**Insertion Sort:**

#include <stdio.h>

void insertionSort(int arr[], int n) {

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

int key = arr[i];

int j = i - 1;

while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j];

j--;

}

arr[j + 1] = key;

}

}

void printArray(int arr[], int n) {

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

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

printf("\n");

}

int main() {

int arr[] = {5, 2, 9, 1, 5, 6};

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

printf("Original array:\n");

printArray(arr, n);

insertionSort(arr, n);

printf("Sorted array:\n");

printArray(arr, n);

return 0;

}

**Quick Sort:**

#include <stdio.h>

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

int temp = \*a;

\*a = \*b;

\*b = temp;

}

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

int pivot = arr[high];

int i = low - 1;

for (int j = low; j < high; 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 n) {

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

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

printf("\n");

}

int main() {

int arr[] = {10, 7, 8, 9, 1, 5};

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

quickSort(arr, 0, n - 1);

printArray(arr, n);

return 0;

}

**Merge Sort:**

#include <stdio.h>

#include <stdlib.h>

void merge(int arr[], int l, int m, int r) {

int n1 = m - l + 1;

int n2 = r - m;

int \*L = (int \*)malloc(n1 \* sizeof(int));

int \*R = (int \*)malloc(n2 \* sizeof(int));

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

L[i] = arr[l + i];

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

R[j] = arr[m + 1 + j];

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

while (i < n1 && j < n2) {

if (L[i] <= R[j]) arr[k++] = L[i++];

else arr[k++] = R[j++];

}

while (i < n1) arr[k++] = L[i++];

while (j < n2) arr[k++] = R[j++];

free(L);

free(R);

}

void mergeSort(int arr[], int l, int r) {

if (l < r) {

int m = l + (r - l) / 2;

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

void printArray(int arr[], int n) {

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

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

printf("\n");

}

int main() {

int arr[] = {12, 11, 13, 5, 6, 7};

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

mergeSort(arr, 0, n - 1);

printArray(arr, n);

return 0;

}

**Fractional Knapsack:**

#include <stdio.h>

#include <stdlib.h>

struct Item {

int weight, value;

};

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

double r1 = ((double)((struct Item \*)a)->value) / ((struct Item \*)a)->weight;

double r2 = ((double)((struct Item \*)b)->value) / ((struct Item \*)b)->weight;

return r2 > r1 ? 1 : -1;

}

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

qsort(arr, n, sizeof(struct Item), compare);

double totalValue = 0.0;

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

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

W -= arr[i].weight;

totalValue += arr[i].value;

} else {

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

break;

}

}

return totalValue;

}

int main() {

struct Item items[] = {{60, 100}, {100, 120}, {120, 240}};

int capacity = 50;

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

double maxValue = fractionalKnapsack(capacity, items, n);

printf("Maximum value in knapsack: %.2f\n", maxValue);

return 0;

}

**Fibbonacci top down:**

#include <stdio.h>

int fib(int n, int dp[]) {

if (n <= 1) return n;

if (dp[n] != -1) return dp[n];

dp[n] = fib(n - 1, dp) + fib(n - 2, dp);

return dp[n];

}

int main() {

int n = 10;

int dp[100];

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

printf("Fibonacci(%d): %d\n", n, fib(n, dp));

return 0;

}

**Factorial bottom up:**

#include <stdio.h>

int factorial(int n) {

int dp[100];

dp[0] = 1;

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

dp[i] = dp[i - 1] \* i;

return dp[n];

}

int main() {

int n = 5;

printf("Factorial(%d): %d\n", n, factorial(n));

return 0;

}

**LCS:**

#include <stdio.h>

#include <string.h>

int max(int a, int b) {

if (a > b)

return a;

else

return b;

}

int lcs(char a[], char b[]) {

int len1 = strlen(a);

int len2 = strlen(b);

int dp[100][100];

for (int i = 0; i <= len1; i++) {

for (int j = 0; j <= len2; j++) {

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

dp[i][j] = 0;

} else if (a[i - 1] == b[j - 1]) {

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

} else {

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

}

}

}

return dp[len1][len2];

}

int main() {

char str1[] = "abcde";

char str2[] = "ace";

int ans = lcs(str1, str2);

printf("LCS length: %d\n", ans);

return 0;

}

**Minimum coin:**

#include <stdio.h>

#define INF 1000000

int minCoins(int coins[], int n, int amount) {

int dp[1000];

for (int i = 0; i <= amount; i++) dp[i] = INF;

dp[0] = 0;

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

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

if (coins[j] <= i) {

if (dp[i - coins[j]] + 1 < dp[i]) {

dp[i] = dp[i - coins[j]] + 1;

}

}

}

}

if (dp[amount] == INF)

return -1;

else

return dp[amount];

}

int main() {

int coins[] = {1, 2, 5};

int n = 3;

int amount = 11;

int result = minCoins(coins, n, amount);

printf("Minimum coins needed: %d\n", result);

return 0;

}

**Maximum ways:**

#include <stdio.h>

int countWays(int coins[], int n, int amount) {

int dp[1000] = {0};

dp[0] = 1;

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

for (int j = coins[i]; j <= amount; j++) {

dp[j] = dp[j] + dp[j - coins[i]];

}

}

return dp[amount];

}

int main() {

int coins[] = {1, 2, 5};

int n = 3;

int amount = 5;

int ways = countWays(coins, n, amount);

printf("Maximum number of ways: %d\n", ways);

return 0;

}