**AOA**

1. **Insertion Sort : Best - theta(n) , Worst,Average - theta(nsqr)**

#include<stdio.h>

void main()

{ int n,i,j,key;

int a[10];

printf("Enter size of array: ");

scanf("%d",&n);

printf("Enter elements of array: ");

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

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

}

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

{ key=a[j];

i=j-1;

while(i>=0 && a[i]>key)

{ a[i+1]=a[i];

i=i-1;

}

a[i+1]=key;

}

printf("After sorting, we get: ");

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

{ printf("%d ",a[i]);

}

}

**2) Selection sort : theta(nsqr)**

#include<stdio.h>

void main()

{ int n,i,j,a[10],smallest,temp;

printf("Enter size of array: ");

scanf("%d",&n);

printf("Enter the elements of array: ");

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

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

}

for(j=0;j<=n-1;j++)

{ smallest=j;

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

{ if(a[i]<a[smallest])

{ smallest=i;

}

}

temp=a[j];

a[j]=a[smallest];

a[smallest]=temp;

}

printf("After sorting, we get: ");

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

{ printf("%d ",a[i]);

}

}

**3) Merge Sort : theta(nlogn)**

#include <stdio.h>

void Merge(int arr[], int left, int mid, int right);

void MergeSort(int arr[], int left, int right);

int main()

{ int n;

printf("Enter size of array: ");

scanf("%d", &n);

int arr[n];

printf("Enter array elements: ");

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

{ scanf("%d", &arr[i]);

}

MergeSort(arr, 0, n - 1);

printf("Sorted array: ");

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

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

}

return 0;

}

void Merge(int arr[], int left, int mid, int right)

{ int n1 = mid - left + 1;

int n2 = right - mid;

int leftArray[n1];

int rightArray[n2];

for (int i = 0; i < n1; i++)

{ leftArray[i] = arr[left + i];

}

for (int j = 0; j < n2; j++)

{ rightArray[j] = arr[mid + 1 + j];

}

int i = 0, j = 0, k = left;

while (i < n1 && j < n2)

{ if (leftArray[i] <= rightArray[j])

{ arr[k] = leftArray[i];

i++;

}

else

{ arr[k] = rightArray[j];

j++;

}

k++;

}

while (i < n1)

{ arr[k] = leftArray[i];

i++;

k++;

}

while (j < n2)

{ arr[k] = rightArray[j];

j++;

k++;

}

}

void MergeSort(int arr[], int left, int right)

{ if (left < right)

{ int mid = (left + right) / 2;

MergeSort(arr, left, mid);

MergeSort(arr, mid + 1, right);

Merge(arr, left, mid, right);

}

}

**4) Quick Sort : Worst - theta(nsqr) Best,Average - theta(nlogn)**

#include <stdio.h>

void swap(int\* a, int\* b)

{ int t = \*a;

\*a = \*b;

\*b = t;

}

int partition(int arr[], int low, int high)

{ int pivot = arr[high];

int i = (low - 1);

int count = 0;

for (int j = low; j <= high - 1; j++)

{ if (arr[j] < pivot)

{ i++;

swap(&arr[i], &arr[j]);

count++;

}

}

swap(&arr[i + 1], &arr[high]);

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)

{ int i;

for (i = 0; i < size; i++)

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

printf("\n");

}

int main()

{ int x;

int count;

printf("Enter number of array elements : ");

scanf("%d", &x);

int arr[x];

printf("Enter array elements : ");

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

{ scanf("%d", &arr[i]);

}

int n = sizeof(arr) / sizeof(arr[0]);

quickSort(arr, 0, n - 1);

printf("Sorted array: \n");

printArray(arr, n);

return 0;

}

**5) MinMax by Divide and Conquer : 0,1,T(n/2)+T(n/2)+2 (n=1,2,>2)**

#include<stdio.h>

int m, n;

int A[100];

void maxmin(int l, int r)

{

int max, min, mid;

if(l==r) //1 element in list

{

m = n = A[l];

}

else

{

if(l == r-1) //2 elements in list

{

if(A[l] <A[r])

{

m = A[r];

n = A[l];

}

else

{

m = A[l];

n = A[r];

}

}

else

{

mid = (l+r)/2; //>2 elements in list

printf("Mid element : %d\n",mid);

maxmin(l, mid);

max = m; min = n;

maxmin(mid+1, r);

if(m <max)

m = max;

printf("Max element : %d\n",m);

if(n > min)

n = min;

printf("Min element : %d\n",n);

}

}

}

int main ()

{

int i, num;

printf ("\nEnter the total number of numbers : ");

scanf ("%d",&num);

printf ("Enter the numbers : \n");

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

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

m = A[0];

n = A[0];

maxmin(1, num);

printf ("Minimum element in an array : %d\n", n);

printf ("Maximum element in an array : %d\n", m);

return 0;

}

**6) Binary Search : Worst - theta(logn) Average,Best - 1**

#include <stdio.h>

int binary\_search(int A[], int key, int len) {

int low = 0;

int high = len - 1;

while (low <= high) {

int mid = low + ((high - low) / 2);

if (A[mid] == key) {

return mid;

}

if (key < A[mid]) {

high = mid - 1;

} else {

low = mid + 1;

}

}

return -1;}

int main() {

int n;

printf("Enter the number of elements in the array: ");

scanf("%d", &n);

int a[n];

printf("Enter %d sorted elements: ", n);

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

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

}

int key;

printf("Enter the key to search for: ");

scanf("%d", &key);

int position = binary\_search(a, key, n);

if (position == -1) {

printf("Key %d not found in the array.\n", key);

} else {

printf("Key %d found at index %d.\n", key, position);

}

return 0;

}

**7) Prim's Algorithm for MST : O(|E|log|V|)**

#include<stdio.h>

#define INF 9999

int main()

{ int n;

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

scanf("%d", &n);

int g[n][n];

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

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

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

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

}

}

int t[n];

int mincost = 0;

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

{ t[i] = 0;

}

t[0] = 1;

printf("Edges of the minimum spanning tree:\n");

for(int k = 0; k < n - 1; k++)

{ int min\_edge = INF, v1, v2;

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

{ if(t[i] == 1)

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

{ if(g[i][j] > 0 && t[j] == 0 && g[i][j] < min\_edge)

{ min\_edge = g[i][j];

v1 = i;

v2 = j;

}

}

}

}

t[v2] = 1;

mincost += min\_edge;

printf("%d - %d : %d\n", v1, v2, min\_edge);

}

printf("Total cost of the minimum spanning tree: %d\n", mincost);

return 0;

**8) Kruskal's Algorithm for MST : O(|E|log|E|)**

#include <stdio.h>

#include <stdlib.h>

typedef struct {

int u, v, weight;

} Edge;

typedef struct {

int parent, rank;

} Subset;

int compare(const void \*a, const void \*b) {

Edge \*e1 = (Edge \*)a;

Edge \*e2 = (Edge \*)b;

return e1->weight - e2->weight;

}

int find(Subset subsets[], int i) {

if (subsets[i].parent != i) {

subsets[i].parent = find(subsets, subsets[i].parent);

}

return subsets[i].parent;

}

void unionSets(Subset subsets[], int x, int y) {

int rootX = find(subsets, x);

int rootY = find(subsets, y);

if (subsets[rootX].rank < subsets[rootY].rank) {

subsets[rootX].parent = rootY;

} else if (subsets[rootX].rank > subsets[rootY].rank) {

subsets[rootY].parent = rootX;

} else {

subsets[rootY].parent = rootX;

subsets[rootX].rank++;

}

}

void kruskal(int vertices, int edgeCount, Edge edges[]) {

qsort(edges, edgeCount, sizeof(Edge), compare);

Subset subsets[vertices];

for (int i = 0; i < vertices; i++) {

subsets[i].parent = i;

subsets[i].rank = 0;

}

Edge result[vertices - 1];

int e = 0;

int i = 0;

while (e < vertices - 1 && i < edgeCount) {

Edge nextEdge = edges[i++];

int x = find(subsets, nextEdge.u);

int y = find(subsets, nextEdge.v);

if (x != y) {

result[e++] = nextEdge;

unionSets(subsets, x, y);

}

}

for (int j = 0; j < e; j++) {

printf("%d - %d: %d\n", result[j].u, result[j].v, result[j].weight);

}

}

int main() {

int vertices = 4;

int edgeCount = 5;

Edge edges[edgeCount];

edges[0] = (Edge){0, 1, 10};

edges[1] = (Edge){0, 2, 6};

edges[2] = (Edge){0, 3, 5};

edges[3] = (Edge){1, 3, 15};

edges[4] = (Edge){2, 3, 4};

kruskal(vertices, edgeCount, edges);

return 0;

}

**9) Job Sequencing by Greedy Approach : O(nsqr)**

#include <stdio.h>

#include <stdlib.h>

typedef struct {

char id;

int deadline;

int profit;

} Job;

int compare(const void \*a, const void \*b) {

Job \*job1 = (Job \*)a;

Job \*job2 = (Job \*)b;

return job2->profit - job1->profit;

}

void jobSequencing(Job jobs[], int n) {

qsort(jobs, n, sizeof(Job), compare);

int maxDeadline = 0;

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

if (jobs[i].deadline > maxDeadline) {

maxDeadline = jobs[i].deadline;

}

}

int result[maxDeadline];

int slot[maxDeadline];

for (int i = 0; i < maxDeadline; i++) {

slot[i] = 0;

}

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

for (int j = jobs[i].deadline - 1; j >= 0; j--) {

if (slot[j] == 0) {

result[j] = i;

slot[j] = 1;

break;

}

}

}

printf("Job sequence to maximize profit:\n");

for (int i = 0; i < maxDeadline; i++) {

if (slot[i] == 1) {

printf("%c ", jobs[result[i]].id);

}

}

printf("\n");

}

int main() {

Job jobs[] = {

{'1', 5,200},

{'2', 3,180},

{'3', 3,190},

{'4', 2,300},

{'5', 4,120},

{'6', 2,100}

};

int n = sizeof(jobs) / sizeof(jobs[0]);

jobSequencing(jobs, n);

return 0;

}

**10) Fractional Knapsack by Greedy Approach : O(nlogn)**

#include <stdio.h>

#include <stdlib.h>

typedef struct {

float weight;

float value;

float cost;

} Item;

int compare(const void \*a, const void \*b) {

float costA = ((Item \*)a)->cost;

float costB = ((Item \*)b)->cost;

return (costA < costB) - (costA > costB);

}

void getMaxValue(Item \*items, int numItems, int capacity) {

qsort(items, numItems, sizeof(Item), compare);

float totalValue = 0;

printf("\nItems added to the knapsack:\n");

for (int i = 0; i < numItems; i++) {

float itemWeight = items[i].weight;

float itemValue = items[i].value;

if (capacity >= itemWeight) {

capacity -= (int)itemWeight;

totalValue += itemValue;

printf("Item with weight %.2f and value %.2f (full item)\n", itemWeight, itemValue);

} else {

float fraction = (float)capacity / itemWeight;

totalValue += itemValue \* fraction;

printf("Item with weight %.2f and value %.2f (%.2f%% of item)\n", itemWeight, itemValue, fraction \* 100);

capacity -= (int)(itemWeight \* fraction);

break;

}

}

printf("\nTotal value in the knapsack = %.2f\n", totalValue);

}

int main() {

int numItems;

printf("Enter number of items: ");

scanf("%d", &numItems);

printf("Enter capacity of knapsack: ");

int capacity;

scanf("%d", &capacity);

Item \*items = (Item \*)malloc(numItems \* sizeof(Item));

for (int i = 0; i < numItems; i++) {

printf("Enter weight of item %d: ", i + 1);

scanf("%f", &items[i].weight); // Use %f for float input

}

for (int i = 0; i < numItems; i++) {

printf("Enter value of item %d: ", i + 1);

scanf("%f", &items[i].value);

items[i].cost = items[i].value / items[i].weight;

}

getMaxValue(items, numItems, capacity);

free(items);

return 0;

}

**11) All pair shortest path by Dynamic Programming : O(VE)**

#include <stdio.h>

#include <limits.h>

void printMatrix(int n, int matrix[][n], int infinity) {

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

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

if (matrix[i][j] == infinity) {

printf("INF\t");

} else {

printf("%d\t", matrix[i][j]);

}

}

printf("\n");

}

printf("\n");

}

void apsp(int n, int matrix[][n], int infinity) {

int i, j, k;

for (k = 0; k < n; k++) {

printf("A%d:\n", k);

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

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

if (matrix[i][k] != infinity && matrix[k][j] != infinity &&

matrix[i][j] > matrix[i][k] + matrix[k][j]) {

matrix[i][j] = matrix[i][k] + matrix[k][j];

}

}

}

printMatrix(n, matrix, infinity);

}

}

int main() {

int n, i, j, infinity;

printf("Enter size: ");

scanf("%d", &n);

printf("Enter infinity value: ");

scanf("%d", &infinity);

int matrix[n][n];

printf("Enter matrix elements (use %d for infinity):\n", infinity);

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

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

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

if (matrix[i][j] == infinity) {

matrix[i][j] = INT\_MAX;

}

}

}

printf("Initial Matrix:\n");

printMatrix(n, matrix, INT\_MAX);

apsp(n, matrix, INT\_MAX);

return 0;

}

**12) Longest Common Subsequence by Dynamic Programming : O(mn)**

#include<stdio.h>

#include<string.h>

void print\_lcs(char x[], int i, int j);

char b[100][100];

int c[100][100];

void lcs(char x[], char y[])

{

int m=strlen(x);

int n=strlen(y);

for( int i=1; i<=m;i++)

{

c[i][0]=0;

}

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

{

c[0][i]=0;

}

for( int i=1; i<=m;i++)

{

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

{

if(x[i-1]==y[j-1])

{

c[i][j]=c[i-1][j-1]+1;

b[i][j]='/';

}

else if(c[i-1][j]>c[i][j-1])

{

c[i][j]=c[i-1][j];

b[i][j]='!';

}

else

{

c[i][j]=c[i][j-1];

b[i][j]='-';

}

}

}

printf("%d\n",c[m][n]);

print\_lcs(x, m, n);

}

void print\_lcs(char x[], int i, int j) {

if (i == 0 || j == 0)

return;

if (b[i][j] == '/') {

print\_lcs(x, i - 1, j - 1);

printf("%c", x[i - 1]);

} else if (b[i][j] == '!') {

print\_lcs(x, i - 1, j);

} else {

print\_lcs(x, i, j - 1);

}

}

int main() {

char x[100], y[100];

printf("Enter sequence 1: ");

scanf("%s", x);

printf("Enter sequence 2: ");

scanf("%s", y);

lcs(x, y);

return 0;

}

**13) N-Queen by Backtracking : O(N)**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

int x[100];

int place(int k, int i, int n) {

for (int j = 1; j < k; j++) {

if (x[j] == i || abs(x[j] - i) == abs(j - k))

return 0;

}

return 1;

}

void printQueens(int n) {

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

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

if (x[i] == j) {

printf("Q ");

} else {

printf(". ");

}

}

printf("\n");

}

printf("\n");

}

void NQueens(int k, int n) {

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

if (place(k, i, n)) {

x[k] = i;

if (k == n) {

printQueens(n);

} else {

NQueens(k + 1, n);

}

}

}

}

int main() {

int n;

printf("Enter the number of queens: ");

scanf("%d", &n);

if (n <= 0) {

printf("Invalid input. Number of queens must be greater than 0.\n");

return 1;

}

printf("Possible placements of %d queens:\n", n);

NQueens(1, n);

return 0;

}

**14) Sum of Subset by Backtracking :**

#include <stdio.h>

#include <stdbool.h>

#define MAX\_SIZE 100

void subsetSum(int set[], int n, int sum, int subset[], int subSize, int totalSum, int nodeCount) {

if (totalSum == sum) {

printf("Subset found: ");

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

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

printf("\n");

return;

}

if (nodeCount >= n || totalSum > sum)

return;

subset[subSize] = set[nodeCount];

subsetSum(set, n, sum, subset, subSize + 1, totalSum + set[nodeCount], nodeCount + 1);

subsetSum(set, n, sum, subset, subSize, totalSum, nodeCount + 1);

}

int main() {

int set[MAX\_SIZE];

int sum, n;

printf("Enter the number of elements in the set: ");

scanf("%d", &n);

printf("Enter the elements of the set:\n");

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

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

printf("Enter the target sum: ");

scanf("%d", &sum);

int subset[MAX\_SIZE];

subsetSum(set, n, sum, subset, 0, 0, 0);

return 0;

}

**15) Rabin Karp Algorithm for string matching :**

**Preprocessing time complexity is Ɵ(m)**

Average and best case time complexity is Ɵ(n+m)

**Worst-case time complexity is Ɵ((n-m+1)m)**

#include <stdio.h>

#include <string.h>

#define d 256

void Rabin\_Karp\_Matching(char \*T, char \*P, int q)

{ int n = strlen(T);

int m = strlen(P);

int h = 1;

int p = 0, t0 = 0;

for (int i = 0; i < m - 1; i++)

h = (h \* d) % q;

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

{ p = (d \* p + P[i]) % q;

t0 = (d \* t0 + T[i]) % q;

}

for (int s = 0; s < n - m; s++)

{ if (p == t0)

{ int match = 1;

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

{ if (T[s + i] != P[i])

{ match = 0;

break;

}

}

if (match)

printf("Pattern occurs with shift %d\n", s);

}

if (s < n - m)

{ t0 = (d \* (t0 - T[s] \* h) + T[s + m]) % q;

if (t0 < 0)

t0 = (t0 + q);

}

}

}

int main()

{ char T[100], P[100];

int q;

printf("Enter the text: ");

fgets(T, sizeof(T), stdin);

T[strcspn(T, "\n")] = '\0';

printf("Enter the pattern: ");

fgets(P, sizeof(P), stdin);

P[strcspn(P, "\n")] = '\0';

printf("Enter a prime number (q): ");

scanf("%d", &q);

Rabin\_Karp\_Matching(T, P, q);

return 0;

}

**16) Naïve string matching algorithm :**

**Preprocessing time complexity is 0**

**Worst-case time complexity is Ɵ((n-m+1)m)**

#include <stdio.h>

#include <string.h>

int main() {

char txt1[] = "AACAABCDA";

char pat1[] = "AAB";

int patlen=strlen(pat1);

int txtlen=strlen(txt1);

int n=txtlen-patlen+1;

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

{

int j;

for(j=0;j<patlen;j++)

{

if(txt1[i+j]!=pat1[j])

{

break;

}

}

if(j==patlen)

{

printf("Pattern found at index : %d\t",i);

}

}

return 0;

}