DSA with C++ - PBL Question Bank Answers

1. Bubble Sort

A warehouse system stores package IDs in the order they arrive. To prepare for dispatch, the IDs must be sorted in ascending order.
Write a program using Bubble Sort to arrange the following IDs: [5, 4, 3, 2, 1]

Code:

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

int main() {
  int arr[] = {5, 4, 3, 2, 1};
  int n = 5;

  for (int i = 0; i < n - 1; i++) {
    for (int j = 0; j < n - i - 1; j++) {
      if (arr[j] > arr[j + 1]) {
        swap(arr[j], arr[j + 1]);
      }
    }
  }

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

Explanation: Compares adjacent elements and swaps if they are out of order.
Output: 1 2 3 4 5

2. Insertion Sort

Code:

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

int main() {
  int arr[] = {5, 4, 3, 2, 1};
  int n = 5;

  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;
  }

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

Explanation: Inserts each element into its correct position like sorting playing cards.
Output: 1 2 3 4 5

3. Selection Sort

Code:

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

int main() {
  int arr[] = {5, 4, 3, 2, 1};
  int n = 5;

  for (int i = 0; i < n - 1; i++) {
    int minIndex = i;
    for (int j = i + 1; j < n; j++)
      if (arr[j] < arr[minIndex])
        minIndex = j;
    swap(arr[i], arr[minIndex]);
  }

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

Explanation: Finds the smallest element and places it at the start.
Output: 1 2 3 4 5

4. Linked List

Code:

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

struct Node {
```

```
    int data;
    Node* next;
};

int main() {
    Node* head = new Node{111, nullptr};
    head->next = new Node{123, nullptr};
    head->next->next = new Node{124, nullptr};

    Node* temp = head;
    cout << "Patient IDs: ";
    while (temp != nullptr) {
        cout << temp->data << " -> ";
        temp = temp->next;
    }
    cout << "NULL";
}
```

Explanation: Creates and displays linked list nodes.
Output: 111 -> 123 -> 124 -> NULL

5. Graph – Adjacency List & Matrix

Code:

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

int main() {
    int V = 4;
    vector<int> adj[4];

    adj[0] = {1, 2};
    adj[1] = {0, 3};
    adj[2] = {0, 3};
    adj[3] = {1, 2};

    cout << "Adjacency List:\n";
    for (int i = 0; i < V; i++) {
        cout << i << " -> ";
        for (int x : adj[i]) cout << x << " ";
        cout << endl;
    }

    cout << "\nAdjacency Matrix:\n";
```

```
  int matrix[4][4] = {{0,1,1,0},{1,0,0,1},{1,0,0,1},{0,1,1,0}};
  for (int i = 0; i < V; i++) {
    for (int j = 0; j < V; j++)
      cout << matrix[i][j] << " ";
    cout << endl;
  }
}
```

Explanation: Shows both adjacency list and matrix representations.

6. Binary Tree

Code:

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

struct Node {
  int data;
  Node* left;
  Node* right;
};

Node* newNode(int data) {
  Node* node = new Node;
  node->data = data;
  node->left = node->right = nullptr;
  return node;
}

void inorder(Node* root) {
  if (root == nullptr) return;
  inorder(root->left);
  cout << root->data << " ";
  inorder(root->right);
}

int main() {
  Node* root = newNode(50);
  root->left = newNode(30);
  root->right = newNode(70);
  root->left->left = newNode(20);
  root->left->right = newNode(40);
  root->right->left = newNode(60);

  cout << "Inorder traversal: ";
```

```
    inorder(root);
}
```
Explanation: Builds a binary tree and prints in inorder traversal.
Output: 20 30 40 50 60 70

7. Hashing

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

int main() {
    int table_size = 3;
    int hashTable[3] = {-1, -1, -1};
    int keys[] = {1, 2, 3, 4};

    for (int key : keys) {
        int index = key % table_size;
        while (hashTable[index] != -1)
            index = (index + 1) % table_size;
        hashTable[index] = key;
    }

    cout << "Hash Table:\n";
    for (int i = 0; i < table_size; i++)
        cout << i << " -> " << hashTable[i] << endl;
}
```
Explanation: Uses modulo hashing and linear probing to resolve collisions.

8. Graph – Adjacency Matrix

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

int main() {
    int V = 4;
    int graph[4][4] = {
        {0, 1, 1, 0},
        {1, 0, 1, 1},
        {1, 1, 0, 1},
        {0, 1, 1, 0}
    };

    cout << "Adjacency Matrix:\n";
    for (int i = 0; i < V; i++) {
```

```
      for (int j = 0; j < V; j++)
         cout << graph[i][j] << " ";
      cout << endl;
   }
}
```

Explanation: Displays adjacency matrix for given graph.

DSA with C++ - Sorting in Descending Order

1. Bubble Sort (Descending Order)

Code:

```
#include <iostream>
```

```
using namespace std;

int main() {
   int arr[] = {5, 4, 3, 2, 1};
   int n = 5;

   for (int i = 0; i < n - 1; i++) {
      for (int j = 0; j < n - i - 1; j++) {
         if (arr[j] < arr[j + 1]) { // changed sign for descending
            swap(arr[j], arr[j + 1]);
         }
      }
   }

   cout << "Sorted IDs in Descending Order: ";
   for (int i = 0; i < n; i++)
      cout << arr[i] << " ";
}
```

Explanation: Compares adjacent elements and swaps if the first is smaller.
Output: 5 4 3 2 1

2. Insertion Sort (Descending Order)

Code:

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

int main() {
   int arr[] = {5, 4, 3, 2, 1};
   int n = 5;

   for (int i = 1; i < n; i++) {
      int key = arr[i];
      int j = i - 1;
      while (j >= 0 && arr[j] < key) { // changed sign for descending
         arr[j + 1] = arr[j];
         j--;
      }
      arr[j + 1] = key;
   }

   cout << "Sorted IDs in Descending Order: ";
   for (int i = 0; i < n; i++)
      cout << arr[i] << " ";
```

```
}
```
Explanation: Inserts elements so that largest come first.
Output: 5 4 3 2 1

3. Selection Sort (Descending Order)

Code:

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

int main() {
   int arr[] = {5, 4, 3, 2, 1};
   int n = 5;

   for (int i = 0; i < n - 1; i++) {
      int maxIndex = i;
      for (int j = i + 1; j < n; j++)
         if (arr[j] > arr[maxIndex]) // changed sign for descending
            maxIndex = j;
      swap(arr[i], arr[maxIndex]);
   }

   cout << "Sorted IDs in Descending Order: ";
   for (int i = 0; i < n; i++)
      cout << arr[i] << " ";
}
```

Explanation: Finds the largest element and places it at the beginning.
Output: 5 4 3 2 1