.000029  
  
...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:3
1 2 3
1 3 2
3 1 2
3 2 1
2 3 1
2 1 3

...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
200
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 2
167 393 456 11 42 229 373 421 919 784 537 198 324 315 370 413 526 91 980 956 87
7 124 895 582 545 814 367 434 364 43 750 87 808 276 178 788 584 403 651 754 399
67 601 97 902 317 492 652 756 301 280 286 441 865 689 444 619 440 729 31 117 97
9 624 528 871 732 829 503 19 270 368 708 715 340 149 796 723 618 245 846 451 92
7 856 743 491 227 365 859 936 432 551 437 228 275 407 474 121 858 395 29 237 23
Sorted array:
11 11 12 19 22 27 29 31 34 42 43 58 59 60 67 69 84 87 91 94 97 97 117 121 123 1
227 228 228 229 235 237 245 270 275 276 280 281 286 301 305 306 313 315 317 324
0 373 378 379 383 386 393 395 399 403 407 413 421 421 426 428 429 432 434 434 4
500 503 505 526 528 529 530 537 539 540 545 551 555 567 567 570 582 584 586 586
9 715 723 729 729 732 736 739 743 750 754 756 763 764 764 771 777 782 784 788 7
857 858 859 862 862 865 871 873 886 895 902 914 915 919 921 925 926 927 928 929
TIME FOR FUNCTION EXECUTION is 0.000040
...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
150
The elements of the array:
383 886 777 915 793 335 386 492 649 421 362 27 690 59 763 926 540 426
167 393 456 11 42 229 373 421 919 784 537 198 324 315 370 413 526 91 9
7 124 895 582 545 814 367 434 364 43 750 87 808 276 178 788 584 403 65
67 601 97 902 317 492 652 756 301 280 286 441 865 689 444 619 440 729
9 624 528 871 732 829 503 19 270 368 708 715

Sorted array:
11 12 19 22 27 31 42 43 58 59 60 67 69 84 87 91 94 97 97 117 123 124 1
05 306 313 315 317 324 327 335 336 353 362 364 367 368 368 368 370 373
456 467 481 492 492 497 503 505 526 528 530 537 539 540 545 567 567 5
08 709 715 729 729 732 736 739 750 754 756 763 771 777 782 784 788 793
915 919 925 926 927 929 932 956 965 980 996

TIME FOR FUNCTION EXECUTION is 0.000019

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

```