## Merge Sort

| | |
|---|---|
| #include<stdio.h><br>void mergeSort(int a[], int n)<br>{ | **mergeSort নামের function বানানো**<br>• a[] = array<br>• n = array size<br>• void = কিছু return করবে না |
| int i, j, k, mid; | **loop এবং index এর জন্য variable**<br>• i → left array index<br>• j → right array index<br>• k → main array index<br>• mid → মাঝখানের position |
| int L[50], R[50]; | **দুইটা temporary array**<br>• L = Left part<br>• R = Right part<br>৫০ দেওয়া হয়েছে যেন জায়গা থাকে। |
| if(n < 2) return; | **Base condition**<br>যদি element 1 টা বা 0 টা হয় — already sorted<br>তাই function বন্ধ। |
| mid = n / 2; | array কে দুই ভাগে ভাগ করার জন্য middle বের করা |
| for(i=0;i<mid;i++)<br>L[i] = a[i]; | প্রথম half copy হচ্ছে L[] তে |
| for(i=mid;i<n;i++)<br>R[i-mid] = a[i]; | দ্বিতীয় half copy হচ্ছে R[] তে |
| mergeSort(L, mid); | Left array আবার sort করার জন্য recursive call |
| mergeSort(R, n-mid); | Right array sort করার জন্য recursive call |
| i=j=k=0; | সব index reset |
| while(i<mid && j<n-mid)<br>{ | যতক্ষণ দুই array তেই element আছে |
| if(L[i] < R[j])<br>a[k++] = L[i++]; | ছোট element main array তে ঢুকছে |
| else<br>a[k++] = R[j++];<br>} | না হলে right array থেকে ঢুকছে |

| | |
|---|---|
| while(i<mid)<br>a[k++] = L[i++]; | Left array এ extra element থাকলে copy |
| while(j<n-mid)<br>a[k++] = R[j++];<br>} | Right array এ extra element থাকলে copy |
| int main()<br>{ | Program শুরু |
| int a[] = {38,27,43,3,9,82,10}; | Input array |
| int n = 7; | size |
| mergeSort(a,n); | sorting শুরু |
| printf("Sorted Array:\n"); | Print message |
| for(int i=0;i<n;i++)<br>printf("%d ",a[i]);<br>} | sorted array print |

## Closest Pair (Divide & Conquer)

| | |
|---|---|
| #include<stdio.h><br>#include<math.h> | **stdio.h**<br>printf() ব্যবহার করার জন্য।<br>**math.h**<br>sqrt() (square root) ব্যবহার করার জন্য। |
| double dist(int x1,int y1,int x2,int y2)<br>{ | **দুইটা point এর distance বের করার function।**<br>• (x1,y1) → প্রথম point<br>• (x2,y2) → দ্বিতীয় point<br>Return type double কারণ distance decimal হতে পারে। |
| return sqrt((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2));<br>} | **Euclidean distance formula:**<br>(x1-x2)2+(y1-y2)2\sqrt{(x1-x2)^2 + (y1-y2)^2}(x1-x2)2+(y1-y2)2<br>মানে:<br>1. x difference square<br>2. y difference square<br>3. যোগ<br>4. square root<br>এটাই distance। |
| double closest(int x[], int y[], int l, int r) | **এই function closest distance বের করে।** |

| | |
|---|---|
| { | • x[] → সব x coordinate<br>• y[] → সব y coordinate<br>• l → left index<br>• r → right index<br>Return করবে minimum distance। |
| if(r - l == 1)<br>return dist(x[l],y[l],x[r],y[r]); | **যদি শুধু 2টা point থাকে:**<br>তাদের distance বের করে return।<br>Example:<br>index 0 & 1<br>বা<br>index 1 & 2 |
| int mid = (l+r)/2; | array মাঝখান থেকে ভাগ করার জন্য। |
| double d1 = closest(x,y,l,mid); | **বাম পাশের points এর minimum distance বের করে।**<br>(recursive call) |
| double d2 = closest(x,y,mid,r); | ডান পাশের points এর minimum distance বের করে। |
| return d1 < d2 ? d1 : d2;<br>} | d1 আর d2 এর মধ্যে যেটা ছোট — সেটাই return। |
| int main()<br>{ | Program শুরু। |
| int x[]={2,4,5};<br>int y[]={3,1,4}; | 3টা point: |
| double ans = closest(x,y,0,2); | **closest function call।**<br>মানে:<br>index 0 থেকে 2 পর্যন্ত point নিয়ে কাজ করো।<br>Result ans এ রাখো। |
| printf("Minimum Distance = %.2lf",ans);<br>} | **final minimum distance print।**<br>%.2lf মানে decimal এর পর 2 digit দেখাবে। |

## Peak Element (Divide & Conquer)

| | |
|---|---|
| #include<stdio.h> | **stdio library add করা হয়েছে**<br>কারণ আমরা printf() ব্যবহার করবো। |
| int peak(int a[], int l, int r)<br>{ | **peak নামের function বানানো**<br>• a[] → array |

| | |
|---|---|
| | • l → left index<br>• r → right index<br>এই function peak value return করবে → তাই int |
| int m = (l+r)/2; | array এর মাঝখানের index বের করা |
| if(a[m] > a[m-1] && a[m] > a[m+1]) | **check করা হচ্ছে:**<br>মাঝখানের element কি তার দুই পাশের element থেকে বড়? |
| return a[m]; | **যদি বড় হয় → এইটাই peak**<br>তাই return করে দাও। |
| if(a[m-1] > a[m]) | যদি বাম পাশ বড় হয় |
| return peak(a,l,m-1); | তাহলে বাম পাশে আবার search করো (recursion) Divide & Conquer part। |
| else<br>return peak(a,m+1,r);<br>} | না হলে ডান পাশে search করো |
| int main()<br>{ | Program শুরু। |
| int a[]={1,3,20,4,1,0}; | Input array |
| printf("Peak = %d", peak(a,1,4));<br>} | **peak function call**<br>কেন 1,4?<br>কারণ:<br>a[0] আর a[5] এর neighbor নাই<br>তাই safe range:<br>3 20 4 1 |

## Fractional Knapsack – (Greedy)

| | |
|---|---|
| #include<stdio.h> | printf() ব্যবহার করার জন্য stdio যোগ করা হয়েছে। |
| int main()<br>{ | Program শুরু। |
| float weight[] = {10,20,30}; | **তিনটা item এর weight:**<br>Item1 = 10<br>Item2 = 20<br>Item3 = 30 |
| float value[] = {60,100,120}; | **তিনটা item এর value:** |

| | |
|---|---|
| | Item1 = 60<br>Item2 = 100<br>Item3 = 120 |
| float capacity = 50; | **Knapsack এর মোট capacity = 50**<br>মানে ব্যাগে সর্বোচ্চ 50 weight ঢুকবে। |
| float ratio[3]; | এখানে value/weight ratio রাখা হবে। |
| float profit = 0; | **মোট profit শুরুতে 0।** |
| for(int i=0;i<3;i++) | loop চলবে 3 বার (৩টা item)। |
| ratio[i] = value[i] / weight[i]; | **প্রতিটা item এর:**<br>value / weight<br>হিসাব করা হচ্ছে। |
| for(int i=0;i<3;i++)<br>{ | এক এক করে item নেবে। |
| if(weight[i] <= capacity)<br>{ | item এর weight কি capacity এর ভিতরে? |
| capacity -= weight[i]; | **capacity কমানো।** |
| profit += value[i];<br>} | profit এ পুরো value যোগ। |
| else<br>{ | fractional অংশ নেওয়া হচ্ছে। |
| profit += ratio[i] * capacity;<br>break;<br>}<br>} | bag ভর্তি — loop বন্ধ। |
| printf("Maximum Profit = %.2f", profit);<br>} | **final profit print।**<br>.2f মানে decimal এর পর 2 digit দেখাবে। |

## All Pairs Shortest Path (Floyd–Warshall)

| | |
|---|---|
| #include<stdio.h> | #include<stdio.h> |
| #define INF 999 | INF মানে খুব বড় সংখ্যা।<br>যেখানে direct রাস্তা নাই, সেখানে 999 বসানো হয়েছে। |
| int main()<br>{ | Program শুরু। |
| int n = 4; | মোট ৪টা node / city। |
| int g[4][4] = { | এটা distance table (matrix)। |

| | |
|---|---|
| {0, 5, INF, 10},<br>{INF, 0, 3, INF},<br>{INF, INF, 0, 1},<br>{INF, INF, INF, 0}<br>}; | মানে:<br>Row = কোথা থেকে<br>Column = কোথায় |
| for(int k=0;k<n;k++) | k = মাঝখানের node (intermediate) |
| for(int i=0;i<n;i++) | i = শুরু node |
| for(int j=0;j<n;j++) | j = শেষ node<br>সব possible combination check করা হচ্ছে:<br>i → k → j |
| if(g[i][j] > g[i][k] + g[k][j]) | Check করছে:<br>direct i→j এর চেয়ে<br>i→k→j কি ছোট? |
| g[i][j] = g[i][k] + g[k][j]; | distance update করে দিচ্ছে।<br>এইটাই **Dynamic Programming**।<br>আগের result ব্যবহার করে নতুন result বানাচ্ছে। |
| printf("All Pairs Shortest Path:\n"); | message print। |
| for(int i=0;i<n;i++)<br>{ | row loop। |
| for(int j=0;j<n;j++)<br>printf("%d ", g[i][j]); | matrix এর সব value print। |
| printf("\n");<br>}<br>} | next line। |

## Huffman Tree

| | |
|---|---|
| #include<stdio.h><br>#include<stdlib.h> | **stdio.h**<br>printf() ব্যবহার করার জন্য।<br>👉 **stdlib.h**<br>malloc() ব্যবহার করার জন্য (memory বানাতে লাগে)। |
| struct Node{<br>char ch;<br>int f; | **Huffman tree এর একেকটা node।**<br>এর ভিতরে:<br>• ch → character (a,b,c...) |

| | |
|---|---|
| struct Node *l,*r;<br>}; | • f → frequency<br>• l → left child<br>• r → right child<br>মানে:<br>একটা box যেটার ভিতরে character + frequency + দুই পাশের link আছে। |
| struct Node* new(char c,int f){ | নতুন node বানানোর function। |
| struct Node* t=(struct Node*)malloc(sizeof(struct Node)); | **computer কে বলছে:**<br>"একটা Node এর জন্য memory দাও।" |
| t->ch=c; t->f=f; | **node এর ভিতরে character আর frequency বসানো।** |
| t->l=t->r=NULL; | শুরুতে left/right কিছু নাই। |
| return t;<br>} | node return। |
| void print(struct Node* root,int a[],int i){ | tree traverse করে Huffman code print করে।<br><br>root → current node<br><br>a[] → 0/1 রাখার array<br><br>i → index |
| if(root->l){ a[i]=0; print(root->l,a,i+1); } | বামে গেলে 0 বসায়। |
| if(root->r){ a[i]=1; print(root->r,a,i+1); } | Right গেলে 1 |
| if(!root->l && !root->r){ | যদি কোনো child না থাকে → এটা character node। |
| printf("%c : ",root->ch); | Print character |
| for(int j=0;j<i;j++)<br>printf("%d",a[j]); | **Print binary code** |
| printf("\n");<br>}<br>} | |
| int main(){ | Program শুরু। |
| struct Node* root=new('$',100); | root node।<br>$ মানে internal node (character না)। |
| root->l=new('f',45); | left child = f |

| | |
|---|---|
| root->r=new('$',55); | এইভাবে নিচের সব line দিয়ে পুরো Huffman tree connect করা হয়েছে। |
| root->r->l=new('c',12);<br>root->r->r=new('$',43);<br><br>root->r->r->l=new('$',25);<br>root->r->r->l->l=new('a',5);<br>root->r->r->l->r=new('b',9);<br><br>root->r->r->r=new('$',18);<br>root->r->r->r->l=new('d',13);<br>root->r->r->r->r=new('e',16); | root → right → right → left → left = a |
| int arr[10];<br>print(root,arr,0);<br>} | Huffman code বের করা শুরু। |

## Travelling Salesperson Problem (TSP – Dynamic Programming)

| | |
|---|---|
| #include<stdio.h> | printf() ব্যবহার করার জন্য। |
| #define N 4 | মোট 4টা city। |
| #define INF 999 | খুব বড় সংখ্যা — শুরুতে minimum খোঁজার সময় ব্যবহার হয়। |
| int dist[N][N] = {<br>{0,10,15,20},<br>{10,0,35,25},<br>{15,35,0,30},<br>{20,25,30,0}<br>}; | কোন city থেকে কোন city যেতে কত খরচ। |
| int visited[N] = {0}; | কোন city ঘোরা হয়েছে সেটা রাখে।<br>0 = না<br>1 = হ্যাঁ |
| int dp[N][N]; | আগের result save রাখে।<br>এইটাই **Dynamic Programming**। |
| int min(int a,int b){<br>return a<b?a:b;<br>} | দুইটার মধ্যে ছোটটা দেয়। |
| int tsp(int city, int count) | মূল algorithm। |

| | |
|---|---|
| { | • city → এখন কোন city তে আছি<br>• count → কয়টা city ঘোরা হয়েছে |
| if(count == N)<br>return dist[city][0]; | সব city ঘোরা শেষ হলে:<br>শুরুতে (city 0) ফিরে যাওয়ার cost return। |
| if(dp[city][count] != -1)<br>return dp[city][count]; | আগেই হিসাব করা থাকলে সেটাই ব্যবহার করো। |
| int ans = INF; | খুব বড় মান দিয়ে শুরু। |
| for(int i=0;i<N;i++)<br>{ | সব city check করবে। |
| if(!visited[i])<br>{ | যদি city visit না হয়ে থাকে: |
| visited[i] = 1; | এখন এই city ঘোরা হলো। |
| ans = min(ans, dist[city][i] + tsp(i, count+1)); | মানে:<br>👉 current city → i<br>👉 তারপর i থেকে বাকি city<br>সবচেয়ে কমটা রাখো। |
| visited[i] = 0;<br>}<br>} | Unmark visited<br>backtracking। |
| return dp[city][count] = ans;<br>} | Save DP result<br>result dp তে রেখে return। |
| int main()<br>{ | Program শুরু। |
| for(int i=0;i<N;i++)<br>for(int j=0;j<N;j++)<br>dp[i][j] = -1; | **dp initialize**<br>dp array empty করা। |
| visited[0] = 1; | city 0 থেকে শুরু। |
| printf("Minimum Tour Cost = %d", tsp(0,1));<br>} | tsp function call:<br>city = 0<br>count = 1<br>Final minimum cost print। |

## Binary Search (Divide & Conquer)

| | |
|---|---|
| #include<stdio.h> | printf() ব্যবহার করার জন্য standard input/output library যোগ করা হয়েছে। |

| | |
|---|---|
| int binarySearch(int a[], int l, int r, int key)<br>{ | binarySearch নামে function বানানো।<br>• a[] → array<br>• l → left index<br>• r → right index<br>• key → যেটা খুঁজছি<br>এই function index return করবে → তাই int |
| if(l > r)<br>return -1; | যদি left index বড় হয়ে যায় right index থেকে,<br>মানে element নাই।<br>তাই -1 return। |
| int mid = (l + r) / 2; | মাঝখানের index বের করা। |
| if(a[mid] == key)<br>return mid; | যদি মাঝখানের element == key হয়,<br>তাহলে index return করে দাও। |
| if(key < a[mid]) | যদি key মাঝখানের element থেকে ছোট হয়,<br>মানে বাম পাশে আছে। |
| return binarySearch(a, l, mid-1, key); | বাম অর্ধেকে আবার binary search চালাও (recursion)<br>এটাই Divide & Conquer। |
| else<br>return binarySearch(a, mid+1, r, key);<br>} | না হলে ডান অর্ধেকে search করো। |
| int main()<br>{ | Program শুরু। |
| int a[] = {2,4,6,8,10,12,14}; | Sorted array (Binary search এর জন্য sorted লাগেই) |
| int n = 7; | মোট element = 7 |
| int key = 10; | আমরা 10 খুঁজছি। |
| int pos = binarySearch(a, 0, n-1, key); | binarySearch call করা হচ্ছে:<br>• array = a<br>• left = 0<br>• right = 6<br>• key = 10<br>Result pos এ রাখছে। |
| if(pos == -1)<br>printf("Element not found"); | যদি -1 আসে → element নাই। |

| | |
|---|---|
| else<br>printf("Element found at index %d", pos);<br>} | না হলে index print। |

## Max-Min (Divide & Conquer)

```
#include<stdio.h>


int max, min;


void findMaxMin(int a[], int l, int r)
{
    if(l == r)   // only one element
    {
        max = min = a[l];
        return;
    }


    if(l + 1 == r)   // two elements
    {
        if(a[l] > a[r])
        {
            max = a[l];
            min = a[r];
        }
        else
        {
```

```
            max = a[r];
            min = a[l];
        }
        return;
    }

    int m = (l+r)/2;

    findMaxMin(a, l, m);      // left half
    findMaxMin(a, m+1, r);  // right half
}

int main()
{
    int a[] = {7,2,9,4,1,5};
    int n = 6;

    findMaxMin(a,0,n-1);

    printf("Maximum = %d\n", max);
    printf("Minimum = %d", min);
}
```

## Quick Sort (Divide & Conquer)

```
#include<stdio.h>


void quickSort(int a[], int l, int r)
{
    int i=l, j=r, pivot=a[(l+r)/2];


    while(i<=j)
    {
        while(a[i] < pivot) i++;
        while(a[j] > pivot) j--;


        if(i<=j)
        {
            int t=a[i];
            a[i]=a[j];
            a[j]=t;
            i++; j--;
        }
    }


    if(l < j) quickSort(a,l,j);
    if(i < r) quickSort(a,i,r);
}
```

```
int main()
{
    int a[]={10,7,8,9,1,5};
    int n=6;

    quickSort(a,0,n-1);

    for(int i=0;i<n;i++)
        printf("%d ",a[i]);
}
```

## Selection Sort (Divide & Conquer)

```
#include<stdio.h>

// Recursive Selection Sort
void selectionSort(int a[], int start, int n)
{
    if(start >= n-1)    // base case
        return;

    int min = start;
```

```
    // find minimum index
    for(int i=start+1;i<n;i++)
        if(a[i] < a[min])
            min = i;

    // swap
    int temp = a[start];
    a[start] = a[min];
    a[min] = temp;

    // recursive call for remaining part
    selectionSort(a, start+1, n);
}

int main()
{
    int a[] = {64,25,12,22,11};
    int n = 5;

    selectionSort(a,0,n);

    printf("Sorted Array:\n");
    for(int i=0;i<n;i++)
        printf("%d ",a[i]);
```

}

## Prim's Algorithm (Greedy)

```
#include<stdio.h>


#define V 4

#define INF 999


int main()

{

    int g[V][V] = {

        {0,10,6,5},

        {10,0,0,15},

        {6,0,0,4},

        {5,15,4,0}

    };


    int visited[V]={0};

    int edges=0, min, x, y;


    visited[0]=1;   // start from node 0


    printf("Edges in MST (Prim):\n");


    while(edges < V-1)
```

```
    {
        min = INF;

        for(int i=0;i<V;i++)
            if(visited[i])
                for(int j=0;j<V;j++)
                    if(!visited[j] && g[i][j])
                        if(g[i][j] < min)
                        {
                            min = g[i][j];
                            x=i; y=j;
                        }

        printf("%d - %d : %d\n", x, y, g[x][y]);
        visited[y]=1;
        edges++;
    }
}
```

## Kruskal's Algorithm (Greedy)

```
#include<stdio.h>

#define V 4

int parent[V];
```

```
int find(int i){
    while(parent[i])
        i = parent[i];
    return i;
}

void uni(int i,int j){
    parent[j] = i;
}

int main()
{
    int g[V][V] = {
        {0,10,6,5},
        {10,0,0,15},
        {6,0,0,4},
        {5,15,4,0}
    };

    int min, a, b, u, v, edges=0;

    printf("Edges in MST (Kruskal):\n");
```

```
while(edges < V-1)
{
    min = 999;

    for(int i=0;i<V;i++)
        for(int j=0;j<V;j++)
            if(g[i][j] && g[i][j] < min)
            {
                min = g[i][j];
                a=u=i; b=v=j;
            }

    u = find(u);
    v = find(v);

    if(u != v)
    {
        printf("%d - %d : %d\n", a, b, min);
        uni(u,v);
        edges++;
    }

    g[a][b] = g[b][a] = 999;    // remove used edge
}
```

}

## Dijkstra Algorithm

```
#include<stdio.h>


#define V 5

#define INF 999


int main()

{

    // Graph (Adjacency Matrix)

    int g[V][V] = {

        {0,10,0,5,0},

        {0,0,1,2,0},

        {0,0,0,0,4},

        {0,3,9,0,2},

        {7,0,6,0,0}

    };


    int dist[V];            // shortest distance from source

    int visited[V]={0};  // visited nodes


    int src = 0;           // source node


    // Step 1: initialize all distances as INF
```

```
for(int i=0;i<V;i++)
    dist[i] = INF;

dist[src] = 0;        // distance of source = 0

// Step 2: main Dijkstra loop
for(int c=0;c<V-1;c++)
{
    int min = INF, u;

    // find unvisited node with minimum distance
    for(int i=0;i<V;i++)
        if(!visited[i] && dist[i] < min)
        {
            min = dist[i];
            u = i;
        }

    visited[u] = 1;   // mark selected node

    // update adjacent nodes
    for(int v=0; v<V; v++)
        if(!visited[v] && g[u][v] &&
           dist[u] + g[u][v] < dist[v])
```

```
                dist[v] = dist[u] + g[u][v];
    }

    // Print result
    printf("Vertex   Distance from Source\n");
    for(int i=0;i<V;i++)
        printf("%d          %d\n", i, dist[i]);
}
```

## Job Sequencing with Deadline (Greedy)

```
#include<stdio.h>

int main()
{
    // Jobs
    char job[] = {'A','B','C','D'};
    int profit[] = {100,19,27,25};
    int deadline[] = {2,1,2,1};

    int n = 4;

    // Step 1: Sort jobs by profit (descending)
    for(int i=0;i<n;i++)
        for(int j=i+1;j<n;j++)
            if(profit[i] < profit[j])
```

```
        {
            int t = profit[i]; profit[i]=profit[j]; profit[j]=t;
            t = deadline[i]; deadline[i]=deadline[j]; deadline[j]=t;
            char c = job[i]; job[i]=job[j]; job[j]=c;
        }

int slot[10]={0};   // time slots
char result[10];

// Step 2: Select jobs greedily
for(int i=0;i<n;i++)
{
    for(int j=deadline[i]-1; j>=0; j--)
    {
        if(slot[j]==0)
        {
            slot[j]=1;
            result[j]=job[i];
            break;
        }
    }
}

printf("Selected Jobs: ");
```

```
    for(int i=0;i<n;i++)
        if(slot[i])
            printf("%c ", result[i]);
}
```

## LCS (Dynamic Programming)

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

int max(int a,int b){
    return a>b?a:b;
}

int main()
{
    char s1[] = "AGGTAB";
    char s2[] = "GXTXAYB";

    int m = strlen(s1);
    int n = strlen(s2);

    int dp[m+1][n+1];

    // Build DP table
    for(int i=0;i<=m;i++)
```

```
    {
        for(int j=0;j<=n;j++)
        {
            if(i==0 || j==0)
                dp[i][j] = 0;

            else if(s1[i-1] == s2[j-1])
                dp[i][j] = dp[i-1][j-1] + 1;

            else
                dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
        }
    }

    printf("Length of LCS = %d\n", dp[m][n]);
}
```