

LAB EXERCISE 1: TOWER OF HANOI

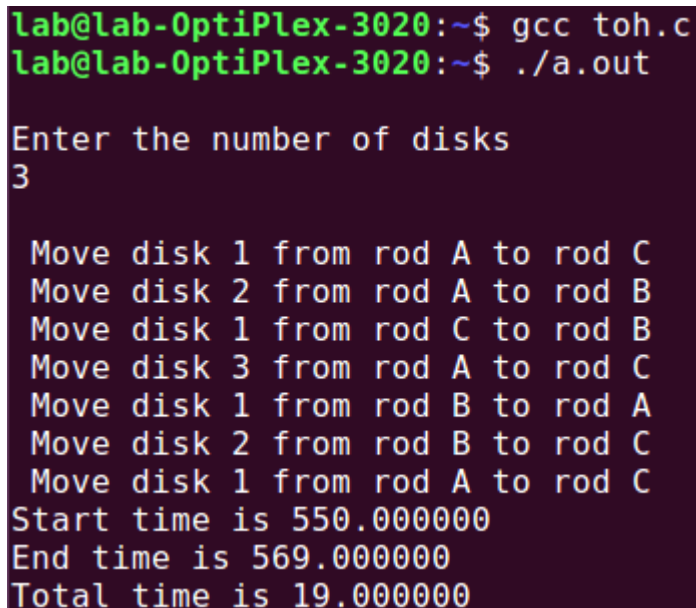
Design a C program to solve the Tower of Hanoi. Compute the time complexity.

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

void towerOfHanoi (int n, char from_rod, char to_rod, char aux_rod)
{
    if (n == 1)
    {
        printf ("\n Move disk 1 from rod %c to rod %c", from_rod, to_rod);
        return;
    }
    towerOfHanoi (n-1, from_rod, aux_rod, to_rod);
    printf ("\n Move disk %d from rod %c to rod %c", n, from_rod, to_rod);
    towerOfHanoi (n-1, aux_rod, to_rod, from_rod);
}

int main ()
{
    int n;
    printf("\nEnter the number of disks\n");
    scanf("%d",&n);
    clock_t start=clock ();
    towerOfHanoi (n, 'A','C', 'B');
    clock_t end=clock();
    printf ("\nStart time is %lf\n",(double)start);
    printf ("End time is %lf\n",(double)end);
    printf ("Total time is %lf\n",(double)(end-start));
    return 0;
}
```

OUTPUT:



```
lab@lab-OptiPlex-3020:~$ gcc toh.c
lab@lab-OptiPlex-3020:~$ ./a.out

Enter the number of disks
3

Move disk 1 from rod A to rod C
Move disk 2 from rod A to rod B
Move disk 1 from rod C to rod B
Move disk 3 from rod A to rod C
Move disk 1 from rod B to rod A
Move disk 2 from rod B to rod C
Move disk 1 from rod A to rod C
Start time is 550.000000
End time is 569.000000
Total time is 19.000000
```

LAB EXERCISE 2: BINARYSEARCH

Apply divide and conquer method and Design a C program to search an element in a given array and Compute the time complexity.

Binary search - recursive method.

```
#include <stdio.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 main()
{
    int n,x;
    printf("Enter size\n");
    scanf("%d",&n);
    int arr[n];
    printf("Enter array elements\n");
    for(int i=0;i<n;i++)
    {
        scanf("%d",&arr[i]);
    }
    printf("Enter key\n");
    scanf("%d",&x);
    clock_t start=clock();
    int result = binarySearch(arr, 0, n - 1, x);
    clock_t end=clock();
    if(result == -1)
        printf("Element is not present in array\n");
    else
        printf("Element is present at index %d\n", result);
    printf("\nStart time is %lf\n",(double)start);
    printf("\nEnd time is %lf\n",(double)end);
    printf("\nTotal time is %lf\n",(double)(end-start));
    return 0;
}
```

OUTPUT:

```
lab@lab-OptiPlex-3020:~$ gcc bs.c
lab@lab-OptiPlex-3020:~$ ./a.out
Enter size
6
Enter array elements
24 34 44 54 64 74
Enter key
44
Element is present at index 2

Start time is 613.000000

End time is 614.000000

Total time is 1.000000
lab@lab-OptiPlex-3020:~$ ./a.out
Enter size
6
Enter array elements
24 34 44 54 64 74
Enter key
88
Element is not present in array

Start time is 584.000000

End time is 584.000000

Total time is 0.000000
```

LAB EXERCISE 3: MERGESORT

Apply Divide and Conquer method Design a C program to sort an array using Merge sort algorithm and compute its time complexity.

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

```
void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;
    // Create temp arrays
    int L[n1], R[n2];
    // Copy data to temp array
```

```

        for (i = 0; i < n1; i++)
            L[i] = arr[l + i];
        for (j = 0; j < n2; j++)
            R[j] = arr[m + 1 + j];
// Merge the temp arrays
i = 0;
j = 0;
k = l;
while (i < n1 && j < n2)
{
    if (L[i] <= R[j])
    {
        arr[k] = L[i];
        i++;
    }
    else
    {
        arr[k] = R[j];
        j++;
    }
    k++;
}
// Copy the remaining elements of L[]
while (i < n1)
{
    arr[k] = L[i];
    i++;
    k++;
}
// Copy the remaining elements of R[]
while (j < n2)
{
    arr[k] = R[j];
    j++;
    k++;
}
}
}
void mergeSort(int arr[], int l, int r)
{
    if (l < r)
    {
        // Finding mid element
        int m = l+(r-l)/2;
        // Recursively sorting both the halves
        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);
        merge(arr, l, m, r);
    }
}
void printArray(int A[], int size)
{
    int i;

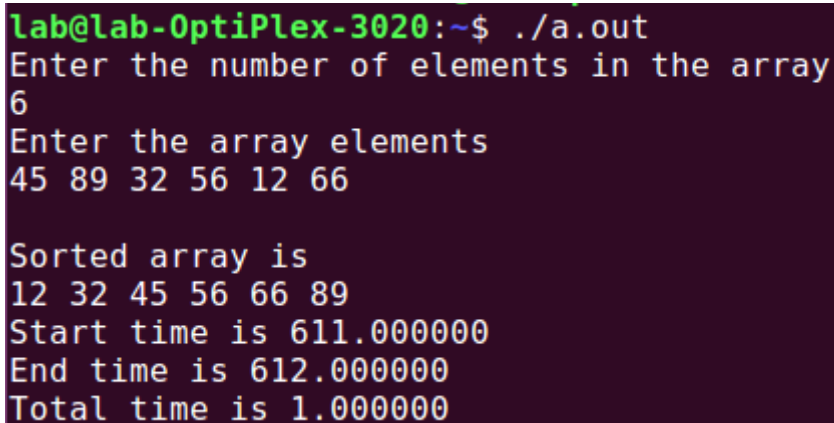
```

```

        for (i=0; i < size; i++)
            printf("%d ", A[i]);
    }
int main()
{
    int arr[25],n;
    printf("Enter the number of elements in the array\n");
    scanf("%d",&n);
    printf("Enter the array elements\n");
    for(int i=0;i<n;i++)
        scanf("%d",&arr[i]);
    clock_t start=clock();
    mergeSort(arr, 0, n- 1);
    clock_t end=clock();
    printf("\nSorted array is\n");
    printArray(arr, n);
    printf("\nStart time is %lf\n",(double)start);
    printf("End time is %lf\n",(double)end);
    printf("Total time is %lf\n",(double)(end-start));
    return 0;
}

```

OUTPUT:



```

lab@lab-OptiPlex-3020:~$ ./a.out
Enter the number of elements in the array
6
Enter the array elements
45 89 32 56 12 66

Sorted array is
12 32 45 56 66 89
Start time is 611.000000
End time is 612.000000
Total time is 1.000000

```

LAB EXCERCISE 4:QUICKSORT

Apply Divide and Conquer method Design a C program to sort an array using Quick sort algorithm and compute its time complexity.

```

#include<stdio.h>
#include<time.h>

void quicksort(int a[],int low,int high);
int partition(int a[],int low,int high);
void swap(int*,int*);

void quicksort(int a[],int low,int high)
{
    if(low<high)
    {
        int pi = partition(a,low,high);
    }
}

```

```

        quicksort(a,low,pi-1);
        quicksort(a,pi+1,high);
    }
}

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

int partition(int a[],int low,int high)
{
    int pivot=a[high];
    int i=low-1;
    for(int j=low;j<=high-1;j++)
    {
        if(a[j]<=pivot)
        {
            i++;
            swap(&a[i],&a[j]);
        }
    }
    swap(&a[i+1],&a[high]); return (i+1);
}

int main()
{
    int a[25],n;
    printf("Enter the number of elements in the array\n");
    scanf("%d",&n);
    printf("Enter the elements to be sorted\n");
    for(int i=0;i<n;i++)
    scanf("%d",&a[i]);
    clock_t start=clock();
    quicksort(a,0,n-1);
    clock_t end=clock();
    printf("The sorted elements are\n");
    for(int k=0;k<=4;k++)
    {
        printf("%d\t",a[k]);
    }
    printf("\nStart time is %lf\n",(double)start);
    printf("End time is %lf\n",(double)end);
    printf("Total time is %lf\n",(double)(end-start));
    return 0;
}

```

OUTPUT:

```
lab@lab-OptiPlex-3020:~$ ./a.out
Enter the number of elements in the array
5
Enter the elements to be sorted
56 23 12 76 54
The sorted elements are
12      23      54      56      76
Start time is 563.000000
End time is 564.000000
Total time is 1.000000
```

LAB EXCERCISE 5:PRIMS AND KRUSKALS ALGORITHM

Apply Greedy method and Design a C program to find the minimum cost spanning tree using Prim's and Kruskal's Algorithm and compute its complexity.

```
#include<stdio.h>
#include<time.h>
int visited[10]={0}, cost[10][10], min, mincost=0;
int i,j,ne=1, a, b, u, v;;

int main()
{
    int num;
    printf("\n\t\t\tPrim's Algorithm");
    printf("\n\nEnter the number of nodes= ");
    scanf("%d", &num);
    printf("\nEnter the adjacency matrix\n\n");
    for(i=1; i<=num; i++)
    {
        for(j=1; j<=num; j++)
        {
            scanf("%d", &cost[i][j]);
            if(cost[i][j]==0)
                cost[i][j]=999;
        }
    }
    clock_t start=clock();
    visited[1]=1;
    while(ne < num)
    {
        for(i=1,min=999;i<=num;i++)
        for(j=1;j<=num;j++)
        if(cost[i][j]< min)
        if(visited[i]!=0)
        {
```

```

        min=cost[i][j];
        a=u=i;
        b=v=j;
    }
    printf("\n Edge %d:(%d - %d) cost:%d",ne++,a,b,min);
    mincost=mincost+min;
    visited[b]=1;
    cost[a][b]=cost[b][a]=999;
}
printf("\n\n Minimun cost=%d",mincost);
clock_t end=clock();
printf("\nStart time is %lf\n",(double)start);
printf("End time is %lf\n",(double)end);
printf("Total time is %lf\n",(double)(end-start));

return 0;
}

```

OUTPUT:

```

lab@lab-OptiPlex-3020:~$ ./a.out

                          Prim's Algorithm

Enter the number of nodes= 4

Enter the adjacency matrix

0      2      1      4
2      0      3      1
1      3      0      2
4      1      2      0

Edge 1:(1 - 3) cost:1
Edge 2:(1 - 2) cost:2
Edge 3:(2 - 4) cost:1

Minimun cost=4
Start time is 637.000000
End time is 648.000000
Total time is 11.000000

```

ANOTHER PROGRAM FOR PRIM'S

```

#include <stdio.h>
#include <limits.h>
#include<time.h>

#define V 4
int totalcost=0;

int minKey(int key[], int mstSet[])
{

```



```

int min = INT_MAX, min_index;
int v;
for (v = 0; v < V; v++)
{
    if (mstSet[v] == 0 && key[v] < min)
        min = key[v], min_index = v;
}
return min_index;
}

int printMST(int parent[], int n, int graph[V][V])
{
    int i;
    printf("Edge  Weight\n");
    for (i = 1; i < V; i++)
    {
        printf("%d - %d  %d \n", parent[i], i, graph[i][parent[i]]);
        totalcost=totalcost+graph[i][parent[i]];
    }
    printf("totalcost=%d",totalcost);
}

void primMST(int graph[V][V])
{
    int parent[V];           // Array to store constructed MST
    int key[V], i, v, count; // Key values used to pick minimum weight edge in cut
    int mstSet[V];           // To represent set of vertices not yet included in MST

    // Initialize all keys as INFINITE
    for (i = 0; i < V; i++)
        key[i] = INT_MAX, mstSet[i] = 0;
    // Always include first 1st vertex in MST.
    key[0] = 0;           // Make key 0 so that this vertex is picked as first vertex
    parent[0] = -1;       // First node is always root of MST
    // The MST will have V vertices
    for (count = 0; count < V - 1; count++)
    {
        int u = minKey(key, mstSet);
        mstSet[u] = 1;
        for (v = 0; v < V; v++)
        {
            if (graph[u][v] && mstSet[v] == 0 && graph[u][v] < key[v])
                parent[v] = u, key[v] = graph[u][v];
        }
    }

    // print the constructed MST
    printMST(parent, V, graph);
}

int main()
{

```

```

int graph[V][V];
printf("enter the cost adjacency matrix\n");
for(int i=0;i<V;i++)
{
    for(int j=0;j<V;j++)
    {
        scanf("%d",&graph[i][j]);
    }
}

clock_t start=clock();
primMST(graph);
clock_t end=clock();
printf("\nStart time is %lf\n",(double)start);
printf("End time is %lf\n",(double)end);
printf("Total time is %lf\n",(double)(end-start));
return 0;
}

```

OUTPUT:

```

lab@lab-OptiPlex-3020:~$ ./a.out
enter the cost adjacency matrix
0 2 1 4
2 0 3 1
1 3 0 2
4 1 2 0
Edge    Weight
0 - 1    2
0 - 2    1
1 - 3    1
totalcost=4
Start time is 620.000000
End time is 643.000000
Total time is 23.000000

```

KRUSKAL'S ALGORITHM

```

#include<stdio.h>
#include<time.h>
int i,j,k,a,b,u,v,n,ne=1;
int min,mincost=0,cost[9][9],parent[9];
int find(int);
int uni(int,int);
void main()
{
    printf("\n\tImplementation of Kruskal's algorithm\n");
    printf("\nEnter the no. of vertices:");
    scanf("%d",&n);
    printf("\nEnter the cost adjacency matrix:\n");
    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;
    }
}
printf("The edges of Minimum Cost Spanning Tree are\n"); clock_t start=clock();
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;
            }
        }
        u=find(u);
        v=find(v);
        if(uni(u,v))
        {
            printf("%d edge (%d,%d) =%d\n",ne++,a,b,min);
            mincost +=min;
        }
        cost[a][b]=cost[b][a]=999;
    }
    printf("\n\tMinimum cost = %d\n",mincost);
    clock_t end=clock();
    printf("Start time is %lf\n",(double)start);
    printf("End time is %lf\n",(double)end);
    printf("Total time is %lf\n",(double)(end-start));
}
int find(int i)
{
    while(parent[i])
        i=parent[i];
    return i;
}
int uni(int i,int j)
{
    if(i!=j)
    {
        parent[j]=i;
        return 1;
    }
}

```

```

        return 0;
    }

```

OUTPUT:

```

Implementation of Kruskal's algorithm

Enter the no. of vertices:4

Enter the cost adjacency matrix:
0 2 1 4
2 0 3 1
1 3 0 2
4 1 2 0

The edges of Minimum Cost Spanning Tree are
1 edge (1,3) =1
2 edge (2,4) =1
3 edge (1,2) =2

Minimum cost = 4
Start time is 596.000000
End time is 605.000000
Total time is 9.000000

```

LAB EXCERCISE 6:FLOYD'S-WARSHALL ALGORITHM

Apply Dynamic Programming Technique and Design a C program to find the all pairs shortest path using Floyd-Warshall Algorithm and computes its complexity.

```

#include<stdio.h>
#include<time.h>
void floyd(int a[10][10], int n)
{
    for(int k=0;k<n;k++)
    {
        for(int i=0;i<n;i++)
        {
            for(int j=0;j<n;j++)
            {
                if(a[i][j]>a[i][k]+a[k][j])
                {
                    a[i][j]=a[i][k]+a[k][j];
                }
            }
        }
    }
    printf("All Pairs Shortest Path is :\n");
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<n;j++)
        {
            printf("%d ",a[i][j]);
        }
    }
}

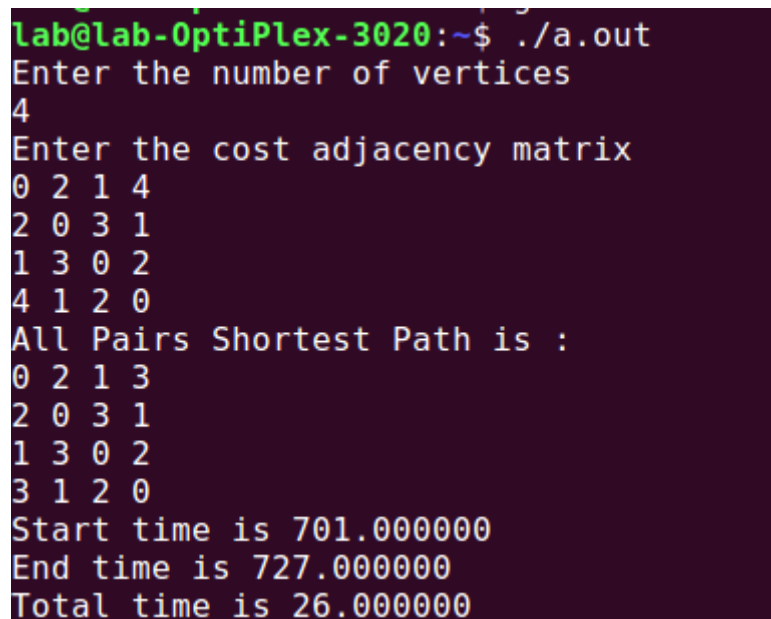
```

```

        printf("\n");
    }
}
int main()
{
    int cost[10][10],n;
    printf("Enter the number of vertices\n");
    scanf("%d",&n);
    printf("Enter the cost adjacency matrix\n");
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<n;j++)
        {
            scanf("%d",&cost[i][j]);
        }
    }
    clock_t start=clock();
    floyd(cost,n);
    clock_t end=clock();
    printf("Start time is %lf\n",(double)start);
    printf("End time is %lf\n",(double)end);
    printf("Total time is %lf\n",(double)(end-start));
    return 0;
}

```

OUTPUT:



```

lab@lab-OptiPlex-3020:~$ ./a.out
Enter the number of vertices
4
Enter the cost adjacency matrix
0 2 1 4
2 0 3 1
1 3 0 2
4 1 2 0
All Pairs Shortest Path is :
0 2 1 3
2 0 3 1
1 3 0 2
3 1 2 0
Start time is 701.000000
End time is 727.000000
Total time is 26.000000

```

LAB EXCERCISE 7:0-1 KNAPSACK

Design a C program to find the optimal solution of 0-1 knapsack problem using dynamic programming and Compute the time complexity.

```

#include<stdio.h>
#include<time.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;
    int 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(val[i-1] + K[i-1][w-wt[i-1]], K[i-1][w]);

            else
                K[i][w] = K[i-1][w];

        }
    }
    return K[n][W];
}

int main()
{
    int i, n, val[20], wt[20], W;
    printf("Enter number of items:");
    scanf("%d", &n);
    printf("Enter value and weight of items:\n");
    for(i = 0; i < n; ++i)
    {
        scanf("%d%d", &val[i], &wt[i]);
    }
    printf("Enter size of knapsack:");
    scanf("%d", &W);
    clock_t start=clock();
    printf("The maximum profit is %d\n", knapSack(W, wt, val, n));
    clock_t end=clock();
    printf("Start time is %lf\n", (double)start);
    printf("End time is %lf\n", (double)end);
    printf("Total time is %lf\n", (double)(end-start));
    return 0;
}

```

OUTPUT:

```

lab@lab-OptiPlex-3020:~$ ./a.out
Enter number of items:4
Enter value and weight of items:
2 3
3 4
1 6
4 5
Enter size of knapsack:8
The maximum profit is 6
Start time is 673.000000
End time is 681.000000
Total time is 8.000000

```

LAB EXCERCISE 8: TRAVELLING SALESMAN PROBLEM

Design a C program to solve the Travelling Salesman Problem using dynamic programming and compute its time complexity.