**DAA DAY 2 PROGRAMS**

11. #include <stdio.h>

#include <string.h>

void copyString(char \*source, char \*destination) {

if (\*source == '\0') {

\*destination = '\0';

return;

}

\*destination = \*source;

copyString(source + 1, destination + 1);

}

int main() {

char source[100], destination[100];

printf("Enter the source string: ");

fgets(source, sizeof(source), stdin);

size\_t len = strlen(source);

if (len > 0 && source[len - 1] == '\n') {

source[len - 1] = '\0';

}

copyString(source, destination);

printf("Copied string: %s\n", destination);

return 0;

}

12. #include <stdio.h>

int binarySearch(int arr[], int size, int target) {

int left = 0, right = size - 1;

while (left <= right) {

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

if (arr[mid] == target)

return mid;

if (arr[mid] < target)

left = mid + 1;

else

right = mid - 1;

}

return -1;

}

int main() {

int arr[100], n, target, result;

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

scanf("%d", &n);

printf("Enter %d elements of the array in sorted order: ", n);

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

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

}

printf("Enter the target element to search: ");

scanf("%d", &target);

result = binarySearch(arr, n, target);

if (result != -1) {

printf("Element found at index %d\n", result);

} else {

printf("Element not found in the array\n");

}

return 0;

}

13. #include <stdio.h>

#include <string.h>

void reverseString(char \*str, int index, int length) {

if (index >= length) {

return;

}

reverseString(str, index + 1, length);

printf("%c", str[index]);

}

int main() {

char str[100];

printf("Enter a string: ");

fgets(str, sizeof(str), stdin);

size\_t len = strlen(str);

if (len > 0 && str[len - 1] == '\n') {

str[len - 1] = '\0';

}

printf("Reversed string: ");

reverseString(str, 0, strlen(str));

printf("\n");

return 0;

}

14. #include <stdio.h>

int findMin(int arr[], int size) {

int min = arr[0];

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

if (arr[i] < min) {

min = arr[i];

}

}

return min;

}

int findMax(int arr[], int size) {

int max = arr[0];

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

if (arr[i] > max) {

max = arr[i];

}

}

return max;

}

int main() {

int n;

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

scanf("%d", &n);

int arr[n];

printf("Enter the elements: ");

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

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

}

int min = findMin(arr, n);

int max = findMax(arr, n);

printf("Minimum value: %d\n", min);

printf("Maximum value: %d\n", max);

return 0;

}

15. #include <stdio.h>

void strassen(int A[2][2], int B[2][2], int C[2][2]) {

int M1 = (A[0][0] + A[1][1]) \* (B[0][0] + B[1][1]);

int M2 = (A[1][0] + A[1][1]) \* B[0][0];

int M3 = A[0][0] \* (B[0][1] - B[1][1]);

int M4 = A[1][1] \* (B[1][0] - B[0][0]);

int M5 = (A[0][0] + A[0][1]) \* B[1][1];

int M6 = (A[1][0] - A[0][0]) \* (B[0][0] + B[0][1]);

int M7 = (A[0][1] - A[1][1]) \* (B[1][0] + B[1][1]);

C[0][0] = M1 + M4 - M5 + M7;

C[0][1] = M3 + M5;

C[1][0] = M2 + M4;

C[1][1] = M1 - M2 + M3 + M6;

}

int main() {

int A[2][2] = {{1, 2}, {3, 4}};

int B[2][2] = {{5, 6}, {7, 8}};

int C[2][2];

strassen(A, B, C);

printf("Result matrix is:\n");

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

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

printf("%d ", C[i][j]);

printf("\n");

}

return 0;

}

16. #include <stdio.h>

void merge(int arr[], int left, int mid, int right) {

int n1 = mid - left + 1;

int n2 = right - mid;

int L[n1], R[n2];

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

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

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

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

int i = 0;

int j = 0;

int k = left;

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

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

} else {

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

void mergeSort(int arr[], int left, int right) {

if (left < right) {

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

mergeSort(arr, left, mid);

mergeSort(arr, mid + 1, right);

merge(arr, left, mid, right);

}

}

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

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

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

printf("\n");

}

int main() {

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

int arr\_size = sizeof(arr) / sizeof(arr[0]);

printf("Given array is \n");

printArray(arr, arr\_size);

mergeSort(arr, 0, arr\_size - 1);

printf("\nSorted array is \n");

printArray(arr, arr\_size);

return 0;

}

17#include <stdio.h>

#include <limits.h>

void findMaxMin(int arr[], int left, int right, int \*max, int \*min) {

if (left == right) {

if (arr[left] > \*max) \*max = arr[left];

if (arr[left] < \*min) \*min = arr[left];

} else if (right == left + 1) {

if (arr[left] > arr[right]) {

if (arr[left] > \*max) \*max = arr[left];

if (arr[right] < \*min) \*min = arr[right];

} else {

if (arr[right] > \*max) \*max = arr[right];

if (arr[left] < \*min) \*min = arr[left];

}

} else {

int mid = (left + right) / 2;

findMaxMin(arr, left, mid, max, min);

findMaxMin(arr, mid + 1, right, max, min);

}

}

int main() {

int arr[] = {3, 5, 1, 9, 2, 8, 4, 7, 6};

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

int max = INT\_MIN, min = INT\_MAX;

findMaxMin(arr, 0, n - 1, &max, &min);

printf("Maximum value: %d\n", max);

printf("Minimum value: %d\n", min);

return 0;

}

18

#include <stdio.h>

#include <stdbool.h>

bool isPrime(int n, int i) {

if (n <= 2)

return (n == 2) ? true : false;

if (n % i == 0)

return false;

if (i \* i > n)

return true;

return isPrime(n, i + 1);

}

void printPrimes(int n, int current) {

if (current > n)

return;

if (isPrime(current, 2))

printf("%d ", current);

printPrimes(n, current + 1);

}

int main() {

int n = 30;

printf("Prime numbers up to %d are: ", n);

printPrimes(n, 2);

return 0;

}

19#include <stdio.h>

#include <stdlib.h>

// Item structure

typedef struct { int value, weight; } Item;

// Comparison function for qsort

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

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

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

return r2 - r1;

}

// Greedy knapsack function

float knapsackGreedy(Item items[], int n, int capacity) {

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

float totalValue = 0.0;

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

if (items[i].weight <= capacity) {

capacity -= items[i].weight;

totalValue += items[i].value;

} else {

totalValue += items[i].value \* ((float)capacity / items[i].weight);

break;

}

}

return totalValue;

}

int main() {

Item items[] = {{60, 10}, {100, 20}, {120, 30}};

int capacity = 50;

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

printf("Maximum value: %.2f\n", knapsackGreedy(items, n, capacity));

return 0;

}

20. #include <stdio.h>

#include <limits.h>

#include <stdbool.h>

#define V 5 // Number of vertices

int minKey(int key[], bool mstSet[]) {

int min = INT\_MAX, min\_index;

for (int v = 0; v < V; v++)

if (!mstSet[v] && key[v] < min)

min = key[v], min\_index = v;

return min\_index;

}

void primMST(int graph[V][V]) {

int parent[V], key[V];

bool mstSet[V] = { false };

for (int i = 0; i < V; i++) key[i] = INT\_MAX;

key[0] = 0, parent[0] = -1;

for (int count = 0; count < V - 1; count++) {

int u = minKey(key, mstSet);

mstSet[u] = true;

for (int v = 0; v < V; v++)

if (graph[u][v] && !mstSet[v] && graph[u][v] < key[v])

parent[v] = u, key[v] = graph[u][v];

}

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

printf("%d - %d: %d\n", parent[i], i, graph[i][parent[i]]);

}

int main() {

int graph[V][V] = {{0, 2, 0, 6, 0}, {2, 0, 3, 8, 5}, {0, 3, 0, 0, 7}, {6, 8, 0, 0, 9}, {0, 5, 7, 9, 0}};

primMST(graph);

return 0;

}