Aim -> Divide and Conquer sort array

```
#include<stdio.h>
#include<time.h>
double merge(int arr[], int lb, int ub){
    clock_t start,end;
    double cpu = 0.0;
    if (lb<ub){
    start = clock();
        int mid = (lb+ub)/2;
        merge(arr,lb,mid);
        merge(arr,mid+1,ub);
        msort(arr,lb,mid,ub);
    end = clock();
    cpu = cpu+((double)(end-start)/CLOCKS_PER_SEC);
    }
    return cpu;
}

void msort(int arr[], int lb, int mid, int ub){
    int s1=mid-lb+1;
    int s2=ub-mid;
        int L[s1];
        int R[s2];
        int i,j,l,r,k;
        for(i=0;i<s1;i++){
            L[i]=arr[lb+i];
        }
        for(j=0;j<s2;j++){
            R[j]=arr[mid+1+j];
        }
        l=0;
        r=0;
        k=lb;
        while(l<s1 && r<s2){
            if(L[l]<=R[r]){
                    arr[k]=L[l];
                    l++;
            }else{
                    arr[k]=R[r];
                    r++;
            }
            k++;
        }
        while (l<s1){
            arr[k]=L[l];
            l++;
            k++;
```

```
        }
        while (r<s2){
            arr[k]=R[r];
            r++;
            k++;
        }
//  return cpu_time_used;
}

int main(){
    int n = 50;
    int arr[50];
    int i,j,k;
    i=0;
    for(k=50;k>0;k--){
        arr[i]=k;
        i++;
    }
    printf("\nUnsorted Array:\n");
    for(i=0;i<n;i++){
        printf("%d\t",arr[i]);
    }
   double t= merge(arr,0,n-1);
    printf("\nSorted Array:\n");
    for(i=0;i<n;i++){
        printf("%d\t",arr[i]);
    }
    printf("\nTotal Time required is: %f\n",t);
    return 0;
}
```

Aim -> Quick sort

```c
#include<stdio.h>
#include<time.h>
double quicksort(int a[],int first,int last){
    int i, j, pivot, temp;
    clock_t start,end;
    double cpu = 0.0;
    if(first<last){
        start=clock();
            pivot=first;
            i=first;
            j=last;

        while(i<j){
            while(a[i]<=a[pivot]&&i<last){
                i++;
    }
        while(a[j]>a[pivot]){
                j--;
    }
        if(i<j){
            temp=a[i];
            a[i]=a[j];
            a[j]=temp;
         }
      }

        temp=a[pivot];
        a[pivot]=a[j];
        a[j]=temp;
        quicksort(a,first,j-1);
        quicksort(a,j+1,last);
    end = clock();
    cpu = cpu+((double)(end-start)/CLOCKS_PER_SEC);
   }
    return cpu;
}

int main(){
    int i, n,k;
    n=50;
    double t;
    int arr[50];
    i=0;
    for(k=50;k>0;k--){
        arr[i]=k;
        i++;
```

```
    }
    printf("Unsroted Array:\n");
    for(i=0;i<n;i++){
            printf("%d\t",arr[i]);
    }

    t=quicksort(arr,0,n-1);

    printf("\nSorted elements:\n ");
    for(i=0;i<n;i++){
             printf("%d\t",arr[i]);
    }
    printf("\n %f is total time required!!\n",t);

   return 0;
}
```

Aim -> Min Max

```
#include<stdio.h>
#include<stdlib.h>
int mn;
int mx;
int count = 0;
void minmax(int arr[],int l,int h){
    int min,max,i;
    int mid = (l+h)/2;
    min = arr[l];
    max = arr[l];
    for(i=l;i<=h;i++){
        if(min>arr[i]){
            min = arr[i];
        }
        if(max<arr[i]){
            max = arr[i];
        }
    }
    if (mn > arr[count]){
        mn = arr[count];
    }
    if(mx < arr[count]){
        mx = arr[count];
    }
    printf("\n[LOW INDEX:%d\tHIGH INDEX:%d\tMIN:%d\tMAX:%d]",l+1,h+1,min,max);
    count = count + 1;
    if(count<h){
```

```
        minmax(arr,l,mid);
        minmax(arr,mid+1,h);
    }
}
int main(){
    int n,i;
    printf("\nEnter no of Elements:");
    scanf("%d",&n);
    int arr[n];
    for(i=0;i<n;i++){
        printf("\nEnter Element %d:",i+1);
        scanf("%d",&arr[i]);
    }
    printf("\nArray Index:\t");
    for(i=0;i<n;i++){
        printf("%d\t",i+1);
    }
    printf("\n");
    printf("\nArray Elements:\t");
    for(i=0;i<n;i++){
        printf("%d\t",arr[i]);
    }
    mn = arr[count];
    mx = arr[count];
    minmax(arr,0,n-1);
    printf("\nMIN ELEMENT:%d\nMAX ELEMENT:%d",mn,mx);
}
```

Aim -> Optimal Merge pattern

```
#include<stdio.h>
#include<stdlib.h>
int cost = 0;
struct node{
    int data;
    struct node *link;
};

void create(struct node **ptr,int d){
    struct node *temp;
    temp = (struct node*)malloc(sizeof(struct node));
    temp->data = d;
    temp->link = *ptr;
    *ptr = temp;
}
```

```
void display(struct node *ptr){
    printf("\n");
    while(ptr!=NULL){
        printf("%d->",ptr->data);
        ptr = ptr->link;
    }
    printf("\b\b");
}
void sort(struct node *ptr){
    struct node *cur = ptr->link;
    while(ptr!=NULL){
        while(cur!=NULL){
            if((ptr->data)>(cur->data)){
                int temp = ptr->data;
                ptr->data = cur->data;
                cur->data = temp;
            }
            cur = cur->link;
        }
        cur = ptr->link;
        ptr = ptr->link;
    }
}

void merge(struct node **ptr){
    struct node *cur = *ptr;
    struct node *temp;
    temp = (struct node*)malloc(sizeof(struct node));
    int sum;
    sum = cur->data + (cur->link)->data;
    temp->data = sum;
    temp->link = ((*ptr)->link)->link;
    //free(cur);
    //free((cur)->link);
    (*ptr) = temp;
    cost=cost+temp->data;
    sort(*ptr);
    display(*ptr);
     while((*ptr)->link!=NULL){
        merge(ptr);
    }
}
int main(){
    struct node *root = NULL;
    create(&root,28);
    create(&root,32);
    create(&root,12);
```

```
    create(&root,5);
    create(&root,84);
    create(&root,53);
    create(&root,91);
    create(&root,3);
    create(&root,11);
    display(root);
    sort(root);
    printf("\nSorted\n");
    display(root);
    merge(&root);
    printf("\nCost:%d",cost);
    return 0;
}
```

Aim -> Job Scheduling using deadline

```
#include<stdio.h>

void sort(int m, int n, int (*jb)[m][n]){
    int i,j;
    for(i=0;i<n;i++){
        for(j=i+1;j<n;j++){
            if((*jb)[1][i] < (*jb)[1][j]){

                int tempp = (*jb)[1][i];
                (*jb)[1][i] = (*jb)[1][j];
                (*jb)[1][j] = tempp;

                int tempj = (*jb)[0][i];
                (*jb)[0][i] = (*jb)[0][j];
                (*jb)[0][j] = tempj;

                int tempd = (*jb)[2][i];
                (*jb)[2][i] = (*jb)[2][j];
                (*jb)[2][j] = tempd;

            }
        }
    }
}

void printa(int p,int q,int jb[p][q]){
    int i,j;
    for(i=0;i<p;i++){
```

```
                if(i == 0){
                        printf("JOBS:\t");
                }else if(i == 1){
                        printf("PROFIT:\t");
                }else if(i == 2){
                        printf("DEAD:\t");
                }

                for(j=0;j<q;j++){
                       printf("%d\t",jb[i][j]);
                }
                printf("\n");
        }
}

int max(int l,int h,int jb[l][h]){
    int i;
    int m = jb[2][0];
    for(i=0;i<h;i++){
        if(m < jb[2][i]){
            m = jb[2][i];
        }
    }
    return m;
}

void schedule(int l,int h,int jb[l][h], int p, int q, int (*a)[p][q]){
    int flag2 = 0;
    int i,j,max = 0;
    int dead[] = {-1};
    int k = 0,m = 0;
    int track[] = {-1};
    int loc,x=0;
    for(i=0;i<h;i++){
        int flag = 0;
        for(j=0;j<q;j++){
            if(jb[2][i] == dead[j]){
                flag = 1;
                track[m] = jb[1][i];
                m++;
            }
        }
        if(flag == 0){
            (*a)[0][i] = jb[0][i];
            (*a)[1][i] = jb[1][i];
            (*a)[2][i] = jb[2][i];
            dead[k] = jb[2][i];
            k++;
```

```
        }
    }
    /*printf("\nTrack:\n");
    for(i=1;i<m;i++){
        printf("%d\t",track[i]);
    }*/
    while((*a)[1][x]!=-1){
            int mx = 0;
            int start;
            int prev = 0;
            /*for(i=0;i<h;i++){
                if(jb[1][i]==track[0]){
                    start = i+1;
                }
            }*/
            for(i=1;i<m;i++){
                if(track[i]>mx && track[i]!=prev){
                    mx = track[i];
                    prev = mx;
                }
            }
            for(i=0;i<h;i++){
                if(jb[1][i]==mx){
                    loc = i;
                }
            }
            for(i=0;i<q;i++){
                if((*a)[1][i]==-1){
                    (*a)[0][i] = jb[0][loc];
                    (*a)[1][i] = jb[1][loc];
                    (*a)[2][i] = jb[2][loc];
                }
            }
        x++;
    }
}

int main(){
    int n,m,i,j;
    printf("\nEnter no of Processes:");
    scanf("%d",&n);
    int jobs[3][n];
    for(j=0;j<n;j++){
        jobs[0][j] = j+1;
        printf("\nInput Details of Process %d",j+1);
        printf("\nEnter Profit:");
        scanf("%d",&jobs[1][j]);
        printf("Enter Deadline:");
```

```
        scanf("%d",&jobs[2][j]);
    }
    printa(3,n,jobs);
    sort(3,n,&jobs);
    printf("\nSorted:\n");
    printa(3,n,jobs);
    m = max(3,n,jobs);
    printf("\nMax DeadLine:%d",m);
    int arr[3][m];
    for(i=0;i<3;i++){
        for(j=0;j<m;j++){
            arr[i][j]=-1;
        }
    }
    schedule(3,n,jobs,3,m,&arr);
    printf("\n");
    printa(3,m,arr);
    return 0;
}
```

Aim -> Knapsack maximum profit using profit

```
#include<stdio.h>

void knapsack(int n,int arr[3][n], int c){
    int i,j,k,w;
    int profit = 0;
    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            if(arr[2][i]>arr[2][j]){
                int tempj = arr[0][i];
                arr[0][i] = arr[0][j];
                arr[0][j] = tempj;

                int tempw = arr[1][i];
                arr[1][i] = arr[1][j];
                arr[1][j] = tempw;

                int tempp = arr[2][i];
                arr[2][i] = arr[2][j];
                arr[2][j] = tempp;
            }
        }
    }
    printa(n,arr);
```

```
    i = 0;
    while(1){
        if (c >= arr[1][i]){
            c = c - arr[1][i];
            profit = profit + arr[2][i];
        }else{
            w = arr[2][i] * c/arr[1][i];
            c = c - c;
            profit = profit + w;
        }
        i++;
        if(c == 0){
            break;
        }
    }
    printf("\n%d",profit);
}

void printa(int n,int arr[3][n]){
    int i,j;
    for(i=0;i<3;i++){
        if(i == 0){
            printf("JOBS:\t");
        }else if(i == 1){
            printf("WEIGHTS:");
        }else{
            printf("PROFIT:\t");
        }
        for(j=0;j<n;j++){
            printf("%d\t",arr[i][j]);
        }
        printf("\n");
    }
}
int main(){
    int i,j,cap,n;
    printf("\nEnter number of Jobs:");
    scanf("%d",&n);
    int arr[3][n];

    for(j=0;j<n;j++){
        arr[0][j] = j+1;
        printf("\nEnter details of job %d:",j+1);
        printf("\nEnter Weight:");
        scanf("%d",&arr[1][j]);
        printf("Enter Profit:");
        scanf("%d",&arr[2][j]);
    }
```

```
    printf("\nEnter Capacity:");
    scanf("%d",&cap);
    printa(n,arr);

    printf("\n");
    knapsack(n,arr,cap);
    return 0;
}
```

Aim -> Knapsack maximum profit using weight

```
#include<stdio.h>

void knapsack(int n,int arr[3][n], int c){
    int i,j,k;
    int w;
    float pw[n];
    int profit = 0;
    printf("PiWi:");
    for(i=0;i<n;i++){
        pw[i] = (arr[2][i] * 1.0) / (arr[1][i] * 1.0);
        printf("%f\t",pw[i]);
    }
    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            if(pw[i] > pw[j]){
                int tempj = arr[0][i];
                arr[0][i] = arr[0][j];
                arr[0][j] = tempj;

                int tempw = arr[1][i];
                arr[1][i] = arr[1][j];
                arr[1][j] = tempw;

                int tempp = arr[2][i];
                arr[2][i] = arr[2][j];
                arr[2][j] = tempp;

                float temppw = pw[i];
                pw[i] = pw[j];
                pw[j] = temppw;
            }
        }
    }
    printf("\nAfter Sorting:\n");
```

```
    printa(n,arr);
    printf("PiWi:\t");
    for(k=0;k<n;k++){
        printf("%f\t",pw[k]);
    }
    i = 0;
    while(1){
        if (c >= arr[1][i]){
            c = c - arr[1][i];
            profit = profit + arr[2][i];
        }else{
            w = arr[2][i] * c/(arr[1][i] * 1.0);
            c = c - c;
            profit = profit + w;
        }
        i++;
        if(c == 0){
            break;
        }
    }
    printf("\n%d",profit);
}

void printa(int n,int arr[3][n]){
    int i,j;
    for(i=0;i<3;i++){
        if(i == 0){
            printf("\nJOBS:\t");
        }else if(i == 1){
            printf("WEIGHTS:");
        }else{
            printf("PROFIT:\t");
        }
        for(j=0;j<n;j++){
            printf("%d\t",arr[i][j]);
        }
        printf("\n");
    }
}
int main(){
    int i,j,cap,n;
    printf("\nEnter number of Jobs:");
    scanf("%d",&n);
    int arr[3][n];
    for(j=0;j<n;j++){
        arr[0][j] = j+1;
        printf("\nEnter details of job %d:",j+1);
        printf("\nEnter Weight:");
```

```
        scanf("%d",&arr[1][j]);
        printf("Enter Profit:");
        scanf("%d",&arr[2][j]);
    }
    printf("\nEnter Capacity:");
    scanf("%d",&cap);
    printa(n,arr);
    printf("\n");
    knapsack(n,arr,cap);
    return 0;
}
```

Aim -> Floyd Warshall Shortest path

```
#include<stdio.h>

void apspa(int (*arr)[3][3]){
    int dist[3][3];
    int k,i,j;
     for(i=0;i<3;i++){
                for(j=0;j<3;j++){
                     dist[i][j] = (*arr)[i][j];
                }
        }
     for(k=0;k<3;k++){
                for(i=0;i<3;i++){
                        for(j=0;j<3;j++){
                if(dist[i][j] > (*arr)[i][k]+(*arr)[k][j] || dist[i][j] == 99)
 {
                    dist[i][j] = (*arr)[i][k]+(*arr)[k][j];
                }
            }
                }
        }
    printf("\nFinal Matrix:\n");
    printa(dist);
}
void printa(int arr[3][3]){
    int i,j;
    for(i=0;i<3;i++){
            for(j=0;j<3;j++){
                    printf("%d\t",arr[i][j]);
                }
                printf("\n");
        }
```

```
}

int main(){

    int arr[3][3] = {{0,4,11},{6,0,2},{3,99,0}};
    int i,j,n=3;
    printf("\nTotal Vertices:%d\n",n);
    printf("\nDirect Cost Matrix:\n");
    printa(arr);
    apspa(arr);
    return 0;
}
```

Aim -> 0/1 Knapsack

```
#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;
    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];
            }
            printf("%d\t",K[i][w]);
            }
        printf("\n");
    }
   return K[n][W];
}

int main(){
    int i, n, val[20], wt[20], W;
    printf("\nEnter number of items:");
    scanf("%d", &n);
```

```
    printf("Enter Profits and Weights:\n");
    for(i=0;i<n;i++){
        printf("\nEnter Profit %d:",i+1);
        scanf("%d",&val[i]);
        printf("Enter Weight %d:",i+1);
        scanf("%d",&wt[i]);
    }

    printf("Enter size of knapsack:");
    scanf("%d", &W);
    printf("Cost : %d\n", knapSack(W, wt, val, n));
    return 0;
}
```

Aim -> Travelling Salesperson

```
#include<stdio.h>

int ary[10][10],completed[10],n,cost=0;

void takeInput()
{
    int i,j;

    printf("Enter the number of rows and col: ");
    scanf("%d",&n);

    printf("\nEnter the Cost Matrix\n");

    for(i=0;i < n;i++)
    {
        printf("\nEnter Elements of Row: %d\n",i+1);

        for( j=0;j < n;j++)
            scanf("%d",&ary[i][j]);

        completed[i]=0;
    }

    printf("\n\nThe cost list is:");

    for( i=0;i < n;i++)
    {
        printf("\n");

        for(j=0;j < n;j++)
```

```
            printf("\t%d",ary[i][j]);
    }
}

void mincost(int city)
{
    int i,ncity;

    completed[city]=1;

    printf("%d--->",city+1);
    ncity=least(city);

    if(ncity==999)
    {
        ncity=0;
        printf("%d",ncity+1);
        cost+=ary[city][ncity];

        return;
    }

    mincost(ncity);
}

int least(int c)
{
    int i,nc=999;
    int min=999,kmin;

    for(i=0;i < n;i++)
    {
        if((ary[c][i]!=0)&&(completed[i]==0))
            if(ary[c][i]+ary[i][c] < min)
            {
                min=ary[i][0]+ary[c][i];
                kmin=ary[c][i];
                nc=i;
            }
    }

    if(min!=999)
        cost+=kmin;

    return nc;
}

int main()
```

```
{
    takeInput();

    printf("\n\nThe Path is:\n");
    mincost(0); //passing 0 because starting vertex

    printf("\n\nMinimum cost is %d\n ",cost);

    return 0;
}
```

Aim -> Chain multiplication of matrix

```
/* A naive recursive implementation that simply
follows the above optimal substructure property */
#include<stdio.h>
#include<limits.h>

// Matrix Ai has dimension p[i-1] x p[i] for i = 1..n
int MatrixChainOrder(int p[], int i, int j)
{
    if(i == j)
        return 0;
    int k;
    int min = INT_MAX;
    int count;

    // place parenthesis at different places between first
    // and last matrix, recursively calculate count of
    // multiplications for each parenthesis placement and
    // return the minimum count
    for (k = i; k <j; k++)
    {
        count = MatrixChainOrder(p, i, k) +
                MatrixChainOrder(p, k+1, j) +
                p[i-1]*p[k]*p[j];

        if (count < min)
            min = count;
    }

    // Return minimum count
    return min;
}

// Driver program to test above function
```

```
int main()
{
    int arr[] = {1, 2, 3, 4, 3};
    int n = sizeof(arr)/sizeof(arr[0]);

    printf("Minimum number of multiplications is %d ",
                         MatrixChainOrder(arr, 1, n-1));
    return 0;
}
```

Aim -> OBST

```
// A naive recursive implementation of optimal binary
// search tree problem
#include <stdio.h>
#include <limits.h>

// A utility function to get sum of array elements
// freq[i] to freq[j]
int sum(int freq[], int i, int j);

// A recursive function to calculate cost of optimal
// binary search tree
int optCost(int freq[], int i, int j)
{
// Base cases
if (j < i)   // no elements in this subarray
    return 0;
if (j == i)  // one element in this subarray
    return freq[i];

// Get sum of freq[i], freq[i+1], ... freq[j]
int fsum = sum(freq, i, j);

// Initialize minimum value
int min = INT_MAX;

// One by one consider all elements as root and
// recursively find cost of the BST, compare the
// cost with min and update min if needed
for (int r = i; r <= j; ++r)
{
    int cost = optCost(freq, i, r-1) +
               optCost(freq, r+1, j);
```

```
    if (cost < min)
        min = cost;
}

// Return minimum value
return min + fsum;
}

// The main function that calculates minimum cost of
// a Binary Search Tree. It mainly uses optCost() to
// find the optimal cost.
int optimalSearchTree(int keys[], int freq[], int n)
{
    // Here array keys[] is assumed to be sorted in
    // increasing order. If keys[] is not sorted, then
    // add code to sort keys, and rearrange freq[]
    // accordingly.
    return optCost(freq, 0, n-1);
}

// A utility function to get sum of array elements
// freq[i] to freq[j]
int sum(int freq[], int i, int j)
{
    int s = 0;
    for (int k = i; k <=j; k++)
    s += freq[k];
    return s;
}

// Driver program to test above functions
int main()
{
    int keys[] = {10, 12, 20};
    int freq[] = {34, 8, 50};
    int n = sizeof(keys)/sizeof(keys[0]);
    printf("Cost of Optimal BST is %d ",
            optimalSearchTree(keys, freq, n));
    return 0;
}
```