

**ST STEPHEN COLLEGE
UNIVERSITY OF DELHI**

COURSE :- BSC (PROG) COMPUTER SCIENCE

SEMESTER :- 3RD

YEAR :- 2025-2026

**PAPER NAME :- DESIGN AND ANALYSIS OF
ALGORITHMS**

UNIQUE PAPER CODE :- 2342572302

PRACTICAL FILE

ROLL NUMBER :- 24080582015

NAME :- VISHAL KUMAR JHA

Practical 1 :- Write a program to sort the elements of an array using Insertion Sort (The program should report the number of comparisons).

The screenshot shows a C++ development environment with the following details:

- Code Area:** The code implements Insertion Sort. It first asks for the number of elements, then reads them into an array. It then iterates through the array, moving each element to its correct position in a sorted subarray to the left. A counter variable `comparisons` keeps track of the number of comparisons made during the sort.
- Output Area:** The terminal window shows the input elements (7, 6, 5, 4, 3, 2, 1), the sorted array (1, 2, 3, 4, 5, 6, 7), and the total number of comparisons (21). The message "...Program finished with exit code 0" is also displayed.
- Code Lines 33-43:** These lines handle the final output of the sorted array and the total comparisons count.

```
#include <iostream>
using namespace std;

int main() {
    int n;
    cout << "Enter number of elements: ";
    cin >> n;

    int arr[n];
    cout << "Enter elements:\n";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    int comparisons = 0;

    // Insertion Sort
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;

        while (j >= 0) {
            comparisons++;
            if (arr[j] > key) {
                arr[j + 1] = arr[j]; // shifting
                j--;
            } else {
                break;
            }
        }
        arr[j + 1] = key;
    }

    cout << "Sorted Array: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }

    cout << "\nTotal Comparisons = " << comparisons << endl;
    return 0;
}
```

Practical 2 :- Write a program to sort the elements of an array using Merge Sort (The program should report the number of comparisons).

The screenshot shows a C++ development environment with the following details:

- Code Area:** The code implements Merge Sort. It uses a global counter `comparisons` to track the number of comparisons. The `merge` function is used to merge two halves of the array, comparing elements from both halves and swapping them if necessary.
- Output Area:** The terminal window shows the input elements (65, 34, 54, 67, 22, 76), the sorted array (22, 34, 54, 65, 67, 76), and the total number of comparisons (10). The message "...Program finished with exit code 0" is also displayed.

```
#include <iostream>
using namespace std;

long long comparisons = 0; // global counter

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 i = 0; i < n2; i++)
        R[i] = arr[mid + 1 + i];

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

    while (i < n1 && j < n2) {
        comparisons++;
        // comparison made between L[i] and R[j]

        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++;
    }
}
```

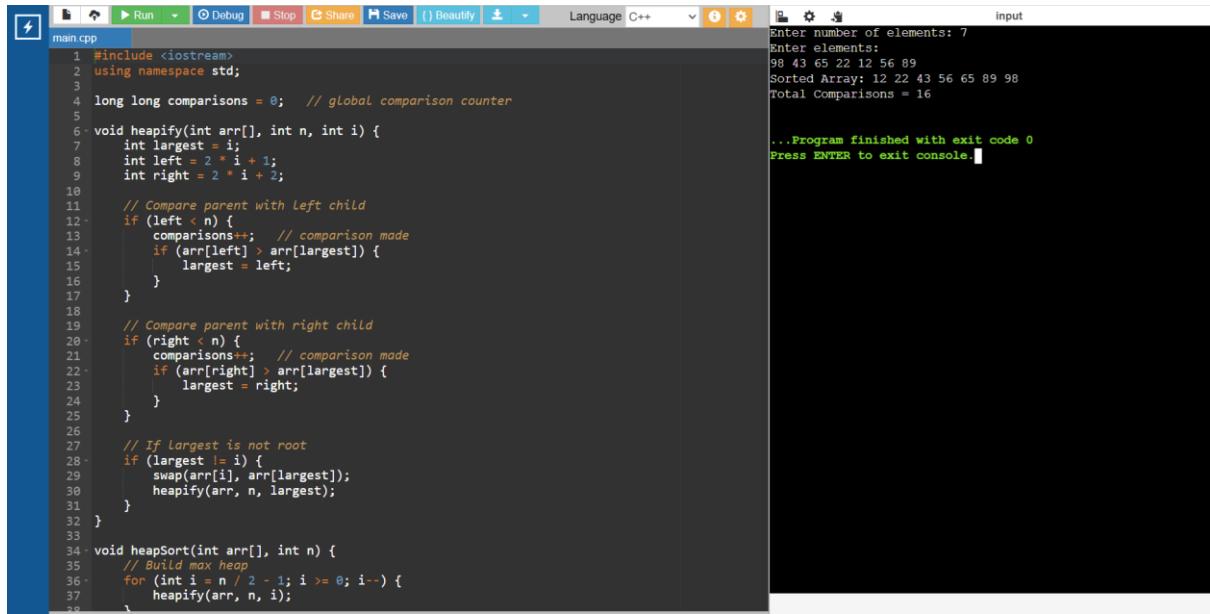


The screenshot shows a C++ IDE interface with the following details:

- Title Bar:** Language C++
- Toolbar:** Run, Debug, Stop, Share, Save, Beautify.
- Code Editor:** File main.cpp contains the merge sort code. The code is as follows:

```
38     while (j < n2) {
39         arr[k] = R[j];
40         j++;
41         k++;
42     }
43 }
45
46 void mergeSort(int arr[], int left, int right) {
47     if (left < right) {
48         int mid = (left + right) / 2;
49
50         mergeSort(arr, left, mid);
51         mergeSort(arr, mid + 1, right);
52         merge(arr, left, mid, right);
53     }
54 }
55
56 int main() {
57     int n;
58     cout << "Enter number of elements: ";
59     cin >> n;
60
61     int arr[n];
62     cout << "Enter elements:\n";
63     for (int i = 0; i < n; i++) {
64         cin >> arr[i];
65     }
66
67     mergeSort(arr, 0, n - 1);
68
69     cout << "Sorted Array: ";
70     for (int i = 0; i < n; i++) {
71         cout << arr[i] << " ";
72     }
73
74     cout << "\nTotal Comparisons = " << comparisons << endl;
75 }
```

Practical 3 :- Write a program to sort the elements of an array using Heap Sort (The program should report the number of comparisons).



The screenshot shows a C++ IDE interface with the following details:

- Title Bar:** Language C++
- Toolbar:** Run, Debug, Stop, Share, Save, Beautify.
- Code Editor:** File main.cpp contains the heap sort code. The code is as follows:

```
1 #include <iostream>
2 using namespace std;
3
4 long long comparisons = 0; // global comparison counter
5
6 void heapify(int arr[], int n, int i) {
7     int largest = i;
8     int left = 2 * i + 1;
9     int right = 2 * i + 2;
10
11    // Compare parent with left child
12    if (left < n) {
13        comparisons++; // comparison made
14        if (arr[left] > arr[largest]) {
15            largest = left;
16        }
17    }
18
19    // Compare parent with right child
20    if (right < n) {
21        comparisons++; // comparison made
22        if (arr[right] > arr[largest]) {
23            largest = right;
24        }
25    }
26
27    // If largest is not root
28    if (largest != i) {
29        swap(arr[i], arr[largest]);
30        heapify(arr, n, largest);
31    }
32}
33
34 void heapSort(int arr[], int n) {
35     // Build max heap
36     for (int i = n / 2 - 1; i >= 0; i--) {
37         heapify(arr, n, i);
38     }
39 }
```

Output Window:

```
Enter number of elements: 7
Enter elements:
98 43 65 22 12 56 89
Sorted Array: 12 22 43 56 65 89 98
Total Comparisons = 16

...Program finished with exit code 0
Press ENTER to exit console.
```

The screenshot shows a C++ code editor with the file 'main.cpp' open. The code implements a heap sort algorithm. It starts by building a max heap from the input array, then repeatedly extracts the maximum element (the root of the heap) and places it at the end of the array, followed by heapifying the reduced heap. The program then prints the sorted array and the total number of comparisons made.

```
32 J
33
34 void heapSort(int arr[], int n) {
35     // Build max heap
36     for (int i = n / 2 - 1; i >= 0; i--) {
37         heapify(arr, n, i);
38     }
39
40     // Extract elements one by one
41     for (int i = n - 1; i > 0; i--) {
42         swap(arr[0], arr[i]);
43         heapify(arr, i, 0);
44     }
45 }
46
47 int main() {
48     int n;
49     cout << "Enter number of elements: ";
50     cin >> n;
51
52     int arr[n];
53     cout << "Enter elements:\n";
54     for (int i = 0; i < n; i++) {
55         cin >> arr[i];
56     }
57
58     heapSort(arr, n);
59
60     cout << "Sorted Array: ";
61     for (int i = 0; i < n; i++) {
62         cout << arr[i] << " ";
63     }
64
65     cout << "\nTotal Comparisons = " << comparisons << endl;
66
67     return 0;
68 }
69
```

Practical 4 :- Write a program to sort the elements of an array using Quick Sort (The program should report the number of comparisons).

The screenshot shows a C++ code editor with the file 'main.cpp' open. The code implements the quick sort algorithm. It uses a helper function 'partitionArray' to rearrange the elements around a pivot, and a recursive function 'quickSort' to divide the array into smaller subproblems. The program prompts the user for the number of elements and their values, then sorts the array and prints the total number of comparisons made.

```
1 #include <iostream>
2 using namespace std;
3
4 long long comparisons = 0; // global comparison counter
5
6 int partitionArray(int arr[], int low, int high) {
7     int pivot = arr[high]; // choose last element as pivot
8     int i = low - 1;
9
10    for (int j = low; j < high; j++) {
11        comparisons++; // comparison made between arr[j] and pivot
12
13        if (arr[j] <= pivot) {
14            i++;
15            swap(arr[i], arr[j]);
16        }
17    }
18
19    swap(arr[i + 1], arr[high]);
20    return i + 1; // pivot index after partition
21 }
22
23 void quickSort(int arr[], int low, int high) {
24     if (low < high) {
25         int pi = partitionArray(arr, low, high);
26
27         quickSort(arr, low, pi - 1); // left part
28         quickSort(arr, pi + 1, high); // right part
29     }
30 }
31
32 int main() {
33     int n;
34     cout << "Enter number of elements: ";
35     cin >> n;
36 }
```

```

31
32 int main() {
33     int n;
34     cout << "Enter number of elements: ";
35     cin >> n;
36
37     int arr[n];
38     cout << "Enter elements:\n";
39     for (int i = 0; i < n; i++) {
40         cin >> arr[i];
41     }
42
43     quickSort(arr, 0, n - 1);
44
45     cout << "Sorted Array: ";
46     for (int i = 0; i < n; i++) {
47         cout << arr[i] << " ";
48     }
49
50     cout << "\nTotal Comparisons = " << comparisons << endl;
51
52     return 0;
53 }
54

```

Practical 5 :- Write a program to multiply two matrices using the Strassen's algorithm for matrix multiplication.

The screenshot shows a C++ development environment with the following details:

- Code Area:** The code is named `main.cpp`. It includes headers for `<iostream>`, declares namespaces `std`, and defines a `main` function. The function reads two 2x2 matrices from the user and performs their multiplication using Strassen's algorithm. The result is then printed to the console.
- Output Area:** The output window shows the input values and the resulting matrix. The input consists of four integers per matrix: `4 5 6 7` for the first and `9 5 2 8` for the second. The resulting matrix is `46 60` and `68 86`.
- IDE Interface:** The interface includes standard tabs like Run, Debug, Stop, Save, and Beautify. It also shows the current language as C++ and various tool icons.

```

1 #include <iostream>
2 using namespace std;
3
4 int main() {
5     int A[2][2], B[2][2], C[2][2];
6
7     cout << "Enter elements of first 2x2 matrix:\n";
8     for (int i = 0; i < 2; i++) {
9         for (int j = 0; j < 2; j++)
10            cin >> A[i][j];
11
12     cout << "Enter elements of second 2x2 matrix:\n";
13     for (int i = 0; i < 2; i++) {
14         for (int j = 0; j < 2; j++)
15            cin >> B[i][j];
16
17     // Strassen's 7 products
18     int M1 = (A[0][0] + A[1][1]) * (B[0][0] + B[1][1]);
19     int M2 = (A[0][0] + A[1][1]) * B[0][0];
20     int M3 = A[0][0] * (B[0][1] - B[1][1]);
21     int M4 = A[1][0] * (B[1][0] - B[0][0]);
22     int M5 = (A[0][0] + A[0][1]) * B[1][1];
23     int M6 = (A[0][0] - A[0][1]) * (B[0][0] + B[0][1]);
24     int M7 = (A[0][1] - A[1][1]) * (B[1][0] + B[1][1]);
25
26     // Result matrix
27     C[0][0] = M1 + M4 - M5 + M7;
28     C[0][1] = M3 + M5;
29     C[1][0] = M2 + M4;
30     C[1][1] = M1 - M2 + M3 + M6;
31
32     cout << "\nResultant Matrix (A x B using Strassen):\n";
33     for (int i = 0; i < 2; i++) {
34         for (int j = 0; j < 2; j++)
35             cout << C[i][j] << " ";
36         cout << endl;
37     }
38 }
39

```

Practical 6 :- Write a program to sort the elements of an array using Count Sort

The screenshot shows a C++ development environment with the following details:

- Title Bar:** Run, Debug, Stop, Save, Beautify, Language C++, Help.
- File:** main.cpp
- Code Area:** The code implements a counting sort algorithm. It first prompts the user for the number of elements and the elements themselves. It then finds the maximum value to determine the size of the count array. The count array stores the frequency of each element. Finally, it reconstructs the sorted array by placing each element at its correct index based on the count array values.
- Output Area:** The output shows the user input and the resulting sorted array. The user enters "Enter number of elements: 7", "Enter elements: 43 56 98 21 34 67 2", and the sorted array is printed as "Sorted Array: 2 21 34 43 56 67 98".
- Bottom Status:** "...Program finished with exit code 0 Press ENTER to exit console."

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5     int n;
6     cout << "Enter number of elements: ";
7     cin >> n;
8
9     int arr[n];
10    cout << "Enter elements:\n";
11    for (int i = 0; i < n; i++)
12        cin >> arr[i];
13
14    // Find maximum element
15    int maxVal = arr[0];
16    for (int i = 1; i < n; i++)
17        if (arr[i] > maxVal)
18            maxVal = arr[i];
19
20    // Create count array
21    int count[maxVal + 1] = {0};
22
23    // Store frequency of each element
24    for (int i = 0; i < n; i++)
25        count[arr[i]]++;
26
27    // Reconstruct the sorted array
28    int index = 0;
29    for (int i = 0; i <= maxVal; i++) {
30        while (count[i] > 0) {
31            arr[index++] = i;
32            count[i]--;
33        }
34    }
35
36    // Output sorted array
37    cout << "Sorted Array: ";
38    for (int i = 0; i < n; i++)
39        cout << arr[i] << " ";
40
41    cout << endl;
42    return 0;
43 }
44
```

Practical 7 :- Display the data stored in a given graph using the Breadth-First Search algorithm

The screenshot shows a C++ development environment with the following details:

- File:** main.cpp
- Code Content:** A C++ program implementing Breadth-First Search (BFS). The code includes imports for `<iostream>`, `<queue>`, and `<vector>`. It defines a `BFS` function that takes a start vertex, a graph represented as a vector of vectors of integers, and the number of vertices. The `main` function reads the number of vertices and edges from the user, initializes the graph, and performs the BFS traversal.
- Output:** The terminal window shows the execution of the program. It prompts for the number of vertices (5), the number of edges (6), and the edges themselves (0 1, 0 2, 1 3, 1 4, 2 3, 3 4). It then asks for the starting vertex (0) and prints the BFS Traversal path (0 1 2 3 4).
- Console:** The bottom right corner shows the message "...Program finished with exit code 0 Press ENTER to exit console."

Practical 8 :- Display the data stored in a given graph using the Depth-First Search algorithm.

```

main.cpp
1 #include <iostream>
2 #include <vector>
3 using namespace std;
4
5 void DFS(int node, vector<vector<int>>& graph, vector<bool>& visited) {
6     visited[node] = true;
7     cout << node << " ";
8
9     for (int adj : graph[node]) {
10         if (!visited[adj]) {
11             DFS(adj, graph, visited);
12         }
13     }
14 }
15
16 int main() {
17     int n, edges;
18     cout << "Enter number of vertices: ";
19     cin >> n;
20
21     cout << "Enter number of edges: ";
22     cin >> edges;
23
24     vector<vector<int>> graph(n);
25     cout << "Enter edges (u v):\n";
26
27     for (int i = 0; i < edges; i++) {
28         int u, v;
29         cin >> u >> v;
30         graph[u].push_back(v);
31         graph[v].push_back(u); // undirected graph
32     }
33
34     int start;
35     cout << "Enter starting vertex: ";
36     cin >> start;
37 }

```

```

55
34     int start;
35     cout << "Enter starting vertex: ";
36     cin >> start;
37
38     vector<bool> visited(n, false);
39
40     cout << "DFS Traversal: ";
41     DFS(start, graph, visited);
42
43     return 0;
44 }
45

```

Practical 9 :- Write a program to determine a minimum spanning tree of a graph using the Prim's algorithm.

```

main.cpp
1 #include <iostream>
2 using namespace std;
3
4 #define INF 999999
5
6 int main() {
7     int n;
8     cout << "Enter number of vertices: ";
9     cin >> n;
10
11    int graph[20][20];
12    cout << "Enter adjacency matrix (0 for no edge):\n";
13
14    for (int i = 0; i < n; i++) {
15        for (int j = 0; j < n; j++) {
16            cin >> graph[i][j];
17            if (graph[i][j] == 0 && i != j)
18                graph[i][j] = INF; // treat 0 as no edge
19        }
20    }
21
22    int visited[20] = {0};
23    visited[0] = 1; // start from vertex 0
24
25    int edges = 0;
26    int minCost = 0;
27
28    cout << "\nEdges in Minimum Spanning Tree:\n";
29
30    while (edges < n - 1) {
31        int u = -1, v = -1, min = INF;
32
33        // Find minimum edge
34        for (int i = 0; i < n; i++) {
35            if (visited[i]) {
36                for (int j = 0; j < n; j++) {
37                    if (!visited[j] && graph[i][j] < min) {

```

```

29
30     while (edges < n - 1) {
31         int u = -1, v = -1, min = INF;
32
33         // Find minimum edge
34         for (int i = 0; i < n; i++) {
35             if (visited[i]) {
36                 for (int j = 0; j < n; j++) {
37                     if (!visited[j] && graph[i][j] < min) {
38                         min = graph[i][j];
39                         u = i;
40                         v = j;
41                     }
42                 }
43             }
44         }
45
46         cout << u << " - " << v << " : " << graph[u][v] << endl;
47         minCost += graph[u][v];
48         visited[v] = 1;
49         edges++;
50     }
51
52     cout << "\nMinimum Cost = " << minCost << endl;
53
54     return 0;
55 }
56

```

Practical 10 :- Write a program to determine the shortest path from a given node s to the other nodes of a graph using the Dijkstra's algorithm.

The screenshot shows a C++ development environment with the code for Dijkstra's algorithm in the main.cpp file. The code reads the number of vertices, the adjacency matrix, and the source vertex, then calculates the shortest distances to all other vertices. The output window shows the input values and the resulting shortest distances from the source vertex.

```

main.cpp
1 #include <iostream>
2 using namespace std;
3
4 #define INF 999999
5
6 int main() {
7     int n;
8     cout << "Enter number of vertices: ";
9     cin >> n;
10
11    int graph[20][20];
12    cout << "Enter adjacency matrix (0 for no edge):\n";
13
14    for (int i = 0; i < n; i++) {
15        for (int j = 0; j < n; j++) {
16            cin >> graph[i][j];
17            if (graph[i][j] == 0 && i != j)
18                graph[i][j] = INF; // treat 0 as no connection
19        }
20    }
21
22    int src;
23    cout << "Enter source vertex: ";
24    cin >> src;
25
26    int dist[20]; // shortest distance
27    int visited[20]; // visited nodes
28
29    // Initialization
30    for (int i = 0; i < n; i++) {
31        dist[i] = graph[src][i];
32        visited[i] = 0;
33    }
34    dist[src] = 0;
35    visited[src] = 1;
36

```

Output:

```

Enter number of vertices: 5
Enter adjacency matrix (0 for no edge):
0 10 0 30 100
10 0 50 0 0
0 50 0 20 10
30 0 20 0 60
100 0 10 60 0
Enter source vertex: 0
Shortest distances from source vertex 0:
To vertex 0 = 0
To vertex 1 = 10
To vertex 2 = 50
To vertex 3 = 30
To vertex 4 = 60
...Program finished with exit code 0
Press ENTER to exit console.

```

```

36     // Dijkstra's main loop
37     for (int count = 1; count < n; count++) {
38         int minDist = INF, u = -1;
39
40         // Pick unvisited vertex with smallest distance
41         for (int i = 0; i < n; i++) {
42             if (!visited[i] && dist[i] < minDist) {
43                 minDist = dist[i];
44                 u = i;
45             }
46         }
47     }
48
49     visited[u] = 1;
50
51     // Update distances
52     for (int v = 0; v < n; v++) {
53         if (!visited[v] && dist[u] + graph[u][v] < dist[v]) {
54             dist[v] = dist[u] + graph[u][v];
55         }
56     }
57 }
58
59 // Output
60 cout << "\nShortest distances from source vertex " << src << ":\n";
61 for (int i = 0; i < n; i++) {
62     cout << "To vertex " << i << " = " << dist[i] << endl;
63 }
64
65 return 0;
66 }
67

```

Practical 11 :- Write a program to solve the 0-1 knapsack problem using Dynamic Programming.

The screenshot shows a C++ development environment with the following details:

- Code Editor:** The code is named `main.cpp`. It includes input prompts for the number of items, weights, profits, and knapsack capacity, followed by a dynamic programming table build and a final profit output.
- Output Window:** The right side of the interface displays the program's run output. It shows the user input and the resulting maximum profit.
- Bottom Panel:** A code editor window at the bottom contains the concluding part of the code, which prints the maximum profit and returns 0.

```

main.cpp
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5     int n, W;
6     cout << "Enter number of items: ";
7     cin >> n;
8
9     int weight[n], profit[n];
10    cout << "Enter weights of items:\n";
11    for (int i = 0; i < n; i++)
12        cin >> weight[i];
13
14    cout << "Enter profits of items:\n";
15    for (int i = 0; i < n; i++)
16        cin >> profit[i];
17
18    cout << "Enter capacity of knapsack: ";
19    cin >> W;
20
21    int dp[n + 1][W + 1];
22
23    // Build DP table
24    for (int i = 0; i <= n; i++) {
25        for (int w = 0; w <= W; w++) {
26            if (i == 0 || w == 0)
27                dp[i][w] = 0;
28            else if (weight[i - 1] <= w)
29                dp[i][w] = max(profit[i - 1] + dp[i - 1][w - weight[i - 1]],
30                               dp[i - 1][w]);
31            else
32                dp[i][w] = dp[i - 1][w];
33        }
34    }
35
36    cout << "\nMaximum Profit = " << dp[n][W] << endl;
37
38    return 0;
39 }

```