Section A

Q1

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

int main() {
    int n = 20;
    int wallets[20];

    // Fill wallets: 20, 30, 40, ..., 210
    for (int i = 0; i < n; i++) {
        wallets[i] = 20 + (i * 10);
    }

    // Insertion Sort
    for (int i = 1; i < n; i++) {
        int key = wallets[i];
        int j = i - 1;
        while (j >= 0 && wallets[j] > key) {
            wallets[j + 1] = wallets[j];
            j--;
        }
        wallets[j + 1] = key;
    }

    // Print result
    cout << "Sorted wallets: ";
    for (int i = 0; i < n; i++) {
        cout << wallets[i] << " ";
    }
```

```
    cout << endl;

    return 0;
}
```

Q2

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

int main() {
    int wallets[10] = {2, 0, 3, 2, 1, 4, 0, 3, 6, 2};
    int n = 10;
    int maxVal = 6;

    // Step 1: Create count array
    int count[7] = {0}; // from 0 to 6

    // Step 2: Count frequency of each wallet value
    for (int i = 0; i < n; i++) {
        count[wallets[i]]++;
    }

    // Step 3: Rebuild sorted array
    int index = 0;
    for (int i = 0; i <= maxVal; i++) {
        while (count[i] > 0) {
            wallets[index++] = i;
            count[i]--;
        }
    }
```

```
    // Step 4: Print sorted wallets
    cout << "Sorted wallets: ";
    for (int i = 0; i < n; i++) {
        cout << wallets[i] << " ";
    }
    cout << endl;

    return 0;
}
```

Q3

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

// Partition function
int partition(int arr[], int low, int high) {
    int pivot = arr[high]; // choose last element as pivot
    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;
}
```

```
// Quick Sort function
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);  // sort left part
        quickSort(arr, pi + 1, high); // sort right part
    }
}

int main() {
    int scores[12] = {45, 12, 78, 34, 23, 89, 67, 11, 90, 54, 32, 76};
    int n = 12;

    quickSort(scores, 0, n - 1);

    cout << "Scores in ascending order: ";
    for (int i = 0; i < n; i++) {
        cout << scores[i] << " ";
    }
    cout << endl;

    return 0;
}
Q4#include <iostream>
using namespace std;

// Partition function
int partition(int arr[], int low, int high) {
    int pivot = arr[high]; // choose last element as pivot
```

```
    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;
}

// Quick Sort function
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);  // left part
        quickSort(arr, pi + 1, high); // right part
    }
}

int main() {
    int deadlines[10] = {25, 12, 45, 7, 30, 18, 40, 22, 10, 35};
    int n = 10;

    quickSort(deadlines, 0, n - 1);

    cout << "Deadlines in ascending order: ";
```

```
    for (int i = 0; i < n; i++) {
        cout << deadlines[i] << " ";
    }
    cout << endl;

    return 0;
}
Q5#include <iostream>
using namespace std;

// Binary Search for descending array
bool binarySearch(double arr[], int n, double key) {
    int low = 0, high = n - 1;
    while (low <= high) {
        int mid = (low + high) / 2;

        if (arr[mid] == key) return true;
        else if (arr[mid] < key) high = mid - 1;  // go left
        else low = mid + 1;                       // go right
    }
    return false;
}

int main() {
    int n = 50;
    double A; // area of first square
    cout << "Enter area of first square: ";
    cin >> A;
```

```
    double areas[50];
    areas[0] = A;

    // generate areas without <cmath>
    for (int i = 1; i < n; i++) {
        areas[i] = areas[i - 1] / 2.0;
    }

    double searchArea;
    cout << "Enter area to search: ";
    cin >> searchArea;

    if (binarySearch(areas, n, searchArea))
        cout << "Area " << searchArea << " is present." << endl;
    else
        cout << "Area " << searchArea << " is NOT present." << endl;

    return 0;
}
```

Q6

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

// Node structure
struct Player {
    string name;
    Player* next;
};

// Function to add a player at the end
```

```
void addPlayer(Player*& head, string playerName) {
    Player* newPlayer = new Player;
    newPlayer->name = playerName;
    newPlayer->next = nullptr;

    if (head == nullptr) {
        head = newPlayer;
    } else {
        Player* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newPlayer;
    }
}

// Function to simulate introductions
void meetPlayers(Player* head) {
    Player* temp = head;
    while (temp != nullptr) {
        cout << "Chief Guest meets " << temp->name << endl;
        temp = temp->next;
    }
}

int main() {
    Player* head = nullptr;

    // Create linked list of players
```

```
    addPlayer(head, "Player 1");
    addPlayer(head, "Player 2");
    addPlayer(head, "Player 3");
    addPlayer(head, "Player 4");
    addPlayer(head, "Player 5");

    // Chief guest meets players one by one
    meetPlayers(head);

    return 0;
}
Q7#include <iostream>
using namespace std;

// Node structure
struct Stop {
    string name;
    Stop* next;
    Stop* prev;
};

// Function to add a stop at the end
void addStop(Stop*& head, string stopName) {
    Stop* newStop = new Stop;
    newStop->name = stopName;
    newStop->next = nullptr;
    newStop->prev = nullptr;

    if (head == nullptr) {
```

```
        head = newStop;
    } else {
        Stop* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newStop;
        newStop->prev = temp;
    }
}

// Function to show onward journey (A → D)
void onwardJourney(Stop* head) {
    cout << "Onward Journey: ";
    Stop* temp = head;
    Stop* last = nullptr;

    while (temp != nullptr) {
        cout << temp->name << " ";
        last = temp; // keep track of last stop
        temp = temp->next;
    }
    cout << endl;
}

// Function to show return journey (D → A)
void returnJourney(Stop* tail) {
    cout << "Return Journey: ";
    Stop* temp = tail;
```

```
    while (temp != nullptr) {
        cout << temp->name << " ";
        temp = temp->prev;
    }
    cout << endl;
}

int main() {
    Stop* head = nullptr;

    // Add bus stops
    addStop(head, "Stop A");
    addStop(head, "Stop B");
    addStop(head, "Stop C");
    addStop(head, "Stop D");

    // Display journeys
    onwardJourney(head);

    // Find tail (last stop) for return journey
    Stop* tail = head;
    while (tail->next != nullptr) {
        tail = tail->next;
    }

    returnJourney(tail);

    return 0;
```

```
}
Q8#include <iostream>
using namespace std;

int main() {
   // Dalta Gang: 4x2 matrix
   int dalta[4][2] = {
      {5, 2},
      {3, 4},
      {1, 6},
      {2, 3}
   };

   // Malta Gang: 2x3 matrix
   int malta[2][3] = {
      {1, 2, 3},
      {4, 5, 6}
   };

   // Display Dalta Gang matrix
   cout << "Dalta Gang matrix (4x2):\n";
   for (int i = 0; i < 4; i++) {
      for (int j = 0; j < 2; j++) {
         cout << dalta[i][j] << " ";
      }
      cout << endl;
   }
   cout << endl;
```

```
// Display Malta Gang matrix
cout << "Malta Gang matrix (2x3):\n";
for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 3; j++) {
        cout << malta[i][j] << " ";
    }
    cout << endl;
}
cout << endl;

// Multiply Dalta (4x2) × Malta (2x3) → result 4x3
int result[4][3] = {0};

for (int i = 0; i < 4; i++) {
    for (int j = 0; j < 3; j++) {
        for (int k = 0; k < 2; k++) {
            result[i][j] += dalta[i][k] * malta[k][j];
        }
    }
}

// Display result
cout << "Result of Dalta x Malta (4x3):\n";
for (int i = 0; i < 4; i++) {
    for (int j = 0; j < 3; j++) {
        cout << result[i][j] << " ";
    }
    cout << endl;
}
```

```
    return 0;
}
```