1. **Write a code to perform Merge sort.**

#include <stdio.h>

void merge(int a[], int beg, int mid, int end)

{

int i, j, k;

int n1 = mid - beg + 1;

int n2 = end - mid;

int LeftArray[n1], RightArray[n2];

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

LeftArray[i] = a[beg + i];

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

RightArray[j] = a[mid + 1 + j];

i = 0;

j = 0;

k = beg;

while (i < n1 && j < n2)

{

if(LeftArray[i] <= RightArray[j])

{

a[k] = LeftArray[i];

i++;

}

else

{

a[k] = RightArray[j];

j++;

}

k++;

}

while (i<n1)

{

a[k] = LeftArray[i];

i++;

k++;

}

while (j<n2)

{

a[k] = RightArray[j];

j++;

k++;

}

}

void mergeSort(int a[], int beg, int end)

{

if (beg < end)

{

int mid = (beg + end) / 2;

mergeSort(a, beg, mid);

mergeSort(a, mid + 1, end);

merge(a, beg, mid, end);

}

}

void printArray(int a[], int n)

{

int i;

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

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

printf("\n");

}

int main()

{

int a[] = { 12, 31, 25, 8, 32, 17, 40, 42 };

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

printf("Before sorting array elements are - \n");

printArray(a, n);

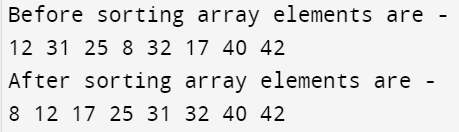
mergeSort(a, 0, n - 1);

printf("After sorting array elements are - \n");

printArray(a, n);

return 0;

}



1. **Write a code to perform Quick sort.**

#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);

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

if (arr[j] < pivot) {

i++;

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

}

}

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 a[] = { 12, 31, 25, 8, 32, 17, 40, 42 };

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

printf("Before sorting array elements are - \n");

printArray(a, n);

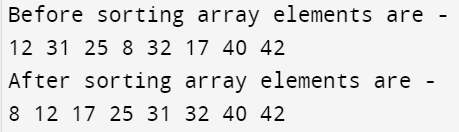
quickSort(a, 0, n - 1);

printf("After sorting array elements are - \n");

printArray(a, n);

return 0;

}



1. **Given a sorted array and a number X, search two elements of the array such that their sum is X. Expected time complexity is O(n2 ) and O(n).**

#include <stdio.h>

#include <limits.h>

#include <stdlib.h>

void printClosest(int arr[], int n, int x)

{

int res\_l, res\_r;

int temp = INT\_MAX;

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

{

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

{

if(abs(arr[i]+arr[j]-x)<temp)

{

res\_l=i;

res\_r=j;

temp=abs(arr[i]+arr[j]-x);

}

}

}

printf(" The closest pair is %d and %d ",arr[res\_l], arr[res\_r]);

}

int main()

{

int arr[] = {10, 22, 28, 29, 30, 40}, x = 54;

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

printClosest(arr, n, x);

return 0;

}



#include <stdio.h>

void find\_pair\_with\_sum\_backtracking(int arr[], int buffer[], int start, int end, int index, int target) {

if (index == 2) {

if (buffer[0] + buffer[1] == target) {

printf("Pair with sum %d: %d %d\n", target, buffer[0], buffer[1]);

}

return;

}

for (int i = start; i <= end && end - i + 1 >= 2 - index; i++) {

buffer[index] = arr[i];

find\_pair\_with\_sum\_backtracking(arr, buffer, i + 1, end, index + 1, target);

}

}

int main() {

int arr[] = {1, 2, 3, 4, 5, 6, 7};

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

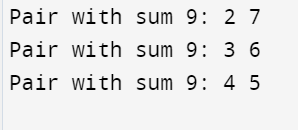
int X = 9;

int buffer[2];

find\_pair\_with\_sum\_backtracking(arr, buffer, 0, n - 1, 0, X);

return 0;

}



1. **Given a sorted array and a number x, write a function that counts the occurrences of x in the array. Expected time complexity is O(n) and O(logn).**

Time complexity O(n)

#include <stdio.h>

int countOccurrences(int arr[], int n, int x)

{

int res = 0;

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

if (x == arr[i])

res++;

return res;

}

int main()

{

int arr[] = {1, 2, 2, 2, 2, 3, 4, 7, 8, 8};

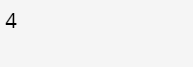
int n = sizeof(arr) / sizeof(arr[0]);

int x = 2;

printf("%d", countOccurrences(arr, n, x));

return 0;

}



Time complexity O(log n)

# include <stdio.h>

int first(int arr[], int low, int high, int x, int n)

{

if(high >= low)

{

int mid = (low + high)/2;

if( ( mid == 0 || x > arr[mid-1]) && arr[mid] == x)

return mid;

else if(x > arr[mid])

return first(arr, (mid + 1), high, x, n);

else

return first(arr, low, (mid -1), x, n);

}

return -1;

}

int last(int arr[], int low, int high, int x, int n)

{

if (high >= low)

{

int mid = (low + high)/2;

if( ( mid == n-1 || x < arr[mid+1]) && arr[mid] == x )

return mid;

else if(x < arr[mid])

return last(arr, low, (mid -1), x, n);

else

return last(arr, (mid + 1), high, x, n);

}

return -1;

}

int count(int arr[], int x, int n)

{

int i;

int j;

i = first(arr, 0, n-1, x, n);

if(i == -1)

return i;

j = last(arr, i, n-1, x, n);

return j-i+1;

}

int main()

{

int arr[] = { 1, 2, 2, 2, 2, 3, 4, 7, 8, 8 };

int x = 2;

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

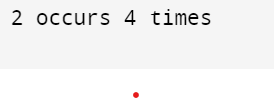
int c = count(arr, x, n);

printf(" %d occurs %d times ", x, c);

getchar();

return 0;

}



1. **Implement Binary Search using Divide and Conquer.**

#include <stdio.h>

int binarySearch(int a[], int beg, int end, int val)

{

int mid;

if(end >= beg)

{ mid = (beg + end)/2;

if(a[mid] == val)

{

return mid+1;

}

else if(a[mid] < val)

{

return binarySearch(a, mid+1, end, val);

}

else

{

return binarySearch(a, beg, mid-1, val);

}

}

return -1;

}

int main() {

int a[] = {11, 14, 25, 30, 40, 41, 52, 57, 70};

int val = 40;

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

int res = binarySearch(a, 0, n-1, val);

printf("The elements of the array are - ");

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

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

printf("\nElement to be searched is - %d", val);

if (res == -1)

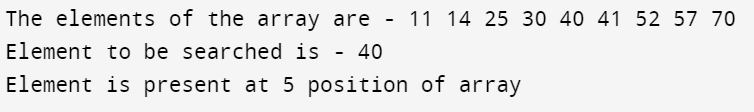
printf("\nElement is not present in the array");

else

printf("\nElement is present at %d position of array", res);

return 0;

}



1. **Implement a greedy algorithm to solve the fractional knapsack problem.**

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

struct Item {

int profit, weight;

};

static int cmp(const void\* a, const void\* b)

{

struct Item\* itemA = (struct Item\*)a;

struct Item\* itemB = (struct Item\*)b;

double r1 = (double)itemA->profit / itemA->weight;

double r2 = (double)itemB->profit / itemB->weight;

if (r1 < r2)

return 1;

else if (r1 > r2)

return -1;

return 0;

}

double fractionalKnapsack(int W, struct Item arr[], int N)

{

qsort(arr, N, sizeof(arr[0]), cmp);

double finalvalue = 0.0;

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

if (arr[i].weight <= W) {

W -= arr[i].weight;

finalvalue += arr[i].profit;

}

else {

finalvalue += arr[i].profit \* ((double)W / (double)arr[i].weight);

break;

}

}

return finalvalue;

}

int main()

{

int W = 15;

struct Item arr[] = { { 5, 1 }, { 10, 3 }, { 15, 5 }, { 7, 4 }, { 8, 1 }, { 9, 3 }, { 4, 2 } };

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

printf("Filled the bag with objects worth %lf\n", fractionalKnapsack(W, arr, N));

return 0;

}



1. **Find the largest and smallest number simultaneously in an array using Divide & Conquer Principle.**

#include<stdio.h>

#include<stdio.h>

int max, min;

int a[100];

void maxmin(int i, int j)

{

int max1, min1, mid;

if(i==j)

{

max = min = a[i];

}

else if(i == j-1)

{

if(a[i] <a[j])

{

max = a[j];

min = a[i];

}

else

{

max = a[i];

min = a[j];

}

}

else

{

mid = (i+j)/2;

maxmin(i, mid);

max1 = max; min1 = min;

maxmin(mid+1, j);

if(max <max1)

max = max1;

if(min > min1)

min = min1;

}

}

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]);

max = a[0];

min = a[0];

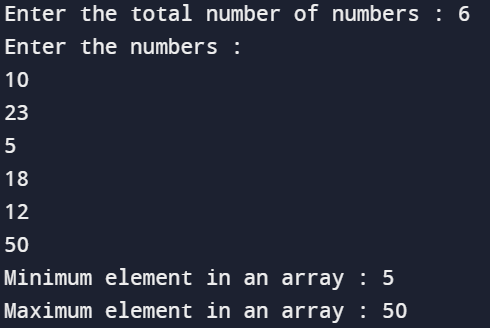
maxmin(1, num);

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

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

return 0;

}



**8. Implement the greedy algorithm to solve the problem of the Job Sequencing with deadlines.**

#include <stdbool.h>

#include <stdio.h>

#include <stdlib.h>

typedef struct Job {

char id;

int dead;

int profit;

} Job;

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

{

Job\* temp1 = (Job\*)a;

Job\* temp2 = (Job\*)b;

return (temp2->profit - temp1->profit);

}

int min(int num1, int num2)

{

return (num1 > num2) ? num2 : num1;

}

void printJobScheduling(Job arr[], int n)

{

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

int result[n];

bool slot[n];

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

slot[i] = false;

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

for (int j = min(n, arr[i].dead) - 1; j >= 0; j--) {

if (slot[j] == false) {

result[j] = i;

slot[j] = true;

break;

}

}

}

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

if (slot[i])

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

}

int main()

{

Job arr[] = { { 'a', 2, 100 },

{ 'b', 1, 19 },

{ 'c', 2, 27 },

{ 'd', 1, 25 },

{ 'e', 3, 15 } };

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

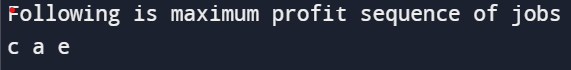
printf(

"Following is maximum profit sequence of jobs \n");

printJobScheduling(arr, n);

return 0;

}



**9. Apply Strassen’s Matrix Multiplication strategy for odd dimensional square matrices.**

#include<stdio.h>

int main(){

int z[2][2];

int i, j;

int m1, m2, m3, m4 , m5, m6, m7;

int x[2][2] = {

{12, 34},

{22, 10}

};

int y[2][2] = {

{3, 4},

{2, 1}

};

printf("The first matrix is: ");

for(i = 0; i < 2; i++) {

printf("\n");

for(j = 0; j < 2; j++)

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

}

printf("\nThe second matrix is: ");

for(i = 0; i < 2; i++) {

printf("\n");

for(j = 0; j < 2; j++)

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

}

m1= (x[0][0] + x[1][1]) \* (y[0][0] + y[1][1]);

m2= (x[1][0] + x[1][1]) \* y[0][0];

m3= x[0][0] \* (y[0][1] - y[1][1]);

m4= x[1][1] \* (y[1][0] - y[0][0]);

m5= (x[0][0] + x[0][1]) \* y[1][1];

m6= (x[1][0] - x[0][0]) \* (y[0][0]+y[0][1]);

m7= (x[0][1] - x[1][1]) \* (y[1][0]+y[1][1]);

z[0][0] = m1 + m4- m5 + m7;

z[0][1] = m3 + m5;

z[1][0] = m2 + m4;

z[1][1] = m1 - m2 + m3 + m6;

printf("\nProduct achieved using Strassen's algorithm: ");

for(i = 0; i < 2 ; i++) {

printf("\n");

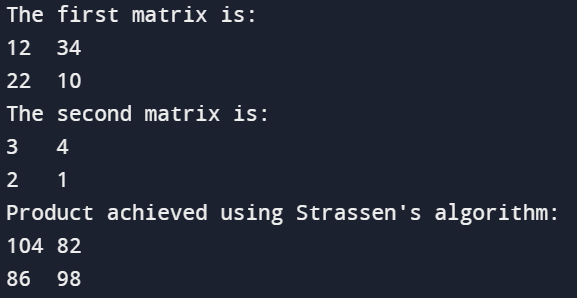
for(j = 0; j < 2; j++)

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

}

return 0;

}



int x[2][2] = {

{2, 3},

{1, 2}

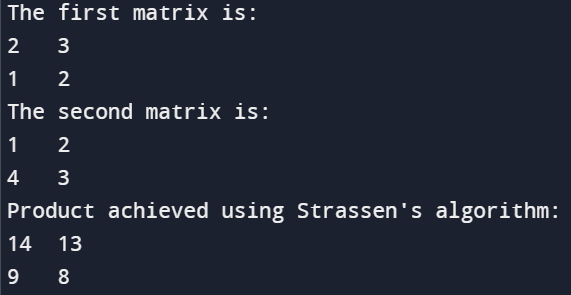
};

int y[2][2] = {

{1, 2},

{4, 3}

};



**10. KMP String Matching: Given a text txt[0..n-1] and a pattern pat[0..m-1], write a function search(char pat[], char txt[]) that prints all occurrences of pat[] in txt[]. You may assume that n > m. Text: A A B A A C A A D A A B A A B A Pattern: A A B A.**

#include <stdio.h>

#include <string.h>

void computeLPSArray(char\* pat, int M, int\* lps);

void KMPSearch(char\* pat, char\* txt)

{

int M = strlen(pat);

int N = strlen(txt);

int lps[M];

computeLPSArray(pat, M, lps);

int i = 0;

int j = 0;

while ((N - i) >= (M - j)) {

if (pat[j] == txt[i]) {

j++;

i++;

}

if (j == M) {

printf("Found pattern at index %d \n", i - j);

j = lps[j - 1];

}

else if (i < N && pat[j] != txt[i]) {

if (j != 0)

j = lps[j - 1];

else

i = i + 1;

}

}

}

void computeLPSArray(char\* pat, int M, int\* lps)

{

int len = 0;

lps[0] = 0;

int i = 1;

while (i < M) {

if (pat[i] == pat[len]) {

len++;

lps[i] = len;

i++;

}

else

{

if (len != 0) {

len = lps[len - 1];

}

else

{

lps[i] = 0;

i++;

}

}

}

}

int main()

{

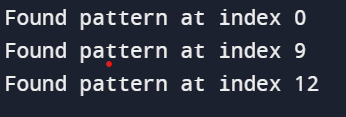
char txt[] = "AABAACAADAABAABA ";

char pat[] = "AABA";

KMPSearch(pat, txt);

return 0;

}



**11. Implement DP strategy to solve the Traveling Salesman Problem (TSP).**

#include <stdio.h>

#include <limits.h>

#define MAX 9999

int n = 4;

int distan[20][20] = {

{0, 22, 26, 30},

{30, 0, 45, 35},

{25, 45, 0, 60},

{30, 35, 40, 0}};

int DP[32][8];

int TSP(int mark, int position) {

int completed\_visit = (1 << n) - 1;

if (mark == completed\_visit) {

return distan[position][0];

}

if (DP[mark][position] != -1) {

return DP[mark][position];

}

int answer = MAX;

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

if ((mark & (1 << city)) == 0) {

int newAnswer = distan[position][city] + TSP(mark | (1 << city), city);

answer = (answer < newAnswer) ? answer : newAnswer;

}

}

return DP[mark][position] = answer;

}

int main() {

for (int i = 0; i < (1 << n); i++) {

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

DP[i][j] = -1;

}

}

printf("Minimum Distance Travelled -> %d\n", TSP(1, 0));

return 0;

}

