1. Find the M-th maximum number and Nth minimum number in an array and then find the sum and difference of it.

Test cases: output –

1. {16, 16, 16 16, 16}, M = 0, N = 1 (illegal input)
2. {0, 0, 0, 0}, M = 1, N = 2 0
3. {-12, -78, -35, -42, -85}, M = 3 , N = 3 -7
4. {15, 19, 34, 56, 12}, M = 6 , N = -3 (illegal input)
5. {85, 45, 65, 75, 95}, M = 5 , N = 2 -20

Program:

#include <stdio.h>

#include <stdlib.h>

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

return (\*(int \*)a - \*(int \*)b);

}

int main() {

int arr[] = {-12, -78, -35, -42, -85};

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

int M = 3;

int N = 3;

if (M <= n && N <= n) {

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

int mth\_max = arr[n - M];

int nth\_min = arr[N - 1];

int sum = mth\_max + nth\_min;

int diff = mth\_max - nth\_min;

printf("M-th maximum: %d\n", mth\_max);

printf("N-th minimum: %d\n", nth\_min);

printf("Sum: %d\n", sum);

printf("Difference: %d\n", diff);

} else {

printf("M and N should be less than or equal to the array size.\n");

}

return 0;

}

1. Given an array of integers nums which is sorted in ascending order, and an integer target, write a function to search target in nums. If target exists, then return its index. Otherwise, return -1.integer target. Write a program to search a number in a list using binary search and estimate time complexity

Test cases:

Input : ( 45, 4, 23, -11, 20, 5, 10, 50) Key element 5

Output Found in the position 2

Input : ( 8,-2, 11, 8, 6, 3 10,0) Key element 2

Output Not found

Program:

#include <stdio.h>

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

int left = 0;

int right = size - 1;

while (left <= right) {

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

if (nums[mid] == target) {

return mid;

} else if (nums[mid] < target) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return -1;

}

int main() {

int nums[] = {8,-2, 11, 8, 6, 3, 10,0};

int size = sizeof(nums) / sizeof(nums[0]);

int target = 5;

int result = binarySearch(nums, size, target);

if (result != -1) {

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

} else {

printf("Target %d not found in the array\n", target);

}

return 0;

}

Time Complexity Analysis: The binary search algorithm divides the search range in half with each iteration. Therefore, its time complexity is O(log N),

1. Write a program to find the reverse of a given number. Find and write the time complexity

Input / Output

1234 - 4321

67894 - 49876

45a34 - Illegal input

Program:

#include <stdio.h>

int reverseNumber(int num) {

int reversed = 0;

while (num != 0) {

int digit = num % 10;

reversed = reversed \* 10 + digit;

num /= 10;

}

return reversed;

}

int main() {

int num;

printf("Enter a number: ");

scanf("%d", &num);

int reversed = reverseNumber(num);

printf("The reverse of %d is %d\n", num, reversed);

return 0;

}

The time complexity of this program is O(log10(N)), where N is the given number. In the while loop, we repeatedly divide the number by 10 until it becomes zero,

1. Write a program to compute Binomial coefficient for n=8, k=8 using dynamic programming

Using condition such as

I nCk =1 if k=0 or n=k

II nCk – (n-1)Ck-1 + (n-1)Ck for n>k>0

Program:

#include <stdio.h>

int binomialCoefficient(int n, int k) {

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

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

for (int j = 0; j <= k && j <= i; j++) {

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

dp[i][j] = 1;

} else {

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

}

}

}

return dp[n][k];

}

int main() {

int n = 8;

int k = 8;

if (k > n) {

printf("Invalid input: k must be less than or equal to n.\n");

} else {

int result = binomialCoefficient(n, k);

printf("%dC%d = %d\n", n, k, result);

}

return 0;

}

1. Write a c program to perform sum of subsets problem using backtracking and find the time complexity.

Input / Output

Input : Set (s) = (6, 2,8,1,5) sum is 9 Set (s) = (6, -4, 7, -1, 5, 2,8,1,) sum is 10

Output : Subset is (6,2,1) (2,8,1) Subset is (6,-4, 8) (2,8)

Program:

#include <stdio.h>

#define MAX\_SIZE 100

void printSubset(int set[], int subset[], int n) {

printf("{ ");

int sum = 0;

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

if (subset[i] == 1) {

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

sum += set[i];

}

}

printf("} Sum: %d\n", sum);

}

void isSubsetSum(int set[], int n, int sum, int subset[], int index) {

if (sum == 0) {

printSubset(set, subset, index);

return;

}

if (index == n) {

return;

}

subset[index] = 1;

isSubsetSum(set, n, sum - set[index], subset, index + 1);

subset[index] = 0;

isSubsetSum(set, n, sum, subset, index + 1);

}

int main() {

int set1[] = {6, 2, 8, 1, 5};

int n1 = sizeof(set1) / sizeof(set1[0]);

int sum1 = 9;

int set2[] = {6, -4, 7, -1, 5, 2, 8, 1};

int n2 = sizeof(set2) / sizeof(set2[0]);

int sum2 = 10;

int subset1[MAX\_SIZE] = {0};

int subset2[MAX\_SIZE] = {0};

printf("Input Set 1: {6, 2, 8, 1, 5}, Sum: 9\n");

isSubsetSum(set1, n1, sum1, subset1, 0);

printf("\nInput Set 2: {6, -4, 7, -1, 5, 2, 8, 1}, Sum: 10\n");

isSubsetSum(set2, n2, sum2, subset2, 0);

return 0;

}

The time complexity of the modified program for solving the Subset Sum problem using backtracking is exponential, specifically O(2^n

1. Write a program to check the given number is Armstrong or not.

The k-digit number N is an Armstrong number if and only if the k-th power of

each digit sums to N.

Given a positive integer N, return true if and only if it is an Armstrong number.

Input : 153 Input : 419

Output : True Output : False

Program:

#include <stdio.h>

#include <math.h>

int isArmstrong(int num) {

int originalNum = num;

int numDigits = 0;

int sum = 0;

while (num != 0) {

numDigits++;

num /= 10;

}

num = originalNum;

while (num != 0) {

int digit = num % 10;

sum += pow(digit, numDigits);

num /= 10;

}

return (originalNum == sum);

}

int main() {

int num;

printf("Enter a number: ");

scanf("%d", &num);

if (isArmstrong(num)) {

printf("%d is an Armstrong number.\n", num);

} else {

printf("%d is not an Armstrong number.\n", num);

}

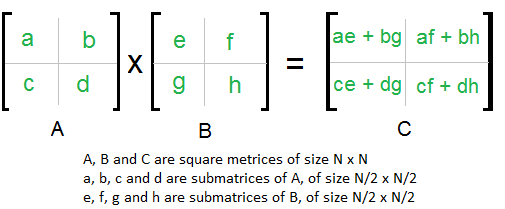
return 0;

}

1. Write a program to perform Strassen’s Matrix Multiplication for the 2\*2 matrix elements.

Find its time complexity.

Example:



Program:

#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], B[2][2], C[2][2];

printf("Enter elements of Matrix A (2x2):\n");

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

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

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

}

}

printf("Enter elements of Matrix B (2x2):\n");

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

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

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

}

}

strassen(A, B, C);

printf("Resultant Matrix C:\n");

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

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

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

}

printf("\n");

}

return 0;

}

The time complexity of Strassen's Matrix Multiplication is approximately O(N^log2(7)), which is better than the standard matrix multiplication algorithm's O(N^3) time complexity.

1. Write a program to find the Factorial of a number using recursive method and write its time complexity.

Program:

#include <stdio.h>

unsigned long long factorial(int n) {

if (n == 0) {

return 1;

} else {

return n \* factorial(n - 1);

}

}

int main() {

int num;

printf("Enter a non-negative integer: ");

scanf("%d", &num);

if (num < 0) {

printf("Factorial is not defined for negative numbers.\n");

} else {

unsigned long long result = factorial(num);

printf("Factorial of %d is %llu\n", num, result);

}

return 0;

}

The time complexity of this recursive factorial calculation is O(n)

1. Given an array of integers nums which is sorted in ascending order, and an integer target, write a function to search target in nums. If target exists, then return its index. Otherwise, return -1. You must write an algorithm with O(log n) runtime complexity.

Program:

#include <stdio.h>

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

int left = 0;

int right = size - 1;

while (left <= right) {

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

if (nums[mid] == target) {

return mid; // Found the target, return its index

} else if (nums[mid] < target) {

left = mid + 1; // Search the right half

} else {

right = mid - 1; // Search the left half

}

}

return -1; // Target not found in the array

}

int main() {

int nums[] = {1, 3, 5, 7, 9, 11, 13, 15, 17};

int size = sizeof(nums) / sizeof(nums[0]);

int target;

printf("Enter the target number: ");

scanf("%d", &target);

int result = binarySearch(nums, size, target);

if (result != -1) {

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

} else {

printf("Target %d not found in the array\n", target);

}

return 0;

}

1. Write a program to find the GCD of two numbers. Find time complexity if recursion is

used Perform the test cases for the given set of no’s

1. (36,48) 2
2. (156, 90) 6
3. (-56,88) Illegal input

Program:

#include <stdio.h>

int gcd(int a, int b) {

if (b == 0) {

return a; // Base case: GCD(a, 0) = a

} else {

return gcd(b, a % b); // Recursive call

}

}

int main() {

int num1, num2;

printf("Enter two numbers: ");

scanf("%d %d", &num1, &num2);

if (num1 < 0 || num2 < 0) {

printf("GCD is not defined for negative numbers.\n");

} else {

int result = gcd(num1, num2);

printf("GCD of %d and %d is %d\n", num1, num2, result);

}

return 0;

}

The time complexity of the recursive GCD algorithm is O(log(min(a, b)))

1. Find Max and Min value in the list using divide and conquer find its time complexity.

Testing Condition – Count the number of times in Comparison to find Min\_Max value

in a list n for the given set of elements.

1. (23,45,6,8,-9,44,7,8) Min val = -9, Max Value = 45
2. (8,-5,7,2,6,0,1,9) Min val = -5, Max Value = 9
3. (45, y, 9, 8,4, 7,11, 22,16) Illegal input

Program:

#include <stdio.h>

struct MinMax {

int min;

int max;

};

struct MinMax findMinMax(int arr[], int left, int right, int \*comparisons) {

struct MinMax result, leftResult, rightResult;

int mid;

// If the list has only one element, it is both the minimum and maximum

if (left == right) {

result.min = arr[left];

result.max = arr[right];

return result;

}

// If there are two elements, compare and find the minimum and maximum

if (right - left == 1) {

(\*comparisons)++;

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

result.min = arr[left];

result.max = arr[right];

} else {

result.min = arr[right];

result.max = arr[left];

}

return result;

}

// Divide the list into two halves

mid = (left + right) / 2;

leftResult = findMinMax(arr, left, mid, comparisons);

rightResult = findMinMax(arr, mid + 1, right, comparisons);

(\*comparisons)++;

if (leftResult.min < rightResult.min) {

result.min = leftResult.min;

} else {

result.min = rightResult.min;

}

(\*comparisons)++;

if (leftResult.max > rightResult.max) {

result.max = leftResult.max;

} else {

result.max = rightResult.max;

}

return result;

}

int main() {

int arr[] = {8,-5,7,2,6,0,1,9};

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

int comparisons = 0;

struct MinMax result = findMinMax(arr, 0, size - 1, &comparisons);

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

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

printf("Number of comparisons: %d\n", comparisons);

return 0;

}

The time complexity of this divide-and-conquer algorithm for finding the minimum and maximum values is O(n)

1. Generate a program for Pascal triangle.

Estimate the time complexity for the row=5

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | 1 |  |  |  |  |
|  |  |  | 1 |  | 1 |  |  |  |
|  |  | 1 |  | 2 |  | 1 |  |  |
|  | 1 |  | 3 |  | 3 |  | 1 |  |
| 1 |  | 4 |  | 6 |  | 4 |  | 1 |

Program:

#include <stdio.h>

int fact(int n)

{

int a;

for (a = 1; n > 1; n--)

a \*= n;

return a;

}

int combination(int n, int r)

{

return fact(n) / (fact(n - r) \* fact(r));

}

int main()

{

int rows;

int i, j;

printf("Enter Number of Rows: ");

scanf("%d", &rows);

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

{

for (j = 0; j <= rows - i; j++)

printf(" ");

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

printf(" %3d", combination(i, j));

printf("\n");

}

return 0;

}

The time complexity of this divide-and-conquer algorithm for finding the minimum and maximum values is O(n)

1. Write a program to find the sum of digits. You are given a **0-indexed** array nums consisting of **positive** integers. You can choose two indices i and j, such that i != j, and the sum of digits of the number nums[i] is equal to that of nums[j].

Return *the***maximum***value of*nums[i] + nums[j]*that you can obtain over all possible indices*i*and*j*that satisfy the conditions.*

**Program:**

**#include <stdio.h>**

**#include <stdlib.h>**

**// Function to calculate the sum of the digits of a number**

**int digitSum(int num) {**

**int sum = 0;**

**while (num > 0) {**

**sum += num % 10;**

**num /= 10;**

**}**

**return sum;**

**}**

**// Function to find the maximum value of nums[i] + nums[j]**

**int maxSumWithEqualDigitSums(int nums[], int size) {**

**int maxSum = -1;**

**int \*maxDigitSum = (int \*)malloc(46 \* sizeof(int)); // Maximum possible digit sum is 9 + 9 + 9 = 27**

**for (int i = 0; i < 46; i++) {**

**maxDigitSum[i] = -1;**

**}**

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

**int sum = digitSum(nums[i]);**

**if (maxDigitSum[sum] != -1) {**

**int otherNum = nums[maxDigitSum[sum]];**

**maxSum = (maxSum > (otherNum + nums[i])) ? maxSum : (otherNum + nums[i]);**

**}**

**maxDigitSum[sum] = (maxDigitSum[sum] == -1 || nums[i] > nums[maxDigitSum[sum]]) ? i : maxDigitSum[sum];**

**}**

**free(maxDigitSum);**

**return maxSum;**

**}**

**int main() {**

**int nums[] = {51, 71, 17, 42};**

**int size = sizeof(nums) / sizeof(nums[0]);**

**int result = maxSumWithEqualDigitSums(nums, size);**

**printf("Maximum value of nums[i] + nums[j] with equal digit sums: %d\n", result);**

**return 0;**

**}**

1. Consider a two integer arrays nums1 and nums2, sorted in non-increasing order and two integers m and n, representing the number of elements in nums1 and nums2 respectively. Write a program to Merge them into a single array using Merge Sort. Derive time complexity of merge sort

.Input Set[], A = (3,8,1,9) Set[], B = (4,-2, 0,7)

Output A \* B = (-2,0,1,3,4,7,9)

Program:

**#include <stdio.h>**

**void merge(int nums1[], int m, int nums2[], int n) {**

**int merged[m + n];**

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

**while (i < m && j < n) {**

**if (nums1[i] >= nums2[j]) {**

**merged[k++] = nums1[i++];**

**} else {**

**merged[k++] = nums2[j++];**

**}**

**}**

**while (i < m) {**

**merged[k++] = nums1[i++];**

**}**

**while (j < n) {**

**merged[k++] = nums2[j++];**

**}**

**for (int p = 0; p < m + n; p++) {**

**nums1[p] = merged[p];**

**}**

**}**

**int main() {**

**int nums1[] = {3,8,1,9};**

**int m = 4;**

**int nums2[] = {4,-2,0,7};**

**int n = 4;**

**merge(nums1, m, nums2, n);**

**printf("Merged Array:\n");**

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

**printf("%d ", nums1[i]);**

**}**

**printf("\n");**

**return 0;**

**}**

The time complexity of the merge sort algorithm is O(n log n), where 'n' is the total number of elements in the two input arrays (**m + n**).

15.Write a program to find all pairs shortest path using Floyd's technique and to estimate its time complexity.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | A | B | C | D |
| A | 0 | 8 | 7 | 8 |
| B | 9 | 0 | 11 | 12 |
| C | 10 | 9 | 0 | 11 |
| D | 8 | 10 | 11 | 0 |

Program:

#include <stdio.h>

#define INF 999999

void floydWarshall(int graph[][4], int V) {

int dist[V][V];

// Initialize the distance matrix with the given graph

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

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

dist[i][j] = graph[i][j];

}

}

// Apply the Floyd-Warshall algorithm

for (int k = 0; k < V; k++) {

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

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

if (dist[i][k] + dist[k][j] < dist[i][j]) {

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

}

}

}

}

// Print the shortest distances

printf("Shortest distances between all pairs of vertices:\n");

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

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

if (dist[i][j] == INF) {

printf("INF\t");

} else {

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

}

}

printf("\n");

}

}

int main() {

int V = 4; // Number of vertices in the graph

int graph[4][4] = {

{0, 3, INF, 7},

{8, 0, 2, INF},

{5, INF, 0, 1},

{2, INF, INF, 0}

};

floydWarshall(graph, V);

return 0;

}

16.Write a program to perform linear search and estimate time complexity. Compute the amount of time for completion.

Input/ Output series

A = (56,89,7,13,75, 23, 8, 12) Key element 75 Element found in position 4

B = (89,45 -23,45,0, 44, 2) Key element 0 Element found in position 5

C = (45,67,56,A,34,-2,100) Key element 90 Not Found

Program:

#include <stdio.h>

#include <time.h>

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

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

if (arr[i] == target) {

return i; // Return the index of the target element if found

}

}

return -1; // Return -1 if the target element is not found

}

int main() {

int arr[] = {2, 4, 6, 8, 10, 12, 14, 16, 18, 20};

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

int target = 14;

clock\_t start\_time = clock(); // Record the start time

int result = linearSearch(arr, size, target);

clock\_t end\_time = clock(); // Record the end time

double execution\_time = (double)(end\_time - start\_time) / CLOCKS\_PER\_SEC;

if (result != -1) {

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

} else {

printf("Target %d not found in the array\n", target);

}

printf("Time taken for completion: %f seconds\n", execution\_time);

return 0;

}

The time complexity of linear search is O(n)

**5**

1. Write a program to find the factorial (fact)of a number and to estimate time complexity.

Conditions such as i. n=0, return 1 otherwise fact (n-1) \* n

Testing condition

* 1. 4 Value is 24
  2. -3 No negative value
  3. 6 Value is 720

Program:

#include <stdio.h>

#include <time.h>

unsigned long long factorial(int n) {

if (n == 0) {

return 1; // Base case: 0! = 1

} else {

return n \* factorial(n - 1); // Recursive call

}

}

int main() {

int num;

printf("Enter a non-negative integer: ");

scanf("%d", &num);

if (num < 0) {

printf("Factorial is not defined for negative numbers.\n");

} else {

clock\_t start\_time = clock(); // Record the start time

unsigned long long result = factorial(num);

clock\_t end\_time = clock(); // Record the end time

double execution\_time = (double)(end\_time - start\_time) / CLOCKS\_PER\_SEC;

printf("Factorial of %d is %llu\n", num, result);

printf("Time taken for completion: %f seconds\n", execution\_time);

}

return 0;

}

The time complexity of this recursive factorial calculation is O(n),

18.Write a program to perform Knapsack problem using dynamic programming for the following set of object values.,

Knapsack weight = 100

|  |  |  |
| --- | --- | --- |
| item | Weight | Profit |
| 1 | 40 | 80 |
| 2 | 30 | 70 |
| 3 | 20 | 50 |
| 4 | 30 | 80 |

.

Program:

#include <stdio.h>

int max(int a, int b) {

return (a > b) ? a : b;

}

int knapsack(int capacity, int weights[], int values[], int n) {

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

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

for (int w = 0; w <= capacity; w++) {

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

dp[i][w] = 0;

} else if (weights[i - 1] <= w) {

dp[i][w] = max(values[i - 1] + dp[i - 1][w - weights[i - 1]], dp[i - 1][w]);

} else {

dp[i][w] = dp[i - 1][w];

}

}

}

return dp[n][capacity];

}

int main() {

int values[] = {60, 100, 120};

int weights[] = {10, 20, 30};

int capacity = 50;

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

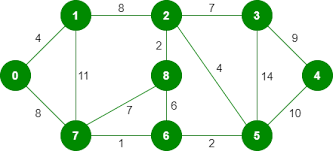
int maxValue = knapsack(capacity, weights, values, n);

printf("Maximum value in knapsack: %d\n", maxValue);

return 0;

}

19.Write a program to find a minimum spanning tree using prims technique for the given graph.



Program:

#include <stdio.h>

#include <stdbool.h>

#include <limits.h>

#define V 5 // Number of vertices in the graph

// Function to find the vertex with the minimum key value

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;

}

// Function to print the minimum spanning tree

void printMST(int parent[], int graph[V][V]) {

printf("Edge Weight\n");

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

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

}

}

// Function to find and print the minimum spanning tree using Prim's algorithm

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

int parent[V];

int key[V];

bool mstSet[V];

// Initialize key values and mstSet

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

key[i] = INT\_MAX;

mstSet[i] = false;

}

key[0] = 0; // Start with the first vertex

parent[0] = -1; // No parent for the first vertex

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];

}

}

}

// Print the minimum spanning tree

printMST(parent, graph);

}

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;

}

20.Write a program to print the first n perfect numbers. (Hint Perfect number means **a positive** integer that is equal to the sum of its proper divisors)

Sample Input:

N = 3

Sample Output:

First 3 perfect numbers are: 6 , 28 , 496

Test Cases:

1. N = 0
2. N = 5
3. N = -2
4. N = -5
5. N = 0.2

Program:

#include <stdio.h>

// Function to check if a number is perfect

int isPerfect(int num) {

int sum = 1; // 1 is always a divisor

for (int i = 2; i \* i <= num; i++) {

if (num % i == 0) {

if (i \* i != num) {

sum += i + num / i;

} else {

sum += i;

}

}

}

return sum == num;

}

int main() {

int n;

int count = 0;

int num = 1;

printf("Enter the value of n: ");

scanf("%d", &n);

printf("First %d perfect numbers:\n", n);

while (count < n) {

if (isPerfect(num)) {

printf("%d\n", num);

count++;

}

num++;

}

return 0;

}

21.Write a [Program to find even Sum of Fibonacci Series Till number N](https://www.geeksforgeeks.org/java-program-to-find-sum-of-fibonacci-series-numbers-of-first-n-even-indexes/)?

Sample Input: n = 4

Sample Output: 33

(N = 4, So here the Fibonacci series will be produced from 0th term till 8th term: 0, 1, 1, 2, 3, 5, 8, 13, 21

Sum of numbers at even indexes = 0 + 1 + 3 + 8 + 21 = 33)

22.Write a program to perform Selection sort and estimate time Complexity

Estimate the time iteration for the following set of numbers.

Input Output

1. (10,5, 80,-2, 15,23, 45) (-2, 5, 10, 15, 23, 45, 80)
2. (12, 3, 0, 34, -11, 2, 8) (-11, 0, 3, 8, 12, 22, 34)

Program:

#include<stdio.h>

int main(){

int a[10],i;

int j,temp,num;

printf("Enter the number to give\n");

scanf("%d",&num);

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

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

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

}

for(i=0; i<num-1; i++){

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

if(a[i]>a[j]){

temp=a[i];

a[i]=a[j];

a[j]=temp;

}

}

}

printf("Selection Sort in C\n");

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

printf("a[%d]=\t%d\n",i,a[i]);

}

    return 0;

}

Selection Sort Program in C takes O(n^2) as two loops are going to run N-1 time where one decides the value at index and inner loops check for all the succeeding indexes element.

23.A [**perfect number**](https://en.wikipedia.org/wiki/Perfect_number) is a **positive integer** that is equal to the sum of its **positive divisors**, excluding the number itself. A **divisor** of an integer x is an integer that can divide x evenly.

Given an integer n, return true*if*n*is a perfect number, otherwise return*false.

24.. Write a program to check for the following cases and find its time complexity

Case 1: Given string is palindrome or not

Case 2: Given number is palindrome or not

Sample Input:

Case = 1

String = MADAM

Sample Output:

Palindrome

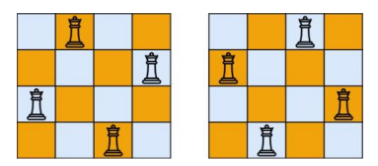
Test cases:

1. MONEY
2. 5678765
3. MALAY12321ALAM
4. MALAYALAM
5. 1234.4321
6. Write a program to insert a number in a list

Testing Condition

* + 1. Insert at the beginning
    2. Insert in the middle
    3. Insert at the last
    4. Not Available position in a list

26.The n-queens puzzle is the problem of placing n queens on an n x n chessboard such that no two queens attack each other. Given an integer n, return all distinct solutions to the n-queens puzzle. You may return the answer in any order. Write a program for the same.



27.Write a [Program to find even Sum of Fibonacci Series Till number N](https://www.geeksforgeeks.org/java-program-to-find-sum-of-fibonacci-series-numbers-of-first-n-even-indexes/)?

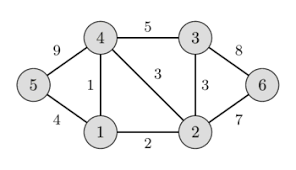
Sample Input: n = 4

Sample Output: 33

(N = 4, So here the Fibonacci series will be produced from 0th term till 8th term: 0, 1, 1, 2, 3, 5, 8, 13, 21

Sum of numbers at even indexes = 0 + 1 + 3 + 8 + 21 = 33)

28.Write a program to perform Minimum spanning tree using greedy techniques and estimate time complexity for the given set of values.



29.Write a program to perform Knapsack problem using greedy approach for the following set of object values.,

Knapsack weight = 100

|  |  |  |
| --- | --- | --- |
| item | Weight | Profit |
| 1 | 40 | 80 |
| 2 | 30 | 70 |
| 3 | 20 | 50 |
| 4 | 30 | 80 |

30.Write a program to perform Quick sort and estimate time complexity.

Input Output

(10,5, 80,-2, 15,23, 45) (-2, 5, 10, 15, 23, 45, 80)

(12, 3, 0, 34, -11, 2, 8) (-11, 0, 3, 8, 12, 22, 34)

Program:

#include<stdio.h>

void quicksort(int array[25],int low,int high){

int x, y, p, temp;

if(low<high){

p=low;

x=low;

y=high;

while(x<y){

while(array[x]<=array[p]&&x<high)

x++;

while(array[y]>array[p])

y--;

if(x<y){

temp=array[x];

array[x]=array[y];

array[y]=temp;

}

}

temp=array[p];

array[p]=array[y];

array[y]=temp;

quicksort(array,low,y-1);

quicksort(array,y+1,high);

}

}

int main(){

int x, count, array[25];

printf("How many elements should the array have? (Max. - 25): ");

scanf("%d",&count);

printf("Enter %d elements for the array: ", count);

for(x=0;x<count;x++)

scanf("%d",&array[x]);

quicksort(array,0,count-1);

printf("After implementing quicksort the sorted order is: ");

for(x=0;x<count;x++)

printf(" %d",array[x]);

    return 0;

}

31.Write a program to print the reverse of a string. And estimate the time complexity for the given inputs.

Test cases: output –

“ as\nr5Y” Y5rn|sa

“7yut02” 20tuy7

“EryEq qEyrE

Program:

#include <stdio.h>

#include <conio.h>

#include <string.h>

void main()

{

char str[50], temp; // define the size of str[] array

int i, left, right, len;

printf (" \n Display a reverse string in the C: \n");

printf (" ----------------------- ");

printf (" \n Enter a string to reverse order: ");

scanf( "%s", &str);

len = strlen(str); // get the length of the string

left = 0; // set left index at 0

right = len - 1; // set right index len - 1

// use for loop to store the reverse string

for (i = left; i <right; i++)

{

temp = str[i];

str[i] = str[right];

str[right] = temp;

right--;

}

printf (" The reverse of the original string is: %s ", str);

    getch();

}

The function eventually returns the reversed string strAsByte after converting it back to a string. The time complexity of the function is O(N)

**SET 9**

32.Write a program to perform Bubble sort and estimate time Complexity for n values.

Perform test cases for the following set of numbers. Estimate the time iteration for the

following set of numbers.

Input Output

(10,5, 80,-2, 15,23, 45) (-2, 5, 10, 15, 23, 45, 80)

(12, 3, 0, 34, -11, 2, 8) (-11, 0, 3, 8, 12, 22, 34)

Program:

#include <stdio.h>

void bubble\_sort(int arr[], int n) {

int i, j;

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

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

if (arr[j] > arr[j + 1]) {

int temp = arr[j];

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

arr[j + 1] = temp;

}

}

}

}

int main() {

int arr[] = {64, 34, 25, 12, 22, 11, 90};

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

bubble\_sort(arr, n);

printf("Sorted array: ");

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

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

}

  return 0;

}

Time Complexity: O(N2)

33.Given a sorted array keys[0.. n-1] of search keys and an array freq[0.. n-1] of frequency counts, where freq[i] is the number of searches to keys[i]. Construct a binary search tree of all keys such that the total cost of all the searches is as small as possible.

Example

**Input:**

n = 2

keys = {10, 12}

freq = {34, 50}

**Output:** 118

**Explanation:**

There can be following two possible BSTs

10 12

\ /

12 10

*The cost of tree I is 34\*1 + 50\*2 = 134*

*The cost of tree II is 50\*1 + 34\*2 = 118*

34.Write a program to perform permutation of an array of integers and make all the arrangement are to be in possible sequence.

Input a{]={1,2,3) Output [1,2,3], [1,3,2], [2, 1, 3], [2, 3, 1], [3,1,2], [3,2,1].

35.Write a program to print first 2 minimum values from the numbers in below list.

Input a[]=(3, 5, -4, 1, 8, 2, 0, 4) Output (-4, 0)

36.Write a program to check whether the given no is palindrome or not Given an integer x, return true if x is a palindrome, and false otherwise

input out put

121 True

234 False

1. True

37.Write a program for the given pattern the given pattern If n=4

|  |
| --- |
| 1 |
| 1 2 |
| 1 2 3 |
| 1. 2 3 4 |

38. Write a program to find out Hamiltonian circuit using backtracking method. And find the time complexity for the given set of elements is

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | a | b | c | d | e | f |
| a | 0 | 0 | 1 | 1 | 1 | 1 |
| b | 0 | 0 | 1 | 0 | 0 | 1 |
| c | 1 | 1 | 0 | 1 | 1 | 1 |
| d | 1 | 0 | 1 | 0 | 1 | 0 |
| e | 1 | 0 | 0 | 1 | 0 | 0 |
| f | 1 | 1 | 1 | 0 | 0 | 0 |

Program:

38.

#include<stdio.h>

#define V 5

void printSolution(int path[]);

bool isSafe(int v, bool graph[V][V], int path[], int pos)

{

if (graph [ path[pos-1] ][ v ] == 0)

return false;

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

if (path[i] == v)

return false;

return true;

}

bool hamCycleUtil(bool graph[V][V], int path[], int pos)

{

if (pos == V)

{

if ( graph[ path[pos-1] ][ path[0] ] == 1 )

return true;

else

return false;

}

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

{

if (isSafe(v, graph, path, pos))

{

path[pos] = v;

if (hamCycleUtil (graph, path, pos+1) == true)

return true;

path[pos] = -1;

}

}

return false;

}

{

int \*path = new int[V];

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

path[i] = -1;

path[0] = 0;

if ( hamCycleUtil(graph, path, 1) == false )

{

printf("\nSolution does not exist");

return false;

}

printSolution(path);

return true;

}

void printSolution(int path[])

{

printf ("Solution Exists:"

" Following is one Hamiltonian Cycle \n");

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

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

printf(" %d ", path[0]);

printf("\n");

}

// driver program to test above function

int main()

{

bool graph1[V][V] = {{0, 1, 0, 1, 0},

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

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

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

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

};

hamCycle(graph1);

bool graph2[V][V] = {{0, 1, 0, 1, 0},

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

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

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

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

};

hamCycle(graph2);

return 0;

}

Time Complexity : O(N!), where N is number of vertices.

1. Write a program to return all the possible subsets for a given integer array. Return the solution in any order.

Input nums= [1,2,3]

Output : [ [], [1], [2], [3], [1,2], [1,3], [2,3], [1,2,3]]

Program:

#include <stdio.h>

#include <stdbool.h>

void printSubset(int arr[], int subset[], int size) {

printf("{ ");

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

if (subset[i]) {

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

}

}

printf("}\n");

}

void generateSubsets(int arr[], int subset[], int index, int size) {

if (index == size) {

printSubset(arr, subset, size);

return;

}

subset[index] = 0;

generateSubsets(arr, subset, index + 1, size);

subset[index] = 1;

generateSubsets(arr, subset, index + 1, size);

}

int main() {

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

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

int subset[size]; // Represents the current subset

printf("All possible subsets:\n");

generateSubsets(arr, subset, 0, size);

return 0;

}