



MENEDŻERSKA AKADEMIA  
NAUK STOSOWANYCH  
W WARSZAWIE

## Programming languages and paradigms

Laboratory 01: Elementary Data Structures

CREATED BY:

Othmane Moutaouakkil

76871

54 DPH – Computer Engineering

PROFESSOR:

Kumar Nalinaksh

## Table of Contents

Programming languages and paradigms .....	1
CREATED BY: .....	1
PROFESSOR:.....	1
Task 1: Arrays and Structures .....	3
Task 2: Stacks and Queues .....	5
Task 3: Linked Lists .....	8
Task 4: Trees .....	12
Task 5: Graphs.....	15
Task 6: Sorting .....	19
Task 7: Hashing .....	42
Task 8: Priority Queues.....	46
Task 9: Efficient Binary Search Trees .....	50
Task 10: Multiway Search Trees .....	55
Task 11: Digital Search Structures.....	60

## Task 1: Arrays and Structures

Scenario: The university administration requires a systematic record-keeping system for student information, including roll numbers, names, and academic marks.

```
// Task No. 1: Arrays and Structures

#include <stdio.h>
#include <string.h>
#define MAX_STUDENTS 100

// Step 1: Define Structure (Student)
struct Student {
    int rollNumber;
    char name[50];
    float marks;
};

// Step 2: Declare Array
struct Student students[MAX_STUDENTS];
int numStudents = 0;

// Step 3: Implement Functions
// Function to add a new student
void addStudent(int rollNumber, char name[], float marks) {
    if (numStudents < MAX_STUDENTS) {
        students[numStudents].rollNumber = rollNumber;
        strcpy(students[numStudents].name, name);
        students[numStudents].marks = marks;
        numStudents++;
        printf("Student added successfully.\n");
    } else {
        printf("Error: Maximum number of students reached.\n");
    }
}

// Function to update an existing student
void updateStudent(int rollNumber, char name[], float marks) {
    int found = 0;
    for (int i = 0; i < numStudents; i++) {
        if (students[i].rollNumber == rollNumber) {
            strcpy(students[i].name, name);
            students[i].marks = marks;
            found = 1;
            printf("Student record updated successfully.\n");
            break;
        }
    }
}
```

```

    if (!found) {
        printf("Error: Student with roll number %d not found.\n", rollNumber);
    }
}

// Function to display all student records
void displayStudents() {
    if (numStudents == 0) {
        printf("No student records found.\n");
    } else {
        printf("Student Records:\n");
        for (int i = 0; i < numStudents; i++) {
            printf("Roll Number: %d, Name: %s, Marks: %.2f\n", students[i].rollNumber, students[i].name, students[i].marks);
        }
    }
}

int main() {
    // Add some students
    addStudent(1, "John Dutton", 85.5);
    addStudent(2, "Rip Wheeler", 92.0);
    addStudent(3, "Ross Corbin", 78.2);

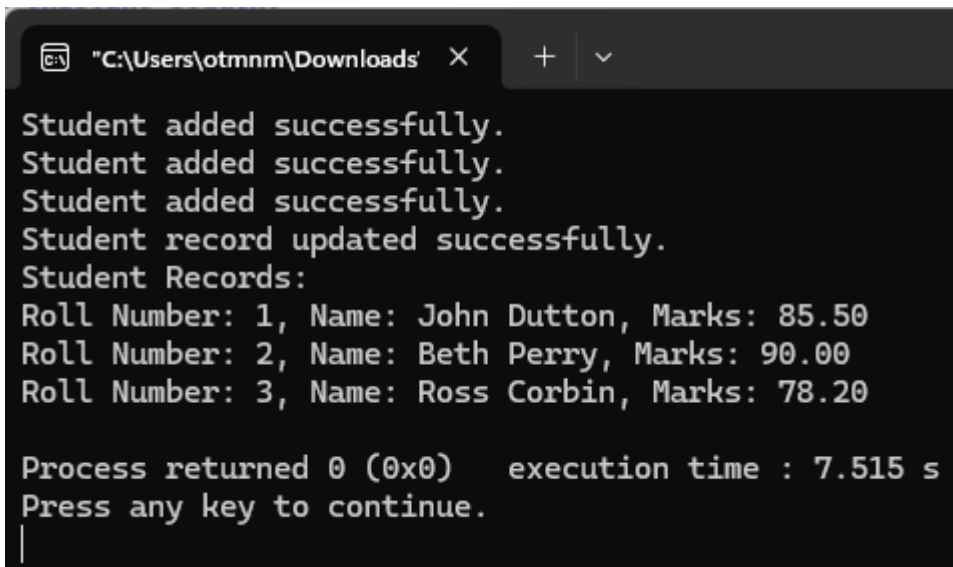
    // Update a student
    updateStudent(2, "Beth Perry", 90.0);

    // Display all student records
    displayStudents();

    return 0;
}

```

Output:



```

Student added successfully.
Student added successfully.
Student added successfully.
Student record updated successfully.
Student Records:
Roll Number: 1, Name: John Dutton, Marks: 85.50
Roll Number: 2, Name: Beth Perry, Marks: 90.00
Roll Number: 3, Name: Ross Corbin, Marks: 78.20

Process returned 0 (0x0)   execution time : 7.515 s
Press any key to continue.
|

```

This code implements a student record management system using a C data structure.

1. A Student structure is defined to encapsulate student information, including rollNumber (integer), name (character array), and marks (floating-point number).
2. An array of Student structures (students) is declared to store multiple student records, with a pre-defined maximum capacity (MAX\_STUDENTS).
3. Functions are implemented:
  - addStudent adds a new student record to the array.
  - updateStudent modifies an existing student record based on roll number.
  - displayStudents iterates through the array and displays all stored student records.
4. The main function demonstrates the usage of these functions by adding sample student records, updating a specific record, and displaying the entire student list.

## Task 2: Stacks and Queues

Scenario: The university encounters the need to manage a queue of tasks associated with student admissions and course enrollments, employing both stack and queue data structures.

```
// Task No. 2: Stacks and Queues

#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100

// Stack Structure and Functions
struct Stack {
    int arr[MAX_SIZE]; // Array to store stack elements
    int top;           // Index of the top element
};

void initStack(struct Stack* stack) {
    stack->top = -1; // Initialize top to -1 for empty stack
}

int isEmpty(struct Stack* stack) {
    return stack->top == -1; // Check if top is -1 (empty)
}

int isFull(struct Stack* stack) {
    return stack->top == MAX_SIZE - 1; // Check if top is at max size (full)
}

void push(struct Stack* stack, int item) {
    if (isFull(stack)) {
        printf("Error: Stack overflow.\n");
        return;
    }
    stack->arr[++stack->top] = item; // Increment top, then assign item
    printf("%d pushed to stack.\n", item);
}

int pop(struct Stack* stack) {
    if (isEmpty(stack)) {
        printf("Error: Stack underflow.\n");
        return -1;
    }
    int item = stack->arr[stack->top--]; // Get item, then decrement top
    printf("%d popped from stack.\n", item);
    return item;
}
```

```

// Queue Structure and Functions
struct Queue {
    int arr[MAX_SIZE]; // Array to store queue elements
    int front, rear;    // Indices for front and rear elements
};

void initQueue(struct Queue* queue) {
    queue->front = 0;
    queue->rear = -1; // Initialize rear to -1 for empty queue
}

int isQueueEmpty(struct Queue* queue) {
    return queue->front > queue->rear; // Check if front is ahead of rear (empty)
}

int isQueueFull(struct Queue* queue) {
    return queue->rear == MAX_SIZE - 1; // Check if rear is at max size (full)
}

void enqueue(struct Queue* queue, int item) {
    if (isQueueFull(queue)) {
        printf("Error: Queue overflow.\n");
        return;
    }
    queue->arr[++queue->rear] = item; // Increment rear, then assign item
    printf("%d enqueued to queue.\n", item);
}

int dequeue(struct Queue* queue) {
    if (isQueueEmpty(queue)) {
        printf("Error: Queue underflow.\n");
        return -1;
    }
    int item = queue->arr[queue->front++]; // Get item, then increment front
    printf("%d dequeued from queue.\n", item);
    return item;
}

```

```

int main() {
    struct Stack stack;
    struct Queue queue;

    initStack(&stack);
    initQueue(&queue);

    // Simulate tasks (e.g., admissions) using stack (LIFO)
    push(&stack, 1);
    push(&stack, 2);
    push(&stack, 3);

    // Simulate tasks (e.g., enrollments) using queue (FIFO)
    enqueue(&queue, 10);
    enqueue(&queue, 20);
    enqueue(&queue, 30);

    // Process tasks
    pop(&stack);
    dequeue(&queue);
    pop(&stack);
    dequeue(&queue);
    pop(&stack);
    dequeue(&queue);

    return 0;
}

```

Output:

```

"C:\Users\otmnm\Downloads" X + v
1 pushed to stack.
2 pushed to stack.
3 pushed to stack.
10 enqueued to queue.
20 enqueued to queue.
30 enqueued to queue.
3 popped from stack.
10 dequeued from queue.
2 popped from stack.
20 dequeued from queue.
1 popped from stack.
30 dequeued from queue.

Process returned 0 (0x0)   execution time : 8.558 s
Press any key to continue.
|

```

This code implements array-based stacks and queues for university task processing.

- Stack and Queue structures manage elements with top (stack) and front/rear (queue) indices.
- Functions handle initialization, emptiness/fullness checks, and element insertion/removal for both structures.
- The main function simulates tasks (e.g., admissions, enrollments) using stacks (LIFO) and queues (FIFO).

### Task 3: Linked Lists

Scenario: The university encounters the need to manage a queue of tasks associated with student admissions and course enrollments, employing both stack and queue data structures.



```
// Task No. 3: Linked Lists
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
// Linked List Node Structure
```

```
struct Node {
    char name[50]; // Name of the club member
    struct Node* next; // Pointer to the next node in the list
};
```

```
// Function to create a new node
```

```
struct Node* createNode(char name[]) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    strcpy(newNode->name, name); // Copy name to the new node
    newNode->next = NULL; // Set next pointer to NULL (end of list)
    return newNode;
}
```

```
// Function to add a member to the linked list
```

```
void addMember(struct Node** head, char name[]) {
    struct Node* newNode = createNode(name);
    if (*head == NULL) {
        *head = newNode; // Set head to the new node (empty list)
    } else {
        struct Node* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next; // Traverse to the last node
        }
        temp->next = newNode; // Add the new node at the end
    }
    printf("Member '%s' added to the list.\n", name);
}
```

```
// Function to delete a member from the linked list
```

```
void deleteMember(struct Node** head, char name[]) {
    struct Node* temp = *head, *prev = NULL;
    if (temp != NULL && strcmp(temp->name, name) == 0) {
        *head = temp->next; // Update head if deleting the first node
        free(temp);
        printf("Member '%s' deleted from the list.\n", name);
        return;
    }
}
```

```

while (temp != NULL && strcmp(temp->name, name) != 0) {
    prev = temp;
    temp = temp->next; // Traverse and find the node to delete
}
if (temp == NULL) {
    printf("Member '%s' not found in the list.\n", name);
    return;
}
prev->next = temp->next; // Bypass the deleted node
free(temp);
printf("Member '%s' deleted from the list.\n", name);
}

// Function to display all members in the list
void displayMembers(struct Node* head) {
    struct Node* temp = head;
    if (temp == NULL) {
        printf("The list is empty.\n");
        return;
    }
    printf("List of members:\n");
    while (temp != NULL) {
        printf("%s\n", temp->name);
        temp = temp->next; // Traverse and print each member
    }
}

int main() {
    struct Node* head = NULL;

    // Add members to the list
    addMember(&head, "John");
    addMember(&head, "Alice");
    addMember(&head, "Bob");

    // Display members in the list
    displayMembers(head);

    // Delete a member from the list
    deleteMember(&head, "Alice");

    // Display updated list
    displayMembers(head);

    return 0;
}

```

Output:

```
"C:\Users\otmnm\Downloads" X + v
Member 'John' added to the list.
Member 'Beth' added to the list.
Member 'Rip' added to the list.
List of members:
John
Beth
Rip
Member 'Beth' deleted from the list.
List of members:
John
Rip

Process returned 0 (0x0)   execution time : 7.997 s
Press any key to continue.
|
```

This code implements a linked list data structure to manage a dynamic list of student club members.

1. Linked List Node:
  - A Node structure is defined to represent a member in the linked list. It includes a name character array and a next pointer for referencing the subsequent node in the list.
2. Linked List Functions:
  - createNode: Allocates memory for a new node, copies the provided name into the node, sets next to NULL, and returns the new node pointer.
  - addMember: Creates a new node, adds it to the end of the list (updating head if empty), and prints a success message.
  - deleteMember: Searches for a member by name. If found, it updates head (if deleting the first node), removes the node, and frees its memory. A message is printed based on success or failure.
  - displayMembers: Iterates through the list, printing the name of each member.
3. Main Function:
  - Initializes an empty linked list (head).
  - Adds sample members using addMember.
  - Displays the entire list using displayMembers.
  - Deletes a specific member using deleteMember.
  - Displays the updated list again using displayMembers.

This code demonstrates the effectiveness of linked lists for managing dynamic collections of student club members.

## Task 4: Trees

Scenario: The university seeks to organize student information hierarchically, classifying it by department and course, employing a binary tree.

```
// Task No. 4: Trees

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Binary Tree Node Structure
struct Node {
    char courseName[50]; // Name of the course
    struct Node* left;   // Pointer to the left child node
    struct Node* right;  // Pointer to the right child node
};

// Function to create a new node
struct Node* createNode(char courseName[]) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    strcpy(newNode->courseName, courseName); // Copy course name
    newNode->left = NULL;                    // Initialize left child to NULL
    newNode->right = NULL;                   // Initialize right child to NULL
    return newNode;
}

// Function to insert a course into the binary tree
struct Node* insertCourse(struct Node* root, char courseName[]) {
    if (root == NULL) {
        return createNode(courseName); // Create a new node if tree is empty
    }
    if (strcmp(courseName, root->courseName) < 0) {
        root->left = insertCourse(root->left, courseName); // Insert left if smaller
    } else if (strcmp(courseName, root->courseName) > 0) {
        root->right = insertCourse(root->right, courseName); // Insert right if larger
    }
    // No duplicates allowed, so return the unchanged root for existing courses
    return root;
}

// Function to create a binary tree of courses (sample)
struct Node* createCourseTree() {
    struct Node* root = NULL;
    root = insertCourse(root, "Web Systems");
    root = insertCourse(root, "Databases");
    root = insertCourse(root, "IT Projects");
    root = insertCourse(root, "Algorithms and Complexity");
    root = insertCourse(root, "Cyber-space Security");
    return root;
}
```

```

// In-order traversal (prints courses in sorted order)
void inorderTraversal(struct Node* root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%s\n", root->courseName);
        inorderTraversal(root->right);
    }
}

// Pre-order traversal (visits root first, then children)
void preorderTraversal(struct Node* root) {
    if (root != NULL) {
        printf("%s\n", root->courseName);
        preorderTraversal(root->left);
        preorderTraversal(root->right);
    }
}

// Post-order traversal (visits leaves first, then parents)
void postorderTraversal(struct Node* root) {
    if (root != NULL) {
        postorderTraversal(root->left);
        postorderTraversal(root->right);
        printf("%s\n", root->courseName);
    }
}

int main() {
    struct Node* root = createCourseTree();

    printf("In-order traversal:\n");
    inorderTraversal(root);

    printf("\nPre-order traversal:\n");
    preorderTraversal(root);

    printf("\nPost-order traversal:\n");
    postorderTraversal(root);

    return 0;
}

```

Output:

```
"C:\Users\otmmn\Downloads" X + v
In-order traversal:
Algorithms and Complexity
Cyber-space Security
Databases
IT Projects
Web Systems

Pre-order traversal:
Web Systems
Databases
Algorithms and Complexity
Cyber-space Security
IT Projects

Post-order traversal:
Cyber-space Security
Algorithms and Complexity
IT Projects
Databases
Web Systems

Process returned 0 (0x0)   execution time : 7.893 s
Press any key to continue.
|
```

This code utilizes a binary tree to represent a hierarchical organization of student course information within a university system.

1. Binary Tree Node:
  - A Node structure is defined to represent a course in the binary tree. It includes a courseName character array and two pointers, left and right, to reference the left and right child nodes, respectively.
2. Binary Tree Functions:
  - createNode: Allocates memory for a new Node, copies the provided courseName into the node, and initializes both left and right pointers to NULL (representing an empty subtree).
  - insertCourse: Inserts a new course (courseName) into the binary tree, maintaining a sorted order based on course names. If the tree is empty, it creates a new node. Otherwise, it recursively inserts the course into the left subtree (for smaller names) or right subtree (for larger names).
3. Tree Creation and Traversal Functions:
  - createCourseTree: Constructs a sample binary tree of courses by calling insertCourse for each course name.
  - inorderTraversal: Implements an in-order traversal, visiting the left subtree, printing the current course name, and then visiting the right subtree. This results in a listing of courses in sorted order.

- preorderTraversal: Implements a pre-order traversal, printing the current course name first, followed by recursive visits to the left and right subtrees.
  - postorderTraversal: Implements a post-order traversal, visiting the left and right subtrees first, and then printing the current course name.
4. Main Function:
- Creates a root node for the course tree using createCourseTree.
  - Performs in-order, pre-order, and post-order traversals to demonstrate different course listing orders based on the traversal type.

This code showcases how a binary tree can be effectively used to organize and display student course information in a hierarchical manner within a university system.

## Task 5: Graphs

Scenario: The university endeavors to model relationships between different departments using a graph.

```
// Task No. 5: Graphs

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_DEPARTMENTS 10

// Graph Node Structure
struct Node {
    char departmentName[50]; // Name of the department
    int numConnections;       // Number of connected departments
    int connections[MAX_DEPARTMENTS]; // Indices of connected departments
};

// Graph Structure
struct Graph {
    int numDepartments;
    struct Node departments[MAX_DEPARTMENTS];
};

// Function to find the index of a department in the graph
int findDepartmentIndex(struct Graph* graph, char departmentName[]) {
    for (int i = 0; i < graph->numDepartments; i++) {
        if (strcmp(graph->departments[i].departmentName, departmentName) == 0) {
            return i;
        }
    }
    return -1;
}

// Function to create a new graph
struct Graph* createGraph() {
    struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));
    graph->numDepartments = 0;
    return graph;
}
```

```

// Function to add a department to the graph
void addDepartment(struct Graph* graph, char departmentName[]) {
    if (graph->numDepartments < MAX_DEPARTMENTS) {
        strcpy(graph->departments[graph->numDepartments].departmentName, departmentName);
        graph->departments[graph->numDepartments].numConnections = 0;
        graph->numDepartments++;
        printf("Department '%s' added to the graph.\n", departmentName);
    } else {
        printf("Error: Maximum number of departments reached.\n");
    }
}

// Function to add an edge between two departments
void addEdge(struct Graph* graph, char dept1[], char dept2[]) {
    int index1 = findDepartmentIndex(graph, dept1);
    int index2 = findDepartmentIndex(graph, dept2);
    if (index1 == -1 || index2 == -1) {
        printf("Error: One or both departments not found in the graph.\n");
        return;
    }
    graph->departments[index1].connections[graph->departments[index1].numConnections++] = index2;
    graph->departments[index2].connections[graph->departments[index2].numConnections++] = index1;
    printf("Edge added between '%s' and '%s'.\n", dept1, dept2);
}

// Function to display the relationships between departments
void displayRelationships(struct Graph* graph) {
    for (int i = 0; i < graph->numDepartments; i++) {
        printf("Department '%s' is connected to:\n", graph->departments[i].departmentName);
        for (int j = 0; j < graph->departments[i].numConnections; j++) {
            int connectedDepartmentIndex = graph->departments[i].connections[j];
            printf("- %s\n", graph->departments[connectedDepartmentIndex].departmentName);
        }
        printf("\n");
    }
}

// Depth-First Search (DFS) traversal
void dfs(struct Graph* graph, int startIndex, int visited[]) {
    visited[startIndex] = 1;
    printf("%s ", graph->departments[startIndex].departmentName);
    for (int i = 0; i < graph->departments[startIndex].numConnections; i++) {
        int connectedDepartmentIndex = graph->departments[startIndex].connections[i];
        if (!visited[connectedDepartmentIndex]) {
            dfs(graph, connectedDepartmentIndex, visited);
        }
    }
}

// Breadth-First Search (BFS) traversal
void bfs(struct Graph* graph, int startIndex) {
    int visited[MAX_DEPARTMENTS] = {0}; // Array to store visited departments (initialized to 0)
    int queue[MAX_DEPARTMENTS]; // Queue to manage departments to be visited
    int front = 0, rear = 0; // Queue front and rear indices

    visited[startIndex] = 1; // Mark starting department as visited
    queue[rear++] = startIndex; // Add starting department to the queue

    while (front != rear) { // Loop until queue is empty
        int currentIndex = queue[front++]; // Get the department at the queue front and remove it
        printf("%s ", graph->departments[currentIndex].departmentName); // Print current department

        for (int i = 0; i < graph->departments[currentIndex].numConnections; i++) {
            int connectedDepartmentIndex = graph->departments[currentIndex].connections[i];
            if (!visited[connectedDepartmentIndex]) { // Check if connected department is not visited
                visited[connectedDepartmentIndex] = 1; // Mark connected department as visited
                queue[rear++] = connectedDepartmentIndex; // Add connected department to the queue
            }
        }
    }
}

```



```
// Main function
```

```
int main() {  
    struct Graph* graph = createGraph();  
  
    // Add departments to the graph  
    addDepartment(graph, "Law");  
    addDepartment(graph, "History");  
    addDepartment(graph, "Engineering");  
    addDepartment(graph, "Business Administration");  
    addDepartment(graph, "Accounting");  
  
    // Add edges between departments  
    addEdge(graph, "Law", "History");  
    addEdge(graph, "Law", "Engineering");  
    addEdge(graph, "History", "Engineering");  
    addEdge(graph, "Engineering", "Business Administration");  
    addEdge(graph, "Business Administration", "Accounting");  
  
    // Display relationships between departments  
    printf("\nDepartment Relationships:\n");  
    displayRelationships(graph);  
  
    // Perform DFS and BFS traversals  
    printf("\nDepth-First Search (DFS) starting from Law:\n");  
    int visited[MAX_DEPARTMENTS] = {0};  
    dfs(graph, findDepartmentIndex(graph, "Law"), visited);  
    printf("\n\nBreadth-First Search (BFS) starting from Law:\n");  
    bfs(graph, findDepartmentIndex(graph, "Law"));  
  
    return 0;  
}
```

Output:

```
"C:\Users\otmnm\Downloads" X + v
Department 'Law' added to the graph.
Department 'History' added to the graph.
Department 'Engineering' added to the graph.
Department 'Business Administration' added to the graph.
Department 'Accounting' added to the graph.
Edge added between 'Law' and 'History'.
Edge added between 'Law' and 'Engineering'.
Edge added between 'History' and 'Engineering'.
Edge added between 'Engineering' and 'Business Administration'.
Edge added between 'Business Administration' and 'Accounting'.

Department Relationships:
Department 'Law' is connected to:
- History
- Engineering

Department 'History' is connected to:
- Law
- Engineering

Department 'Engineering' is connected to:
- Law
- History
- Business Administration

Department 'Business Administration' is connected to:
- Engineering
- Accounting

Department 'Accounting' is connected to:
- Business Administration

Depth-First Search (DFS) starting from Law:
Law History Engineering Business Administration Accounting

Breadth-First Search (BFS) starting from Law:
Law History Engineering Business Administration Accounting
Process returned 0 (0x0)    execution time : 7.864 s
Press any key to continue.
|
```

This code implements a graph data structure to model departmental relationships within an academic institution.

- Data Structures:
  - A Node struct represents a department, containing its name, number of connections, and an array of indices for connected departments.
  - A Graph struct holds an array of Nodes and the total number of departments.
- Graph Operations:
  - Functions are implemented to create a new graph, add departments, establish connections (edges) between departments, and display the existing departmental relationships.
- Traversal Algorithms:
  - Depth-First Search (DFS) and Breadth-First Search (BFS) algorithms are implemented to explore and navigate the departmental connections.
- Main Function:
  - The main function demonstrates the usage of the implemented functionalities. It creates a graph, adds departments representing academic disciplines (e.g., Law, History), establishes connections between them, displays the relationships, and performs DFS and BFS traversals starting from a specific department (e.g., Law).

## Task 6: Sorting

Scenario: The university mandates the sorting of student records based on their roll numbers for expedited retrieval.

```

// Task No. 6: Sorting

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// Define student structure
typedef struct Student {
    int rollNumber;
    char name[50];
    float marks;
} Student;

// Generate random student records
void generateRandomStudents(Student students[], int size) {
    for (int i = 0; i < size; i++) {
        students[i].rollNumber = i + 1;
        sprintf(students[i].name, "Student%d", i + 1);
        students[i].marks = (float)rand() / RAND_MAX * 100.0;
    }
}

// Swap two students
void swap(Student* a, Student* b) {
    Student temp = *a;
    *a = *b;
    *b = temp;
}

// Bubble Sort
void bubbleSort(Student students[], int size) {
    for (int i = 0; i < size - 1; i++) {
        for (int j = 0; j < size - i - 1; j++) {
            if (students[j].rollNumber > students[j + 1].rollNumber) {
                swap(&students[j], &students[j + 1]);
            }
        }
    }
}

```

```

// Insertion Sort
void insertionSort(Student students[], int size) {
    for (int i = 1; i < size; i++) {
        Student key = students[i];
        int j = i - 1;
        while (j >= 0 && students[j].rollNumber > key.rollNumber) {
            students[j + 1] = students[j];
            j--;
        }
        students[j + 1] = key;
    }
}

// Quick Sort partition
int partition(Student students[], int low, int high) {
    int pivot = students[high].rollNumber;
    int i = low - 1;
    for (int j = low; j <= high - 1; j++) {
        if (students[j].rollNumber < pivot) {
            i++;
            swap(&students[i], &students[j]);
        }
    }
    swap(&students[i + 1], &students[high]);
    return i + 1;
}

// Quick Sort
void quickSort(Student students[], int low, int high) {
    if (low < high) {
        int pi = partition(students, low, high);
        quickSort(students, low, pi - 1);
        quickSort(students, pi + 1, high);
    }
}

// Display student records
void displayStudents(Student students[], int size) {
    printf("Student Records:\n");
    for (int i = 0; i < size; i++) {
        printf("Roll Number: %d, Name: %s, Marks: %.2f\n", students[i].rollNumber, students[i].name, students[i].marks);
    }
}

```

```

int main() {
    int size = 1000;
    Student* students = (Student*)malloc(size * sizeof(Student));

    // Generate random student records
    srand(time(NULL));
    generateRandomStudents(students, size);

    // Sort using different algorithms
    clock_t start, end;

    // Bubble Sort
    start = clock();
    bubbleSort(students, size);
    end = clock();
    printf("Bubble Sort Time: %.6f seconds\n", (double)(end - start) / CLOCKS_PER_SEC);

    // Reset student records
    generateRandomStudents(students, size);

    // Insertion Sort
    start = clock();
    insertionSort(students, size);
    end = clock();
    printf("Insertion Sort Time: %.6f seconds\n", (double)(end - start) / CLOCKS_PER_SEC);

    // Reset student records
    generateRandomStudents(students, size);

    // Quick Sort
    start = clock();
    quickSort(students, 0, size - 1);
    end = clock();
    printf("Quick Sort Time: %.6f seconds\n", (double)(end - start) / CLOCKS_PER_SEC);

    // Display sorted student records
    displayStudents(students, size);

    free(students);
    return 0;
}

```

Output:

Bubble Sort Time: 0.001000 seconds

Insertion Sort Time: 0.000000 seconds

Quick Sort Time: 0.005000 seconds

Student Records:

Roll Number: 1, Name: Student1, Marks: 41.00

Roll Number: 2, Name: Student2, Marks: 3.58

Roll Number: 3, Name: Student3, Marks: 41.89

Roll Number: 4, Name: Student4, Marks: 99.33

Roll Number: 5, Name: Student5, Marks: 8.26

Roll Number: 6, Name: Student6, Marks: 92.61

Roll Number: 7, Name: Student7, Marks: 91.68

Roll Number: 8, Name: Student8, Marks: 31.59

Roll Number: 9, Name: Student9, Marks: 70.00

Roll Number: 10, Name: Student10, Marks: 21.89

Roll Number: 11, Name: Student11, Marks: 90.15

Roll Number: 12, Name: Student12, Marks: 87.51

Roll Number: 13, Name: Student13, Marks: 17.15  
Roll Number: 14, Name: Student14, Marks: 28.89  
Roll Number: 15, Name: Student15, Marks: 44.35  
Roll Number: 16, Name: Student16, Marks: 37.84  
Roll Number: 17, Name: Student17, Marks: 11.91  
Roll Number: 18, Name: Student18, Marks: 53.57  
Roll Number: 19, Name: Student19, Marks: 4.50  
Roll Number: 20, Name: Student20, Marks: 71.40  
Roll Number: 21, Name: Student21, Marks: 19.36  
Roll Number: 22, Name: Student22, Marks: 56.28  
Roll Number: 23, Name: Student23, Marks: 73.04  
Roll Number: 24, Name: Student24, Marks: 80.57  
Roll Number: 25, Name: Student25, Marks: 13.35  
Roll Number: 26, Name: Student26, Marks: 12.30  
Roll Number: 27, Name: Student27, Marks: 83.30  
Roll Number: 28, Name: Student28, Marks: 37.64  
Roll Number: 29, Name: Student29, Marks: 10.10  
Roll Number: 30, Name: Student30, Marks: 23.88  
Roll Number: 31, Name: Student31, Marks: 26.11  
Roll Number: 32, Name: Student32, Marks: 27.56  
Roll Number: 33, Name: Student33, Marks: 32.31  
Roll Number: 34, Name: Student34, Marks: 40.13  
Roll Number: 35, Name: Student35, Marks: 52.36  
Roll Number: 36, Name: Student36, Marks: 52.83  
Roll Number: 37, Name: Student37, Marks: 57.12  
Roll Number: 38, Name: Student38, Marks: 25.74  
Roll Number: 39, Name: Student39, Marks: 29.63  
Roll Number: 40, Name: Student40, Marks: 75.99  
Roll Number: 41, Name: Student41, Marks: 51.05  
Roll Number: 42, Name: Student42, Marks: 68.76  
Roll Number: 43, Name: Student43, Marks: 81.07  
Roll Number: 44, Name: Student44, Marks: 15.53  
Roll Number: 45, Name: Student45, Marks: 44.82  
Roll Number: 46, Name: Student46, Marks: 68.42  
Roll Number: 47, Name: Student47, Marks: 84.64  
Roll Number: 48, Name: Student48, Marks: 60.66  
Roll Number: 49, Name: Student49, Marks: 51.52  
Roll Number: 50, Name: Student50, Marks: 93.37  
Roll Number: 51, Name: Student51, Marks: 60.81  
Roll Number: 52, Name: Student52, Marks: 20.70  
Roll Number: 53, Name: Student53, Marks: 80.91  
Roll Number: 54, Name: Student54, Marks: 51.07  
Roll Number: 55, Name: Student55, Marks: 26.88  
Roll Number: 56, Name: Student56, Marks: 74.67  
Roll Number: 57, Name: Student57, Marks: 53.23  
Roll Number: 58, Name: Student58, Marks: 58.96  
Roll Number: 59, Name: Student59, Marks: 46.02  
Roll Number: 60, Name: Student60, Marks: 36.66  
Roll Number: 61, Name: Student61, Marks: 65.53  
Roll Number: 62, Name: Student62, Marks: 58.00  
Roll Number: 63, Name: Student63, Marks: 49.13

Roll Number: 64, Name: Student64, Marks: 2.65  
Roll Number: 65, Name: Student65, Marks: 13.63  
Roll Number: 66, Name: Student66, Marks: 59.19  
Roll Number: 67, Name: Student67, Marks: 57.72  
Roll Number: 68, Name: Student68, Marks: 94.50  
Roll Number: 69, Name: Student69, Marks: 22.91  
Roll Number: 70, Name: Student70, Marks: 63.77  
Roll Number: 71, Name: Student71, Marks: 85.89  
Roll Number: 72, Name: Student72, Marks: 66.45  
Roll Number: 73, Name: Student73, Marks: 91.02  
Roll Number: 74, Name: Student74, Marks: 47.67  
Roll Number: 75, Name: Student75, Marks: 49.50  
Roll Number: 76, Name: Student76, Marks: 90.11  
Roll Number: 77, Name: Student77, Marks: 95.53  
Roll Number: 78, Name: Student78, Marks: 50.78  
Roll Number: 79, Name: Student79, Marks: 60.71  
Roll Number: 80, Name: Student80, Marks: 83.32  
Roll Number: 81, Name: Student81, Marks: 69.92  
Roll Number: 82, Name: Student82, Marks: 61.82  
Roll Number: 83, Name: Student83, Marks: 36.03  
Roll Number: 84, Name: Student84, Marks: 98.57  
Roll Number: 85, Name: Student85, Marks: 0.70  
Roll Number: 86, Name: Student86, Marks: 39.30  
Roll Number: 87, Name: Student87, Marks: 95.92  
Roll Number: 88, Name: Student88, Marks: 36.66  
Roll Number: 89, Name: Student89, Marks: 49.01  
Roll Number: 90, Name: Student90, Marks: 59.21  
Roll Number: 91, Name: Student91, Marks: 84.13  
Roll Number: 92, Name: Student92, Marks: 1.13  
Roll Number: 93, Name: Student93, Marks: 49.75  
Roll Number: 94, Name: Student94, Marks: 30.12  
Roll Number: 95, Name: Student95, Marks: 84.91  
Roll Number: 96, Name: Student96, Marks: 59.19  
Roll Number: 97, Name: Student97, Marks: 91.43  
Roll Number: 98, Name: Student98, Marks: 66.33  
Roll Number: 99, Name: Student99, Marks: 66.20  
Roll Number: 100, Name: Student100, Marks: 24.66  
Roll Number: 101, Name: Student101, Marks: 29.67  
Roll Number: 102, Name: Student102, Marks: 59.56  
Roll Number: 103, Name: Student103, Marks: 26.89  
Roll Number: 104, Name: Student104, Marks: 28.66  
Roll Number: 105, Name: Student105, Marks: 37.77  
Roll Number: 106, Name: Student106, Marks: 39.98  
Roll Number: 107, Name: Student107, Marks: 76.33  
Roll Number: 108, Name: Student108, Marks: 93.60  
Roll Number: 109, Name: Student109, Marks: 47.22  
Roll Number: 110, Name: Student110, Marks: 67.09  
Roll Number: 111, Name: Student111, Marks: 24.17  
Roll Number: 112, Name: Student112, Marks: 76.06  
Roll Number: 113, Name: Student113, Marks: 81.41  
Roll Number: 114, Name: Student114, Marks: 41.08



Roll Number: 115, Name: Student115, Marks: 8.73  
Roll Number: 116, Name: Student116, Marks: 94.39  
Roll Number: 117, Name: Student117, Marks: 35.59  
Roll Number: 118, Name: Student118, Marks: 75.39  
Roll Number: 119, Name: Student119, Marks: 41.90  
Roll Number: 120, Name: Student120, Marks: 6.31  
Roll Number: 121, Name: Student121, Marks: 6.37  
Roll Number: 122, Name: Student122, Marks: 20.97  
Roll Number: 123, Name: Student123, Marks: 98.81  
Roll Number: 124, Name: Student124, Marks: 52.49  
Roll Number: 125, Name: Student125, Marks: 23.55  
Roll Number: 126, Name: Student126, Marks: 82.94  
Roll Number: 127, Name: Student127, Marks: 30.35  
Roll Number: 128, Name: Student128, Marks: 31.48  
Roll Number: 129, Name: Student129, Marks: 87.81  
Roll Number: 130, Name: Student130, Marks: 20.27  
Roll Number: 131, Name: Student131, Marks: 26.65  
Roll Number: 132, Name: Student132, Marks: 21.76  
Roll Number: 133, Name: Student133, Marks: 67.08  
Roll Number: 134, Name: Student134, Marks: 69.09  
Roll Number: 135, Name: Student135, Marks: 9.69  
Roll Number: 136, Name: Student136, Marks: 16.47  
Roll Number: 137, Name: Student137, Marks: 54.83  
Roll Number: 138, Name: Student138, Marks: 80.21  
Roll Number: 139, Name: Student139, Marks: 83.07  
Roll Number: 140, Name: Student140, Marks: 86.46  
Roll Number: 141, Name: Student141, Marks: 43.48  
Roll Number: 142, Name: Student142, Marks: 11.62  
Roll Number: 143, Name: Student143, Marks: 17.22  
Roll Number: 144, Name: Student144, Marks: 37.25  
Roll Number: 145, Name: Student145, Marks: 15.87  
Roll Number: 146, Name: Student146, Marks: 66.42  
Roll Number: 147, Name: Student147, Marks: 98.06  
Roll Number: 148, Name: Student148, Marks: 75.66  
Roll Number: 149, Name: Student149, Marks: 8.08  
Roll Number: 150, Name: Student150, Marks: 16.88  
Roll Number: 151, Name: Student151, Marks: 17.18  
Roll Number: 152, Name: Student152, Marks: 51.52  
Roll Number: 153, Name: Student153, Marks: 46.64  
Roll Number: 154, Name: Student154, Marks: 5.31  
Roll Number: 155, Name: Student155, Marks: 31.89  
Roll Number: 156, Name: Student156, Marks: 90.26  
Roll Number: 157, Name: Student157, Marks: 13.32  
Roll Number: 158, Name: Student158, Marks: 62.29  
Roll Number: 159, Name: Student159, Marks: 72.98  
Roll Number: 160, Name: Student160, Marks: 81.94  
Roll Number: 161, Name: Student161, Marks: 40.50  
Roll Number: 162, Name: Student162, Marks: 32.86  
Roll Number: 163, Name: Student163, Marks: 28.56  
Roll Number: 164, Name: Student164, Marks: 42.36  
Roll Number: 165, Name: Student165, Marks: 90.42

Roll Number: 166, Name: Student166, Marks: 51.45  
Roll Number: 167, Name: Student167, Marks: 81.96  
Roll Number: 168, Name: Student168, Marks: 11.83  
Roll Number: 169, Name: Student169, Marks: 21.30  
Roll Number: 170, Name: Student170, Marks: 55.96  
Roll Number: 171, Name: Student171, Marks: 31.85  
Roll Number: 172, Name: Student172, Marks: 7.29  
Roll Number: 173, Name: Student173, Marks: 93.50  
Roll Number: 174, Name: Student174, Marks: 81.77  
Roll Number: 175, Name: Student175, Marks: 72.71  
Roll Number: 176, Name: Student176, Marks: 93.37  
Roll Number: 177, Name: Student177, Marks: 18.85  
Roll Number: 178, Name: Student178, Marks: 26.25  
Roll Number: 179, Name: Student179, Marks: 63.81  
Roll Number: 180, Name: Student180, Marks: 88.01  
Roll Number: 181, Name: Student181, Marks: 6.26  
Roll Number: 182, Name: Student182, Marks: 24.44  
Roll Number: 183, Name: Student183, Marks: 64.77  
Roll Number: 184, Name: Student184, Marks: 68.30  
Roll Number: 185, Name: Student185, Marks: 6.75  
Roll Number: 186, Name: Student186, Marks: 26.42  
Roll Number: 187, Name: Student187, Marks: 65.81  
Roll Number: 188, Name: Student188, Marks: 92.09  
Roll Number: 189, Name: Student189, Marks: 11.06  
Roll Number: 190, Name: Student190, Marks: 17.11  
Roll Number: 191, Name: Student191, Marks: 90.94  
Roll Number: 192, Name: Student192, Marks: 1.12  
Roll Number: 193, Name: Student193, Marks: 2.82  
Roll Number: 194, Name: Student194, Marks: 69.06  
Roll Number: 195, Name: Student195, Marks: 2.01  
Roll Number: 196, Name: Student196, Marks: 21.15  
Roll Number: 197, Name: Student197, Marks: 20.62  
Roll Number: 198, Name: Student198, Marks: 68.90  
Roll Number: 199, Name: Student199, Marks: 82.02  
Roll Number: 200, Name: Student200, Marks: 24.82  
Roll Number: 201, Name: Student201, Marks: 31.94  
Roll Number: 202, Name: Student202, Marks: 8.00  
Roll Number: 203, Name: Student203, Marks: 25.43  
Roll Number: 204, Name: Student204, Marks: 72.83  
Roll Number: 205, Name: Student205, Marks: 72.12  
Roll Number: 206, Name: Student206, Marks: 78.03  
Roll Number: 207, Name: Student207, Marks: 14.08  
Roll Number: 208, Name: Student208, Marks: 99.04  
Roll Number: 209, Name: Student209, Marks: 51.51  
Roll Number: 210, Name: Student210, Marks: 62.09  
Roll Number: 211, Name: Student211, Marks: 6.40  
Roll Number: 212, Name: Student212, Marks: 55.20  
Roll Number: 213, Name: Student213, Marks: 83.87  
Roll Number: 214, Name: Student214, Marks: 61.86  
Roll Number: 215, Name: Student215, Marks: 18.27  
Roll Number: 216, Name: Student216, Marks: 80.80

Roll Number: 217, Name: Student217, Marks: 39.27  
Roll Number: 218, Name: Student218, Marks: 51.59  
Roll Number: 219, Name: Student219, Marks: 23.65  
Roll Number: 220, Name: Student220, Marks: 13.79  
Roll Number: 221, Name: Student221, Marks: 74.43  
Roll Number: 222, Name: Student222, Marks: 99.48  
Roll Number: 223, Name: Student223, Marks: 53.00  
Roll Number: 224, Name: Student224, Marks: 7.71  
Roll Number: 225, Name: Student225, Marks: 43.75  
Roll Number: 226, Name: Student226, Marks: 46.96  
Roll Number: 227, Name: Student227, Marks: 17.75  
Roll Number: 228, Name: Student228, Marks: 70.96  
Roll Number: 229, Name: Student229, Marks: 44.24  
Roll Number: 230, Name: Student230, Marks: 86.78  
Roll Number: 231, Name: Student231, Marks: 38.78  
Roll Number: 232, Name: Student232, Marks: 93.66  
Roll Number: 233, Name: Student233, Marks: 97.13  
Roll Number: 234, Name: Student234, Marks: 30.18  
Roll Number: 235, Name: Student235, Marks: 7.19  
Roll Number: 236, Name: Student236, Marks: 77.91  
Roll Number: 237, Name: Student237, Marks: 19.80  
Roll Number: 238, Name: Student238, Marks: 4.06  
Roll Number: 239, Name: Student239, Marks: 55.44  
Roll Number: 240, Name: Student240, Marks: 37.00  
Roll Number: 241, Name: Student241, Marks: 90.62  
Roll Number: 242, Name: Student242, Marks: 68.60  
Roll Number: 243, Name: Student243, Marks: 66.87  
Roll Number: 244, Name: Student244, Marks: 79.11  
Roll Number: 245, Name: Student245, Marks: 60.28  
Roll Number: 246, Name: Student246, Marks: 96.07  
Roll Number: 247, Name: Student247, Marks: 1.93  
Roll Number: 248, Name: Student248, Marks: 41.30  
Roll Number: 249, Name: Student249, Marks: 12.38  
Roll Number: 250, Name: Student250, Marks: 1.28  
Roll Number: 251, Name: Student251, Marks: 69.09  
Roll Number: 252, Name: Student252, Marks: 89.25  
Roll Number: 253, Name: Student253, Marks: 26.68  
Roll Number: 254, Name: Student254, Marks: 64.60  
Roll Number: 255, Name: Student255, Marks: 18.57  
Roll Number: 256, Name: Student256, Marks: 48.51  
Roll Number: 257, Name: Student257, Marks: 47.90  
Roll Number: 258, Name: Student258, Marks: 37.78  
Roll Number: 259, Name: Student259, Marks: 87.12  
Roll Number: 260, Name: Student260, Marks: 82.72  
Roll Number: 261, Name: Student261, Marks: 13.43  
Roll Number: 262, Name: Student262, Marks: 73.57  
Roll Number: 263, Name: Student263, Marks: 71.81  
Roll Number: 264, Name: Student264, Marks: 84.66  
Roll Number: 265, Name: Student265, Marks: 42.87  
Roll Number: 266, Name: Student266, Marks: 69.51  
Roll Number: 267, Name: Student267, Marks: 61.15

Roll Number: 268, Name: Student268, Marks: 95.51  
Roll Number: 269, Name: Student269, Marks: 42.60  
Roll Number: 270, Name: Student270, Marks: 66.60  
Roll Number: 271, Name: Student271, Marks: 1.49  
Roll Number: 272, Name: Student272, Marks: 18.10  
Roll Number: 273, Name: Student273, Marks: 28.60  
Roll Number: 274, Name: Student274, Marks: 93.56  
Roll Number: 275, Name: Student275, Marks: 26.88  
Roll Number: 276, Name: Student276, Marks: 39.74  
Roll Number: 277, Name: Student277, Marks: 20.47  
Roll Number: 278, Name: Student278, Marks: 97.09  
Roll Number: 279, Name: Student279, Marks: 30.63  
Roll Number: 280, Name: Student280, Marks: 30.18  
Roll Number: 281, Name: Student281, Marks: 9.90  
Roll Number: 282, Name: Student282, Marks: 49.03  
Roll Number: 283, Name: Student283, Marks: 6.47  
Roll Number: 284, Name: Student284, Marks: 30.52  
Roll Number: 285, Name: Student285, Marks: 56.71  
Roll Number: 286, Name: Student286, Marks: 70.79  
Roll Number: 287, Name: Student287, Marks: 37.68  
Roll Number: 288, Name: Student288, Marks: 98.43  
Roll Number: 289, Name: Student289, Marks: 15.47  
Roll Number: 290, Name: Student290, Marks: 65.85  
Roll Number: 291, Name: Student291, Marks: 62.10  
Roll Number: 292, Name: Student292, Marks: 25.50  
Roll Number: 293, Name: Student293, Marks: 46.01  
Roll Number: 294, Name: Student294, Marks: 0.88  
Roll Number: 295, Name: Student295, Marks: 91.29  
Roll Number: 296, Name: Student296, Marks: 92.30  
Roll Number: 297, Name: Student297, Marks: 10.78  
Roll Number: 298, Name: Student298, Marks: 26.10  
Roll Number: 299, Name: Student299, Marks: 11.94  
Roll Number: 300, Name: Student300, Marks: 76.74  
Roll Number: 301, Name: Student301, Marks: 12.24  
Roll Number: 302, Name: Student302, Marks: 75.56  
Roll Number: 303, Name: Student303, Marks: 47.56  
Roll Number: 304, Name: Student304, Marks: 81.36  
Roll Number: 305, Name: Student305, Marks: 73.48  
Roll Number: 306, Name: Student306, Marks: 37.87  
Roll Number: 307, Name: Student307, Marks: 8.73  
Roll Number: 308, Name: Student308, Marks: 95.22  
Roll Number: 309, Name: Student309, Marks: 15.05  
Roll Number: 310, Name: Student310, Marks: 38.12  
Roll Number: 311, Name: Student311, Marks: 9.85  
Roll Number: 312, Name: Student312, Marks: 55.97  
Roll Number: 313, Name: Student313, Marks: 31.27  
Roll Number: 314, Name: Student314, Marks: 21.53  
Roll Number: 315, Name: Student315, Marks: 80.73  
Roll Number: 316, Name: Student316, Marks: 27.76  
Roll Number: 317, Name: Student317, Marks: 19.04  
Roll Number: 318, Name: Student318, Marks: 79.51

Roll Number: 319, Name: Student319, Marks: 50.97  
Roll Number: 320, Name: Student320, Marks: 60.53  
Roll Number: 321, Name: Student321, Marks: 62.32  
Roll Number: 322, Name: Student322, Marks: 8.65  
Roll Number: 323, Name: Student323, Marks: 35.29  
Roll Number: 324, Name: Student324, Marks: 46.49  
Roll Number: 325, Name: Student325, Marks: 25.40  
Roll Number: 326, Name: Student326, Marks: 43.44  
Roll Number: 327, Name: Student327, Marks: 98.02  
Roll Number: 328, Name: Student328, Marks: 39.16  
Roll Number: 329, Name: Student329, Marks: 58.12  
Roll Number: 330, Name: Student330, Marks: 53.22  
Roll Number: 331, Name: Student331, Marks: 24.81  
Roll Number: 332, Name: Student332, Marks: 50.81  
Roll Number: 333, Name: Student333, Marks: 66.04  
Roll Number: 334, Name: Student334, Marks: 43.94  
Roll Number: 335, Name: Student335, Marks: 79.63  
Roll Number: 336, Name: Student336, Marks: 93.86  
Roll Number: 337, Name: Student337, Marks: 48.91  
Roll Number: 338, Name: Student338, Marks: 55.55  
Roll Number: 339, Name: Student339, Marks: 75.32  
Roll Number: 340, Name: Student340, Marks: 81.80  
Roll Number: 341, Name: Student341, Marks: 60.25  
Roll Number: 342, Name: Student342, Marks: 95.43  
Roll Number: 343, Name: Student343, Marks: 35.70  
Roll Number: 344, Name: Student344, Marks: 55.32  
Roll Number: 345, Name: Student345, Marks: 91.54  
Roll Number: 346, Name: Student346, Marks: 98.60  
Roll Number: 347, Name: Student347, Marks: 77.44  
Roll Number: 348, Name: Student348, Marks: 49.25  
Roll Number: 349, Name: Student349, Marks: 33.80  
Roll Number: 350, Name: Student350, Marks: 55.33  
Roll Number: 351, Name: Student351, Marks: 89.71  
Roll Number: 352, Name: Student352, Marks: 66.00  
Roll Number: 353, Name: Student353, Marks: 19.90  
Roll Number: 354, Name: Student354, Marks: 96.76  
Roll Number: 355, Name: Student355, Marks: 39.92  
Roll Number: 356, Name: Student356, Marks: 71.02  
Roll Number: 357, Name: Student357, Marks: 0.63  
Roll Number: 358, Name: Student358, Marks: 79.11  
Roll Number: 359, Name: Student359, Marks: 83.44  
Roll Number: 360, Name: Student360, Marks: 75.93  
Roll Number: 361, Name: Student361, Marks: 57.75  
Roll Number: 362, Name: Student362, Marks: 57.22  
Roll Number: 363, Name: Student363, Marks: 5.65  
Roll Number: 364, Name: Student364, Marks: 25.07  
Roll Number: 365, Name: Student365, Marks: 6.96  
Roll Number: 366, Name: Student366, Marks: 87.88  
Roll Number: 367, Name: Student367, Marks: 74.28  
Roll Number: 368, Name: Student368, Marks: 50.87  
Roll Number: 369, Name: Student369, Marks: 94.17

Roll Number: 370, Name: Student370, Marks: 53.77  
Roll Number: 371, Name: Student371, Marks: 30.20  
Roll Number: 372, Name: Student372, Marks: 13.88  
Roll Number: 373, Name: Student373, Marks: 37.94  
Roll Number: 374, Name: Student374, Marks: 48.45  
Roll Number: 375, Name: Student375, Marks: 94.95  
Roll Number: 376, Name: Student376, Marks: 92.99  
Roll Number: 377, Name: Student377, Marks: 21.41  
Roll Number: 378, Name: Student378, Marks: 13.16  
Roll Number: 379, Name: Student379, Marks: 22.79  
Roll Number: 380, Name: Student380, Marks: 41.40  
Roll Number: 381, Name: Student381, Marks: 86.77  
Roll Number: 382, Name: Student382, Marks: 65.95  
Roll Number: 383, Name: Student383, Marks: 75.81  
Roll Number: 384, Name: Student384, Marks: 74.13  
Roll Number: 385, Name: Student385, Marks: 35.32  
Roll Number: 386, Name: Student386, Marks: 13.90  
Roll Number: 387, Name: Student387, Marks: 49.84  
Roll Number: 388, Name: Student388, Marks: 2.51  
Roll Number: 389, Name: Student389, Marks: 86.40  
Roll Number: 390, Name: Student390, Marks: 88.87  
Roll Number: 391, Name: Student391, Marks: 29.60  
Roll Number: 392, Name: Student392, Marks: 81.47  
Roll Number: 393, Name: Student393, Marks: 98.19  
Roll Number: 394, Name: Student394, Marks: 97.60  
Roll Number: 395, Name: Student395, Marks: 0.97  
Roll Number: 396, Name: Student396, Marks: 85.00  
Roll Number: 397, Name: Student397, Marks: 3.58  
Roll Number: 398, Name: Student398, Marks: 26.64  
Roll Number: 399, Name: Student399, Marks: 98.71  
Roll Number: 400, Name: Student400, Marks: 75.73  
Roll Number: 401, Name: Student401, Marks: 64.18  
Roll Number: 402, Name: Student402, Marks: 92.80  
Roll Number: 403, Name: Student403, Marks: 17.54  
Roll Number: 404, Name: Student404, Marks: 83.94  
Roll Number: 405, Name: Student405, Marks: 95.60  
Roll Number: 406, Name: Student406, Marks: 79.73  
Roll Number: 407, Name: Student407, Marks: 64.94  
Roll Number: 408, Name: Student408, Marks: 61.87  
Roll Number: 409, Name: Student409, Marks: 67.20  
Roll Number: 410, Name: Student410, Marks: 51.27  
Roll Number: 411, Name: Student411, Marks: 83.61  
Roll Number: 412, Name: Student412, Marks: 28.75  
Roll Number: 413, Name: Student413, Marks: 29.34  
Roll Number: 414, Name: Student414, Marks: 82.20  
Roll Number: 415, Name: Student415, Marks: 21.78  
Roll Number: 416, Name: Student416, Marks: 72.34  
Roll Number: 417, Name: Student417, Marks: 71.30  
Roll Number: 418, Name: Student418, Marks: 96.91  
Roll Number: 419, Name: Student419, Marks: 79.53  
Roll Number: 420, Name: Student420, Marks: 22.55

Roll Number: 421, Name: Student421, Marks: 62.98  
Roll Number: 422, Name: Student422, Marks: 56.83  
Roll Number: 423, Name: Student423, Marks: 9.18  
Roll Number: 424, Name: Student424, Marks: 62.72  
Roll Number: 425, Name: Student425, Marks: 83.55  
Roll Number: 426, Name: Student426, Marks: 86.99  
Roll Number: 427, Name: Student427, Marks: 97.91  
Roll Number: 428, Name: Student428, Marks: 94.19  
Roll Number: 429, Name: Student429, Marks: 90.10  
Roll Number: 430, Name: Student430, Marks: 82.61  
Roll Number: 431, Name: Student431, Marks: 10.76  
Roll Number: 432, Name: Student432, Marks: 19.93  
Roll Number: 433, Name: Student433, Marks: 29.46  
Roll Number: 434, Name: Student434, Marks: 86.03  
Roll Number: 435, Name: Student435, Marks: 22.11  
Roll Number: 436, Name: Student436, Marks: 62.63  
Roll Number: 437, Name: Student437, Marks: 46.34  
Roll Number: 438, Name: Student438, Marks: 74.92  
Roll Number: 439, Name: Student439, Marks: 13.68  
Roll Number: 440, Name: Student440, Marks: 83.00  
Roll Number: 441, Name: Student441, Marks: 90.83  
Roll Number: 442, Name: Student442, Marks: 52.13  
Roll Number: 443, Name: Student443, Marks: 67.35  
Roll Number: 444, Name: Student444, Marks: 13.96  
Roll Number: 445, Name: Student445, Marks: 78.52  
Roll Number: 446, Name: Student446, Marks: 78.03  
Roll Number: 447, Name: Student447, Marks: 30.79  
Roll Number: 448, Name: Student448, Marks: 76.20  
Roll Number: 449, Name: Student449, Marks: 6.17  
Roll Number: 450, Name: Student450, Marks: 35.76  
Roll Number: 451, Name: Student451, Marks: 84.10  
Roll Number: 452, Name: Student452, Marks: 90.83  
Roll Number: 453, Name: Student453, Marks: 76.31  
Roll Number: 454, Name: Student454, Marks: 70.21  
Roll Number: 455, Name: Student455, Marks: 85.48  
Roll Number: 456, Name: Student456, Marks: 54.76  
Roll Number: 457, Name: Student457, Marks: 33.59  
Roll Number: 458, Name: Student458, Marks: 91.14  
Roll Number: 459, Name: Student459, Marks: 24.21  
Roll Number: 460, Name: Student460, Marks: 94.37  
Roll Number: 461, Name: Student461, Marks: 66.35  
Roll Number: 462, Name: Student462, Marks: 81.31  
Roll Number: 463, Name: Student463, Marks: 58.91  
Roll Number: 464, Name: Student464, Marks: 63.10  
Roll Number: 465, Name: Student465, Marks: 76.18  
Roll Number: 466, Name: Student466, Marks: 0.02  
Roll Number: 467, Name: Student467, Marks: 69.35  
Roll Number: 468, Name: Student468, Marks: 98.69  
Roll Number: 469, Name: Student469, Marks: 68.89  
Roll Number: 470, Name: Student470, Marks: 22.83  
Roll Number: 471, Name: Student471, Marks: 99.78

Roll Number: 472, Name: Student472, Marks: 5.51  
Roll Number: 473, Name: Student473, Marks: 69.88  
Roll Number: 474, Name: Student474, Marks: 8.03  
Roll Number: 475, Name: Student475, Marks: 22.06  
Roll Number: 476, Name: Student476, Marks: 77.83  
Roll Number: 477, Name: Student477, Marks: 16.95  
Roll Number: 478, Name: Student478, Marks: 30.50  
Roll Number: 479, Name: Student479, Marks: 96.60  
Roll Number: 480, Name: Student480, Marks: 29.37  
Roll Number: 481, Name: Student481, Marks: 33.92  
Roll Number: 482, Name: Student482, Marks: 73.54  
Roll Number: 483, Name: Student483, Marks: 59.27  
Roll Number: 484, Name: Student484, Marks: 45.15  
Roll Number: 485, Name: Student485, Marks: 37.93  
Roll Number: 486, Name: Student486, Marks: 19.37  
Roll Number: 487, Name: Student487, Marks: 12.44  
Roll Number: 488, Name: Student488, Marks: 20.81  
Roll Number: 489, Name: Student489, Marks: 83.69  
Roll Number: 490, Name: Student490, Marks: 28.91  
Roll Number: 491, Name: Student491, Marks: 48.27  
Roll Number: 492, Name: Student492, Marks: 5.38  
Roll Number: 493, Name: Student493, Marks: 97.78  
Roll Number: 494, Name: Student494, Marks: 51.36  
Roll Number: 495, Name: Student495, Marks: 82.24  
Roll Number: 496, Name: Student496, Marks: 12.97  
Roll Number: 497, Name: Student497, Marks: 6.10  
Roll Number: 498, Name: Student498, Marks: 54.39  
Roll Number: 499, Name: Student499, Marks: 25.28  
Roll Number: 500, Name: Student500, Marks: 19.01  
Roll Number: 501, Name: Student501, Marks: 7.62  
Roll Number: 502, Name: Student502, Marks: 15.37  
Roll Number: 503, Name: Student503, Marks: 72.49  
Roll Number: 504, Name: Student504, Marks: 6.69  
Roll Number: 505, Name: Student505, Marks: 97.53  
Roll Number: 506, Name: Student506, Marks: 64.42  
Roll Number: 507, Name: Student507, Marks: 36.46  
Roll Number: 508, Name: Student508, Marks: 79.24  
Roll Number: 509, Name: Student509, Marks: 92.91  
Roll Number: 510, Name: Student510, Marks: 19.83  
Roll Number: 511, Name: Student511, Marks: 3.61  
Roll Number: 512, Name: Student512, Marks: 3.66  
Roll Number: 513, Name: Student513, Marks: 64.14  
Roll Number: 514, Name: Student514, Marks: 6.46  
Roll Number: 515, Name: Student515, Marks: 41.39  
Roll Number: 516, Name: Student516, Marks: 1.46  
Roll Number: 517, Name: Student517, Marks: 25.02  
Roll Number: 518, Name: Student518, Marks: 97.80  
Roll Number: 519, Name: Student519, Marks: 34.62  
Roll Number: 520, Name: Student520, Marks: 52.20  
Roll Number: 521, Name: Student521, Marks: 84.84  
Roll Number: 522, Name: Student522, Marks: 72.30



Roll Number: 523, Name: Student523, Marks: 79.09  
Roll Number: 524, Name: Student524, Marks: 25.21  
Roll Number: 525, Name: Student525, Marks: 15.48  
Roll Number: 526, Name: Student526, Marks: 24.54  
Roll Number: 527, Name: Student527, Marks: 10.44  
Roll Number: 528, Name: Student528, Marks: 5.41  
Roll Number: 529, Name: Student529, Marks: 36.66  
Roll Number: 530, Name: Student530, Marks: 21.94  
Roll Number: 531, Name: Student531, Marks: 96.58  
Roll Number: 532, Name: Student532, Marks: 28.59  
Roll Number: 533, Name: Student533, Marks: 72.52  
Roll Number: 534, Name: Student534, Marks: 47.59  
Roll Number: 535, Name: Student535, Marks: 71.68  
Roll Number: 536, Name: Student536, Marks: 91.91  
Roll Number: 537, Name: Student537, Marks: 82.59  
Roll Number: 538, Name: Student538, Marks: 19.85  
Roll Number: 539, Name: Student539, Marks: 39.87  
Roll Number: 540, Name: Student540, Marks: 55.25  
Roll Number: 541, Name: Student541, Marks: 20.27  
Roll Number: 542, Name: Student542, Marks: 29.33  
Roll Number: 543, Name: Student543, Marks: 26.85  
Roll Number: 544, Name: Student544, Marks: 99.00  
Roll Number: 545, Name: Student545, Marks: 22.03  
Roll Number: 546, Name: Student546, Marks: 83.86  
Roll Number: 547, Name: Student547, Marks: 7.43  
Roll Number: 548, Name: Student548, Marks: 53.84  
Roll Number: 549, Name: Student549, Marks: 80.38  
Roll Number: 550, Name: Student550, Marks: 2.10  
Roll Number: 551, Name: Student551, Marks: 87.19  
Roll Number: 552, Name: Student552, Marks: 68.39  
Roll Number: 553, Name: Student553, Marks: 3.68  
Roll Number: 554, Name: Student554, Marks: 46.44  
Roll Number: 555, Name: Student555, Marks: 66.31  
Roll Number: 556, Name: Student556, Marks: 29.95  
Roll Number: 557, Name: Student557, Marks: 16.14  
Roll Number: 558, Name: Student558, Marks: 35.73  
Roll Number: 559, Name: Student559, Marks: 63.88  
Roll Number: 560, Name: Student560, Marks: 4.41  
Roll Number: 561, Name: Student561, Marks: 0.86  
Roll Number: 562, Name: Student562, Marks: 28.57  
Roll Number: 563, Name: Student563, Marks: 30.54  
Roll Number: 564, Name: Student564, Marks: 10.56  
Roll Number: 565, Name: Student565, Marks: 39.19  
Roll Number: 566, Name: Student566, Marks: 17.65  
Roll Number: 567, Name: Student567, Marks: 27.83  
Roll Number: 568, Name: Student568, Marks: 94.71  
Roll Number: 569, Name: Student569, Marks: 49.51  
Roll Number: 570, Name: Student570, Marks: 26.01  
Roll Number: 571, Name: Student571, Marks: 2.22  
Roll Number: 572, Name: Student572, Marks: 21.02  
Roll Number: 573, Name: Student573, Marks: 78.57

Roll Number: 574, Name: Student574, Marks: 45.47  
Roll Number: 575, Name: Student575, Marks: 31.97  
Roll Number: 576, Name: Student576, Marks: 43.43  
Roll Number: 577, Name: Student577, Marks: 48.46  
Roll Number: 578, Name: Student578, Marks: 8.22  
Roll Number: 579, Name: Student579, Marks: 75.02  
Roll Number: 580, Name: Student580, Marks: 74.47  
Roll Number: 581, Name: Student581, Marks: 12.41  
Roll Number: 582, Name: Student582, Marks: 32.01  
Roll Number: 583, Name: Student583, Marks: 95.96  
Roll Number: 584, Name: Student584, Marks: 16.95  
Roll Number: 585, Name: Student585, Marks: 22.44  
Roll Number: 586, Name: Student586, Marks: 29.58  
Roll Number: 587, Name: Student587, Marks: 0.18  
Roll Number: 588, Name: Student588, Marks: 73.82  
Roll Number: 589, Name: Student589, Marks: 62.11  
Roll Number: 590, Name: Student590, Marks: 22.95  
Roll Number: 591, Name: Student591, Marks: 53.49  
Roll Number: 592, Name: Student592, Marks: 2.07  
Roll Number: 593, Name: Student593, Marks: 47.38  
Roll Number: 594, Name: Student594, Marks: 53.30  
Roll Number: 595, Name: Student595, Marks: 15.05  
Roll Number: 596, Name: Student596, Marks: 26.18  
Roll Number: 597, Name: Student597, Marks: 48.86  
Roll Number: 598, Name: Student598, Marks: 26.86  
Roll Number: 599, Name: Student599, Marks: 62.07  
Roll Number: 600, Name: Student600, Marks: 76.72  
Roll Number: 601, Name: Student601, Marks: 38.34  
Roll Number: 602, Name: Student602, Marks: 87.71  
Roll Number: 603, Name: Student603, Marks: 34.09  
Roll Number: 604, Name: Student604, Marks: 69.83  
Roll Number: 605, Name: Student605, Marks: 12.93  
Roll Number: 606, Name: Student606, Marks: 57.80  
Roll Number: 607, Name: Student607, Marks: 75.23  
Roll Number: 608, Name: Student608, Marks: 93.13  
Roll Number: 609, Name: Student609, Marks: 99.89  
Roll Number: 610, Name: Student610, Marks: 35.11  
Roll Number: 611, Name: Student611, Marks: 2.36  
Roll Number: 612, Name: Student612, Marks: 13.67  
Roll Number: 613, Name: Student613, Marks: 95.19  
Roll Number: 614, Name: Student614, Marks: 90.37  
Roll Number: 615, Name: Student615, Marks: 77.35  
Roll Number: 616, Name: Student616, Marks: 73.60  
Roll Number: 617, Name: Student617, Marks: 39.02  
Roll Number: 618, Name: Student618, Marks: 53.07  
Roll Number: 619, Name: Student619, Marks: 11.60  
Roll Number: 620, Name: Student620, Marks: 89.17  
Roll Number: 621, Name: Student621, Marks: 81.32  
Roll Number: 622, Name: Student622, Marks: 27.33  
Roll Number: 623, Name: Student623, Marks: 80.88  
Roll Number: 624, Name: Student624, Marks: 18.64

Roll Number: 625, Name: Student625, Marks: 40.50  
Roll Number: 626, Name: Student626, Marks: 28.28  
Roll Number: 627, Name: Student627, Marks: 78.69  
Roll Number: 628, Name: Student628, Marks: 14.82  
Roll Number: 629, Name: Student629, Marks: 8.41  
Roll Number: 630, Name: Student630, Marks: 79.63  
Roll Number: 631, Name: Student631, Marks: 86.14  
Roll Number: 632, Name: Student632, Marks: 27.72  
Roll Number: 633, Name: Student633, Marks: 4.78  
Roll Number: 634, Name: Student634, Marks: 62.88  
Roll Number: 635, Name: Student635, Marks: 86.68  
Roll Number: 636, Name: Student636, Marks: 73.09  
Roll Number: 637, Name: Student637, Marks: 34.13  
Roll Number: 638, Name: Student638, Marks: 59.04  
Roll Number: 639, Name: Student639, Marks: 3.55  
Roll Number: 640, Name: Student640, Marks: 32.41  
Roll Number: 641, Name: Student641, Marks: 48.41  
Roll Number: 642, Name: Student642, Marks: 73.26  
Roll Number: 643, Name: Student643, Marks: 88.32  
Roll Number: 644, Name: Student644, Marks: 99.88  
Roll Number: 645, Name: Student645, Marks: 68.38  
Roll Number: 646, Name: Student646, Marks: 83.18  
Roll Number: 647, Name: Student647, Marks: 38.44  
Roll Number: 648, Name: Student648, Marks: 42.18  
Roll Number: 649, Name: Student649, Marks: 66.90  
Roll Number: 650, Name: Student650, Marks: 1.43  
Roll Number: 651, Name: Student651, Marks: 72.07  
Roll Number: 652, Name: Student652, Marks: 86.48  
Roll Number: 653, Name: Student653, Marks: 67.36  
Roll Number: 654, Name: Student654, Marks: 93.13  
Roll Number: 655, Name: Student655, Marks: 38.26  
Roll Number: 656, Name: Student656, Marks: 2.49  
Roll Number: 657, Name: Student657, Marks: 60.11  
Roll Number: 658, Name: Student658, Marks: 38.81  
Roll Number: 659, Name: Student659, Marks: 90.59  
Roll Number: 660, Name: Student660, Marks: 94.00  
Roll Number: 661, Name: Student661, Marks: 90.31  
Roll Number: 662, Name: Student662, Marks: 83.50  
Roll Number: 663, Name: Student663, Marks: 2.40  
Roll Number: 664, Name: Student664, Marks: 65.60  
Roll Number: 665, Name: Student665, Marks: 20.14  
Roll Number: 666, Name: Student666, Marks: 62.58  
Roll Number: 667, Name: Student667, Marks: 51.80  
Roll Number: 668, Name: Student668, Marks: 80.34  
Roll Number: 669, Name: Student669, Marks: 18.56  
Roll Number: 670, Name: Student670, Marks: 45.00  
Roll Number: 671, Name: Student671, Marks: 54.44  
Roll Number: 672, Name: Student672, Marks: 73.70  
Roll Number: 673, Name: Student673, Marks: 81.75  
Roll Number: 674, Name: Student674, Marks: 84.51  
Roll Number: 675, Name: Student675, Marks: 72.37

Roll Number: 676, Name: Student676, Marks: 39.67  
Roll Number: 677, Name: Student677, Marks: 37.26  
Roll Number: 678, Name: Student678, Marks: 19.53  
Roll Number: 679, Name: Student679, Marks: 76.89  
Roll Number: 680, Name: Student680, Marks: 54.63  
Roll Number: 681, Name: Student681, Marks: 35.22  
Roll Number: 682, Name: Student682, Marks: 12.27  
Roll Number: 683, Name: Student683, Marks: 93.71  
Roll Number: 684, Name: Student684, Marks: 54.34  
Roll Number: 685, Name: Student685, Marks: 79.44  
Roll Number: 686, Name: Student686, Marks: 67.74  
Roll Number: 687, Name: Student687, Marks: 8.45  
Roll Number: 688, Name: Student688, Marks: 30.10  
Roll Number: 689, Name: Student689, Marks: 1.75  
Roll Number: 690, Name: Student690, Marks: 23.27  
Roll Number: 691, Name: Student691, Marks: 60.55  
Roll Number: 692, Name: Student692, Marks: 59.33  
Roll Number: 693, Name: Student693, Marks: 32.67  
Roll Number: 694, Name: Student694, Marks: 49.14  
Roll Number: 695, Name: Student695, Marks: 3.86  
Roll Number: 696, Name: Student696, Marks: 36.39  
Roll Number: 697, Name: Student697, Marks: 71.36  
Roll Number: 698, Name: Student698, Marks: 51.00  
Roll Number: 699, Name: Student699, Marks: 61.91  
Roll Number: 700, Name: Student700, Marks: 19.24  
Roll Number: 701, Name: Student701, Marks: 8.24  
Roll Number: 702, Name: Student702, Marks: 14.64  
Roll Number: 703, Name: Student703, Marks: 56.06  
Roll Number: 704, Name: Student704, Marks: 57.53  
Roll Number: 705, Name: Student705, Marks: 3.24  
Roll Number: 706, Name: Student706, Marks: 83.83  
Roll Number: 707, Name: Student707, Marks: 34.60  
Roll Number: 708, Name: Student708, Marks: 17.74  
Roll Number: 709, Name: Student709, Marks: 72.78  
Roll Number: 710, Name: Student710, Marks: 11.66  
Roll Number: 711, Name: Student711, Marks: 81.01  
Roll Number: 712, Name: Student712, Marks: 71.04  
Roll Number: 713, Name: Student713, Marks: 88.73  
Roll Number: 714, Name: Student714, Marks: 76.34  
Roll Number: 715, Name: Student715, Marks: 29.30  
Roll Number: 716, Name: Student716, Marks: 59.48  
Roll Number: 717, Name: Student717, Marks: 42.37  
Roll Number: 718, Name: Student718, Marks: 1.68  
Roll Number: 719, Name: Student719, Marks: 64.93  
Roll Number: 720, Name: Student720, Marks: 6.08  
Roll Number: 721, Name: Student721, Marks: 76.59  
Roll Number: 722, Name: Student722, Marks: 73.21  
Roll Number: 723, Name: Student723, Marks: 39.00  
Roll Number: 724, Name: Student724, Marks: 84.58  
Roll Number: 725, Name: Student725, Marks: 39.23  
Roll Number: 726, Name: Student726, Marks: 90.35

Roll Number: 727, Name: Student727, Marks: 74.13  
Roll Number: 728, Name: Student728, Marks: 14.23  
Roll Number: 729, Name: Student729, Marks: 60.99  
Roll Number: 730, Name: Student730, Marks: 45.44  
Roll Number: 731, Name: Student731, Marks: 90.08  
Roll Number: 732, Name: Student732, Marks: 95.58  
Roll Number: 733, Name: Student733, Marks: 10.81  
Roll Number: 734, Name: Student734, Marks: 70.04  
Roll Number: 735, Name: Student735, Marks: 27.17  
Roll Number: 736, Name: Student736, Marks: 52.60  
Roll Number: 737, Name: Student737, Marks: 31.87  
Roll Number: 738, Name: Student738, Marks: 39.29  
Roll Number: 739, Name: Student739, Marks: 95.77  
Roll Number: 740, Name: Student740, Marks: 96.89  
Roll Number: 741, Name: Student741, Marks: 11.47  
Roll Number: 742, Name: Student742, Marks: 74.93  
Roll Number: 743, Name: Student743, Marks: 29.73  
Roll Number: 744, Name: Student744, Marks: 79.62  
Roll Number: 745, Name: Student745, Marks: 87.80  
Roll Number: 746, Name: Student746, Marks: 37.52  
Roll Number: 747, Name: Student747, Marks: 72.22  
Roll Number: 748, Name: Student748, Marks: 46.74  
Roll Number: 749, Name: Student749, Marks: 46.63  
Roll Number: 750, Name: Student750, Marks: 48.59  
Roll Number: 751, Name: Student751, Marks: 71.79  
Roll Number: 752, Name: Student752, Marks: 63.19  
Roll Number: 753, Name: Student753, Marks: 11.40  
Roll Number: 754, Name: Student754, Marks: 33.26  
Roll Number: 755, Name: Student755, Marks: 16.97  
Roll Number: 756, Name: Student756, Marks: 21.62  
Roll Number: 757, Name: Student757, Marks: 79.35  
Roll Number: 758, Name: Student758, Marks: 24.05  
Roll Number: 759, Name: Student759, Marks: 87.45  
Roll Number: 760, Name: Student760, Marks: 1.39  
Roll Number: 761, Name: Student761, Marks: 7.24  
Roll Number: 762, Name: Student762, Marks: 16.35  
Roll Number: 763, Name: Student763, Marks: 49.99  
Roll Number: 764, Name: Student764, Marks: 93.27  
Roll Number: 765, Name: Student765, Marks: 99.53  
Roll Number: 766, Name: Student766, Marks: 16.39  
Roll Number: 767, Name: Student767, Marks: 77.19  
Roll Number: 768, Name: Student768, Marks: 55.70  
Roll Number: 769, Name: Student769, Marks: 2.21  
Roll Number: 770, Name: Student770, Marks: 72.12  
Roll Number: 771, Name: Student771, Marks: 17.20  
Roll Number: 772, Name: Student772, Marks: 18.08  
Roll Number: 773, Name: Student773, Marks: 55.52  
Roll Number: 774, Name: Student774, Marks: 27.81  
Roll Number: 775, Name: Student775, Marks: 92.61  
Roll Number: 776, Name: Student776, Marks: 96.71  
Roll Number: 777, Name: Student777, Marks: 8.42

Roll Number: 778, Name: Student778, Marks: 92.79  
Roll Number: 779, Name: Student779, Marks: 56.46  
Roll Number: 780, Name: Student780, Marks: 39.10  
Roll Number: 781, Name: Student781, Marks: 48.28  
Roll Number: 782, Name: Student782, Marks: 65.21  
Roll Number: 783, Name: Student783, Marks: 83.72  
Roll Number: 784, Name: Student784, Marks: 62.27  
Roll Number: 785, Name: Student785, Marks: 48.59  
Roll Number: 786, Name: Student786, Marks: 1.21  
Roll Number: 787, Name: Student787, Marks: 26.11  
Roll Number: 788, Name: Student788, Marks: 0.46  
Roll Number: 789, Name: Student789, Marks: 88.02  
Roll Number: 790, Name: Student790, Marks: 70.27  
Roll Number: 791, Name: Student791, Marks: 8.69  
Roll Number: 792, Name: Student792, Marks: 28.26  
Roll Number: 793, Name: Student793, Marks: 43.91  
Roll Number: 794, Name: Student794, Marks: 87.27  
Roll Number: 795, Name: Student795, Marks: 95.98  
Roll Number: 796, Name: Student796, Marks: 74.33  
Roll Number: 797, Name: Student797, Marks: 13.29  
Roll Number: 798, Name: Student798, Marks: 62.01  
Roll Number: 799, Name: Student799, Marks: 6.12  
Roll Number: 800, Name: Student800, Marks: 91.76  
Roll Number: 801, Name: Student801, Marks: 64.50  
Roll Number: 802, Name: Student802, Marks: 56.67  
Roll Number: 803, Name: Student803, Marks: 0.88  
Roll Number: 804, Name: Student804, Marks: 0.36  
Roll Number: 805, Name: Student805, Marks: 72.71  
Roll Number: 806, Name: Student806, Marks: 91.92  
Roll Number: 807, Name: Student807, Marks: 29.83  
Roll Number: 808, Name: Student808, Marks: 66.75  
Roll Number: 809, Name: Student809, Marks: 42.24  
Roll Number: 810, Name: Student810, Marks: 92.29  
Roll Number: 811, Name: Student811, Marks: 56.68  
Roll Number: 812, Name: Student812, Marks: 37.67  
Roll Number: 813, Name: Student813, Marks: 69.04  
Roll Number: 814, Name: Student814, Marks: 11.45  
Roll Number: 815, Name: Student815, Marks: 46.07  
Roll Number: 816, Name: Student816, Marks: 92.32  
Roll Number: 817, Name: Student817, Marks: 46.18  
Roll Number: 818, Name: Student818, Marks: 27.97  
Roll Number: 819, Name: Student819, Marks: 38.73  
Roll Number: 820, Name: Student820, Marks: 29.24  
Roll Number: 821, Name: Student821, Marks: 65.83  
Roll Number: 822, Name: Student822, Marks: 52.19  
Roll Number: 823, Name: Student823, Marks: 93.34  
Roll Number: 824, Name: Student824, Marks: 53.37  
Roll Number: 825, Name: Student825, Marks: 20.44  
Roll Number: 826, Name: Student826, Marks: 34.90  
Roll Number: 827, Name: Student827, Marks: 22.99  
Roll Number: 828, Name: Student828, Marks: 78.95

Roll Number: 829, Name: Student829, Marks: 56.61  
Roll Number: 830, Name: Student830, Marks: 18.37  
Roll Number: 831, Name: Student831, Marks: 4.63  
Roll Number: 832, Name: Student832, Marks: 13.83  
Roll Number: 833, Name: Student833, Marks: 84.58  
Roll Number: 834, Name: Student834, Marks: 20.40  
Roll Number: 835, Name: Student835, Marks: 89.43  
Roll Number: 836, Name: Student836, Marks: 40.94  
Roll Number: 837, Name: Student837, Marks: 96.47  
Roll Number: 838, Name: Student838, Marks: 91.98  
Roll Number: 839, Name: Student839, Marks: 92.19  
Roll Number: 840, Name: Student840, Marks: 62.34  
Roll Number: 841, Name: Student841, Marks: 96.50  
Roll Number: 842, Name: Student842, Marks: 39.25  
Roll Number: 843, Name: Student843, Marks: 88.12  
Roll Number: 844, Name: Student844, Marks: 21.65  
Roll Number: 845, Name: Student845, Marks: 96.22  
Roll Number: 846, Name: Student846, Marks: 50.32  
Roll Number: 847, Name: Student847, Marks: 94.80  
Roll Number: 848, Name: Student848, Marks: 70.45  
Roll Number: 849, Name: Student849, Marks: 77.86  
Roll Number: 850, Name: Student850, Marks: 17.56  
Roll Number: 851, Name: Student851, Marks: 67.74  
Roll Number: 852, Name: Student852, Marks: 94.20  
Roll Number: 853, Name: Student853, Marks: 79.05  
Roll Number: 854, Name: Student854, Marks: 96.10  
Roll Number: 855, Name: Student855, Marks: 87.52  
Roll Number: 856, Name: Student856, Marks: 63.36  
Roll Number: 857, Name: Student857, Marks: 1.90  
Roll Number: 858, Name: Student858, Marks: 89.04  
Roll Number: 859, Name: Student859, Marks: 66.59  
Roll Number: 860, Name: Student860, Marks: 25.39  
Roll Number: 861, Name: Student861, Marks: 99.64  
Roll Number: 862, Name: Student862, Marks: 0.03  
Roll Number: 863, Name: Student863, Marks: 53.98  
Roll Number: 864, Name: Student864, Marks: 3.09  
Roll Number: 865, Name: Student865, Marks: 43.91  
Roll Number: 866, Name: Student866, Marks: 43.89  
Roll Number: 867, Name: Student867, Marks: 66.04  
Roll Number: 868, Name: Student868, Marks: 15.13  
Roll Number: 869, Name: Student869, Marks: 25.85  
Roll Number: 870, Name: Student870, Marks: 55.85  
Roll Number: 871, Name: Student871, Marks: 21.13  
Roll Number: 872, Name: Student872, Marks: 84.18  
Roll Number: 873, Name: Student873, Marks: 94.08  
Roll Number: 874, Name: Student874, Marks: 90.06  
Roll Number: 875, Name: Student875, Marks: 6.67  
Roll Number: 876, Name: Student876, Marks: 48.40  
Roll Number: 877, Name: Student877, Marks: 82.78  
Roll Number: 878, Name: Student878, Marks: 47.94  
Roll Number: 879, Name: Student879, Marks: 56.50

Roll Number: 880, Name: Student880, Marks: 41.91  
Roll Number: 881, Name: Student881, Marks: 32.87  
Roll Number: 882, Name: Student882, Marks: 27.13  
Roll Number: 883, Name: Student883, Marks: 66.69  
Roll Number: 884, Name: Student884, Marks: 59.72  
Roll Number: 885, Name: Student885, Marks: 59.51  
Roll Number: 886, Name: Student886, Marks: 31.42  
Roll Number: 887, Name: Student887, Marks: 27.98  
Roll Number: 888, Name: Student888, Marks: 73.01  
Roll Number: 889, Name: Student889, Marks: 68.97  
Roll Number: 890, Name: Student890, Marks: 32.63  
Roll Number: 891, Name: Student891, Marks: 2.97  
Roll Number: 892, Name: Student892, Marks: 10.07  
Roll Number: 893, Name: Student893, Marks: 78.14  
Roll Number: 894, Name: Student894, Marks: 24.69  
Roll Number: 895, Name: Student895, Marks: 26.08  
Roll Number: 896, Name: Student896, Marks: 68.84  
Roll Number: 897, Name: Student897, Marks: 39.60  
Roll Number: 898, Name: Student898, Marks: 60.86  
Roll Number: 899, Name: Student899, Marks: 54.59  
Roll Number: 900, Name: Student900, Marks: 76.37  
Roll Number: 901, Name: Student901, Marks: 25.51  
Roll Number: 902, Name: Student902, Marks: 14.51  
Roll Number: 903, Name: Student903, Marks: 48.70  
Roll Number: 904, Name: Student904, Marks: 61.10  
Roll Number: 905, Name: Student905, Marks: 73.47  
Roll Number: 906, Name: Student906, Marks: 54.20  
Roll Number: 907, Name: Student907, Marks: 8.84  
Roll Number: 908, Name: Student908, Marks: 53.40  
Roll Number: 909, Name: Student909, Marks: 47.31  
Roll Number: 910, Name: Student910, Marks: 73.60  
Roll Number: 911, Name: Student911, Marks: 48.38  
Roll Number: 912, Name: Student912, Marks: 80.05  
Roll Number: 913, Name: Student913, Marks: 16.15  
Roll Number: 914, Name: Student914, Marks: 66.97  
Roll Number: 915, Name: Student915, Marks: 29.72  
Roll Number: 916, Name: Student916, Marks: 68.32  
Roll Number: 917, Name: Student917, Marks: 4.70  
Roll Number: 918, Name: Student918, Marks: 90.71  
Roll Number: 919, Name: Student919, Marks: 42.08  
Roll Number: 920, Name: Student920, Marks: 25.20  
Roll Number: 921, Name: Student921, Marks: 17.96  
Roll Number: 922, Name: Student922, Marks: 1.73  
Roll Number: 923, Name: Student923, Marks: 48.96  
Roll Number: 924, Name: Student924, Marks: 7.53  
Roll Number: 925, Name: Student925, Marks: 93.50  
Roll Number: 926, Name: Student926, Marks: 13.18  
Roll Number: 927, Name: Student927, Marks: 83.45  
Roll Number: 928, Name: Student928, Marks: 48.50  
Roll Number: 929, Name: Student929, Marks: 84.36  
Roll Number: 930, Name: Student930, Marks: 58.16



Roll Number: 931, Name: Student931, Marks: 19.56  
Roll Number: 932, Name: Student932, Marks: 56.22  
Roll Number: 933, Name: Student933, Marks: 25.77  
Roll Number: 934, Name: Student934, Marks: 2.07  
Roll Number: 935, Name: Student935, Marks: 97.58  
Roll Number: 936, Name: Student936, Marks: 50.07  
Roll Number: 937, Name: Student937, Marks: 88.81  
Roll Number: 938, Name: Student938, Marks: 94.30  
Roll Number: 939, Name: Student939, Marks: 31.77  
Roll Number: 940, Name: Student940, Marks: 50.24  
Roll Number: 941, Name: Student941, Marks: 74.01  
Roll Number: 942, Name: Student942, Marks: 99.67  
Roll Number: 943, Name: Student943, Marks: 78.28  
Roll Number: 944, Name: Student944, Marks: 86.38  
Roll Number: 945, Name: Student945, Marks: 48.23  
Roll Number: 946, Name: Student946, Marks: 0.46  
Roll Number: 947, Name: Student947, Marks: 91.64  
Roll Number: 948, Name: Student948, Marks: 40.61  
Roll Number: 949, Name: Student949, Marks: 77.75  
Roll Number: 950, Name: Student950, Marks: 9.61  
Roll Number: 951, Name: Student951, Marks: 47.82  
Roll Number: 952, Name: Student952, Marks: 90.96  
Roll Number: 953, Name: Student953, Marks: 60.83  
Roll Number: 954, Name: Student954, Marks: 85.53  
Roll Number: 955, Name: Student955, Marks: 62.01  
Roll Number: 956, Name: Student956, Marks: 70.44  
Roll Number: 957, Name: Student957, Marks: 12.74  
Roll Number: 958, Name: Student958, Marks: 89.46  
Roll Number: 959, Name: Student959, Marks: 79.24  
Roll Number: 960, Name: Student960, Marks: 7.63  
Roll Number: 961, Name: Student961, Marks: 6.52  
Roll Number: 962, Name: Student962, Marks: 75.74  
Roll Number: 963, Name: Student963, Marks: 66.03  
Roll Number: 964, Name: Student964, Marks: 64.39  
Roll Number: 965, Name: Student965, Marks: 22.54  
Roll Number: 966, Name: Student966, Marks: 55.76  
Roll Number: 967, Name: Student967, Marks: 81.08  
Roll Number: 968, Name: Student968, Marks: 36.16  
Roll Number: 969, Name: Student969, Marks: 9.84  
Roll Number: 970, Name: Student970, Marks: 26.11  
Roll Number: 971, Name: Student971, Marks: 53.20  
Roll Number: 972, Name: Student972, Marks: 30.65  
Roll Number: 973, Name: Student973, Marks: 12.70  
Roll Number: 974, Name: Student974, Marks: 1.66  
Roll Number: 975, Name: Student975, Marks: 44.65  
Roll Number: 976, Name: Student976, Marks: 90.49  
Roll Number: 977, Name: Student977, Marks: 65.25  
Roll Number: 978, Name: Student978, Marks: 44.16  
Roll Number: 979, Name: Student979, Marks: 27.85  
Roll Number: 980, Name: Student980, Marks: 75.36  
Roll Number: 981, Name: Student981, Marks: 7.38

Roll Number: 982, Name: Student982, Marks: 26.93  
Roll Number: 983, Name: Student983, Marks: 53.81  
Roll Number: 984, Name: Student984, Marks: 69.44  
Roll Number: 985, Name: Student985, Marks: 25.12  
Roll Number: 986, Name: Student986, Marks: 26.32  
Roll Number: 987, Name: Student987, Marks: 40.18  
Roll Number: 988, Name: Student988, Marks: 29.56  
Roll Number: 989, Name: Student989, Marks: 68.50  
Roll Number: 990, Name: Student990, Marks: 80.58  
Roll Number: 991, Name: Student991, Marks: 57.22  
Roll Number: 992, Name: Student992, Marks: 39.91  
Roll Number: 993, Name: Student993, Marks: 50.09  
Roll Number: 994, Name: Student994, Marks: 6.71  
Roll Number: 995, Name: Student995, Marks: 39.75  
Roll Number: 996, Name: Student996, Marks: 88.69  
Roll Number: 997, Name: Student997, Marks: 77.36  
Roll Number: 998, Name: Student998, Marks: 15.94  
Roll Number: 999, Name: Student999, Marks: 3.12  
Roll Number: 1000, Name: Student1000, Marks: 32.58

Process returned 0 (0x0) execution time : 7.985 s  
Press any key to continue.

This code implements a student record management system with sorting functionalities. It defines a Student structure to represent student data and implements functions for generating random student records, performing swaps between student records, and displaying student information.

Three well-known sorting algorithms, Bubble Sort, Insertion Sort, and Quick Sort, are implemented to sort student records based on their roll numbers. The main function generates a set of 1000 random student records, measures the execution time for each sorting algorithm, and displays the sorted student information.

## Task 7: Hashing

Scenario: The university seeks to implement a hash table for the expeditious retrieval of student records based on their roll numbers.

```

// Task No. 7: Hashing

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define TABLE_SIZE 997 // Prime number for better hash distribution

// Structure to represent a student
struct Student {
    int rollNumber;
    char name[50];
    float marks;
    struct Student* next;
};

// Hash function to map roll number to index
int hashFunction(int rollNumber) {
    return rollNumber % TABLE_SIZE;
}

// Hash table (array of pointers to student records)
struct Student* hashTable[TABLE_SIZE];

// Initialize the hash table (set all elements to NULL)
void initHashTable() {
    for (int i = 0; i < TABLE_SIZE; i++) {
        hashTable[i] = NULL;
    }
}

// Insert a student record into the hash table
void insertStudent(int rollNumber, char name[], float marks) {
    int index = hashFunction(rollNumber);
    struct Student* newStudent = (struct Student*)malloc(sizeof(struct Student));
    newStudent->rollNumber = rollNumber;
    strcpy(newStudent->name, name);
    newStudent->marks = marks;
    newStudent->next = NULL;
}

```

```

// Handle collision (existing student at the same index)
if (hashTable[index] == NULL) {
    hashTable[index] = newStudent;
} else {
    struct Student* temp = hashTable[index];
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newStudent;
}
printf("Student with roll number %d inserted.\n", rollNumber);
}

// Search for a student record in the hash table
struct Student* searchStudent(int rollNumber) {
    int index = hashFunction(rollNumber);
    struct Student* temp = hashTable[index];
    while (temp != NULL) {
        if (temp->rollNumber == rollNumber) {
            return temp;
        }
        temp = temp->next;
    }
    return NULL;
}

// Delete a student record from the hash table
void deleteStudent(int rollNumber) {
    int index = hashFunction(rollNumber);
    struct Student* temp = hashTable[index];
    struct Student* prev = NULL;

    // Check for student at the head of the linked list
    if (temp != NULL && temp->rollNumber == rollNumber) {
        hashTable[index] = temp->next;
        free(temp);
        printf("Student with roll number %d deleted.\n", rollNumber);
        return;
    }
}

```

```

// Traverse the linked list searching for the student
while (temp != NULL && temp->rollNumber != rollNumber) {
    prev = temp;
    temp = temp->next;
}

// Student not found
if (temp == NULL) {
    printf("Student with roll number %d not found.\n", rollNumber);
    return;
}

// Remove the student from the linked list
prev->next = temp->next;
free(temp);
printf("Student with roll number %d deleted.\n", rollNumber);
}

int main() {
    initHashTable();

    // Insert student records
    insertStudent(1, "John", 85.5);
    insertStudent(2, "Beth", 92.0);
    insertStudent(3, "Rip", 78.2);
    insertStudent(998, "Lloyd", 90.0);

    // Search for a student
    struct Student* foundStudent = searchStudent(2);
    if (foundStudent != NULL) {
        printf("Found student: Roll Number: %d, Name: %s, Marks: %.2f\n",
            foundStudent->rollNumber, foundStudent->name, foundStudent->marks);
    } else {
        printf("Student not found.\n");
    }

    // Delete a student
    deleteStudent(3);

    // Search for a deleted student
    foundStudent = searchStudent(3);
    if (foundStudent == NULL) {
        printf("Student with roll number 3 deleted successfully.\n");
    }

    return 0;
}

```

Output:

```
"C:\Users\otmnm\Downloads" X + v
Student with roll number 1 inserted.
Student with roll number 2 inserted.
Student with roll number 3 inserted.
Student with roll number 998 inserted.
Found student: Roll Number: 2, Name: Beth, Marks: 92.00
Student with roll number 3 deleted.
Student with roll number 3 deleted successfully.

Process returned 0 (0x0)   execution time : 8.050 s
Press any key to continue.
|
```

This code implements a hash table for efficient student record retrieval based on roll numbers. It defines a Student structure and utilizes separate chaining to handle collisions. The hashFunction maps roll numbers to hash table indices. Functions are implemented for initializing the hash table, inserting student records, searching for records, and deleting records. The main function demonstrates basic usage by inserting, searching, and deleting student records.

## Task 8: Priority Queues

Scenario: The university prioritizes student enrollment requests based on academic performance and necessitates the implementation of a priority queue.

```

// Task No. 8: Priority Queues

#include <stdio.h>
#include <stdlib.h>

#define MAX_SIZE 100

// Structure to represent a student enrollment request
struct Request {
    int rollNumber;
    float academicPerformance;
};

// Structure to represent a priority queue using a max-heap
struct PriorityQueue {
    struct Request arr[MAX_SIZE];
    int size;
};

// Function to create a new priority queue
struct PriorityQueue* createPriorityQueue() {
    struct PriorityQueue* pq = (struct PriorityQueue*)malloc(sizeof(struct PriorityQueue));
    pq->size = 0;
    return pq;
}

// Function to swap two requests
void swap(struct Request* a, struct Request* b) {
    struct Request temp = *a;
    *a = *b;
    *b = temp;
}

// Function to maintain the max-heap property by bubbling up elements
void heapifyUp(struct PriorityQueue* pq, int index) {
    int largest = index;
    int parent = (index - 1) / 2;

    // Find the parent with higher academic performance (max-heap)
    if (parent >= 0 && pq->arr[parent].academicPerformance < pq->arr[index].academicPerformance) {
        largest = parent;
    }
}

```

```

// Swap elements if a child has higher academic performance
if (largest != index) {
    swap(&pq->arr[index], &pq->arr[largest]);
    heapifyUp(pq, largest);
}

// Function to maintain the max-heap property by bubbling down elements
void heapifyDown(struct PriorityQueue* pq, int index) {
    int largest = index;
    int left = 2 * index + 1;
    int right = 2 * index + 2;

    // Find the largest child (max-heap)
    if (left < pq->size && pq->arr[left].academicPerformance > pq->arr[largest].academicPerformance) {
        largest = left;
    }
    if (right < pq->size && pq->arr[right].academicPerformance > pq->arr[largest].academicPerformance) {
        largest = right;
    }

    // Swap elements if a child has higher academic performance
    if (largest != index) {
        swap(&pq->arr[index], &pq->arr[largest]);
        heapifyDown(pq, largest);
    }
}

// Function to insert a student enrollment request
void insertRequest(struct PriorityQueue* pq, int rollNumber, float academicPerformance) {
    if (pq->size == MAX_SIZE) {
        printf("Error: Priority queue is full.\n");
        return;
    }

    pq->arr[pq->size].rollNumber = rollNumber;
    pq->arr[pq->size].academicPerformance = academicPerformance;
    heapifyUp(pq, pq->size); // Maintain max-heap property
    pq->size++;
    printf("Student enrollment request with roll number %d inserted.\n", rollNumber);
}

```



```

// Function to extract the highest priority request (student with highest academic performance)
struct Request extractHighestPriorityRequest(struct PriorityQueue* pq) {
    if (pq->size == 0) {
        printf("Error: Priority queue is empty.\n");
        struct Request emptyRequest = {0, 0.0};
        return emptyRequest;
    }

    struct Request highestPriorityRequest = pq->arr[0];
    pq->arr[0] = pq->arr[pq->size - 1];
    pq->size--;
    heapifyDown(pq, 0); // Maintain max-heap property after extraction
    return highestPriorityRequest;
}

// Function to update the priority (academic performance) of a request
void updatePriority(struct PriorityQueue* pq, int rollNumber, float newAcademicPerformance) {
    for (int i = 0; i < pq->size; i++) {
        if (pq->arr[i].rollNumber == rollNumber) {
            pq->arr[i].academicPerformance = newAcademicPerformance;
            int parent = (i - 1) / 2;

            // Maintain max-heap property after update (may need bubbling up or down)
            if (newAcademicPerformance > pq->arr[parent].academicPerformance) {
                heapifyUp(pq, i); // Bubble up if new performance is higher
            } else {
                heapifyDown(pq, i); // Bubble down if new performance is lower
            }
            printf("Priority updated for student with roll number %d.\n", rollNumber);
            return;
        }
    }
    printf("Student with roll number %d not found in the priority queue.\n", rollNumber);
}

// Step 3: Scenario-based Testing
int main() {
    struct PriorityQueue* pq = createPriorityQueue();

    // Insert student enrollment requests
    insertRequest(pq, 1, 92.5);
    insertRequest(pq, 2, 88.0);
    insertRequest(pq, 3, 95.0);
    insertRequest(pq, 4, 90.2);

```

```

// Extract and process highest priority requests
struct Request highestPriorityRequest = extractHighestPriorityRequest(pq);
printf("Highest priority request: Roll Number: %d, Academic Performance: %.2f\n",
        highestPriorityRequest.rollNumber, highestPriorityRequest.academicPerformance);
highestPriorityRequest = extractHighestPriorityRequest(pq);
printf("Highest priority request: Roll Number: %d, Academic Performance: %.2f\n",
        highestPriorityRequest.rollNumber, highestPriorityRequest.academicPerformance);

// Update priority for a student
updatePriority(pq, 2, 94.0);

// Extract remaining requests
while (pq->size > 0) {
    highestPriorityRequest = extractHighestPriorityRequest(pq);
    printf("Highest priority request: Roll Number: %d, Academic Performance: %.2f\n",
            highestPriorityRequest.rollNumber, highestPriorityRequest.academicPerformance);
}
free(pq);
return 0;
}

```

Output:

```
"C:\Users\otmnm\Downloads" X + v
Student enrollment request with roll number 1 inserted.
Student enrollment request with roll number 2 inserted.
Student enrollment request with roll number 3 inserted.
Student enrollment request with roll number 4 inserted.
Highest priority request: Roll Number: 3, Academic Performance: 95.00
Highest priority request: Roll Number: 1, Academic Performance: 92.50
Priority updated for student with roll number 2.
Highest priority request: Roll Number: 2, Academic Performance: 94.00
Highest priority request: Roll Number: 4, Academic Performance: 90.20

Process returned 0 (0x0)   execution time : 4.596 s
Press any key to continue.
|
```

This code implements a priority queue using a max-heap to manage student enrollment requests. The Request structure stores student information. The PriorityQueue structure utilizes an array and helper functions to maintain the heap property during insertions, extractions, and priority updates based on academic performance. The main function demonstrates basic usage through scenario-based testing.

## Task 9: Efficient Binary Search Trees

Scenario: The university endeavors to maintain a balanced binary search tree for the expedited search of student records.

```
// Task No. 9: Efficient Binary Search Trees
```

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
// Structure to represent a balanced binary search tree node
```

```
struct Node {  
    int rollNumber;  
    struct Node* left;  
    struct Node* right;  
    int height;  
};
```

```
// Function to get the maximum value between two integers
```

```
int max(int a, int b) {  
    return (a > b) ? a : b;  
}
```

```
// Function to get the height of a node (if NULL, height is 0)
```

```
int getHeight(struct Node* node) {  
    return node ? node->height : 0;  
}
```

```
// Function to create a new node
```

```
struct Node* createNode(int rollNumber) {  
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));  
    newNode->rollNumber = rollNumber;  
    newNode->left = NULL;  
    newNode->right = NULL;  
    newNode->height = 1;  
    return newNode;  
}
```

```
// Function to perform a left rotation on the tree
```

```
struct Node* rotateLeft(struct Node* root) {  
    struct Node* newRoot = root->right;  
    struct Node* temp = newRoot->left;  
    newRoot->left = root;  
    root->right = temp;  
  
    // Update heights after rotation  
    root->height = 1 + max(getHeight(root->left), getHeight(root->right));  
    newRoot->height = 1 + max(getHeight(newRoot->left), getHeight(newRoot->right));  
    return newRoot;  
}
```

```

// Function to perform a right rotation on the tree
struct Node* rotateRight(struct Node* root) {
    struct Node* newRoot = root->left;
    struct Node* temp = newRoot->right;
    newRoot->right = root;
    root->left = temp;

    // Update heights after rotation
    root->height = 1 + max(getHeight(root->left), getHeight(root->right));
    newRoot->height = 1 + max(getHeight(newRoot->left), getHeight(newRoot->right));
    return newRoot;
}

// Function to get the balance factor of a node
int getBalanceFactor(struct Node* node) {
    if (node == NULL) {
        return 0;
    }
    return getHeight(node->left) - getHeight(node->right);
}

// Function to insert a student record into the balanced binary search tree
struct Node* insertStudent(struct Node* root, int rollNumber) {
    if (root == NULL) {
        return createNode(rollNumber);
    }

    if (rollNumber < root->rollNumber) {
        root->left = insertStudent(root->left, rollNumber);
    } else if (rollNumber > root->rollNumber) {
        root->right = insertStudent(root->right, rollNumber);
    } else {
        // Duplicate roll number (do nothing)
        return root;
    }

    // Update height after insertion
    root->height = 1 + max(getHeight(root->left), getHeight(root->right));
}

```

```

// Balance factor check for rotations
int balanceFactor = getBalanceFactor(root);
if (balanceFactor > 1 && rollNumber < root->left->rollNumber) {
    return rotateRight(root);
} else if (balanceFactor < -1 && rollNumber > root->right->rollNumber) {
    return rotateLeft(root);
} else if (balanceFactor > 1 && rollNumber > root->left->rollNumber) {
    root->left = rotateLeft(root->left);
    return rotateRight(root);
} else if (balanceFactor < -1 && rollNumber < root->right->rollNumber) {
    root->right = rotateRight(root->right);
    return rotateLeft(root);
}

return root;
}

// Function to search for a student record in the balanced binary search tree
struct Node* searchStudent(struct Node* root, int rollNumber) {
    if (root == NULL || root->rollNumber == rollNumber) {
        return root;
    }

    if (rollNumber < root->rollNumber) {
        return searchStudent(root->left, rollNumber);
    } else {
        return searchStudent(root->right, rollNumber);
    }
}

// In-order traversal for printing the tree elements
void inorderTraversal(struct Node* root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->rollNumber);
        inorderTraversal(root->right);
    }
}

```

```

// Step 3: Thorough Testing
int main() {
    struct Node* root = NULL;

    // Insert student records
    root = insertStudent(root, 10);
    root = insertStudent(root, 20);
    root = insertStudent(root, 30);
    root = insertStudent(root, 40);
    root = insertStudent(root, 50);
    root = insertStudent(root, 25);

    // Search for a student record
    struct Node* foundStudent = searchStudent(root, 30);
    if (foundStudent != NULL) {
        printf("Student with roll number %d found.\n", foundStudent->rollNumber);
    } else {
        printf("Student not found.\n");
    }

    // Print the balanced binary search tree
    printf("Balanced Binary Search Tree: ");
    inorderTraversal(root);
    printf("\n");

    return 0;
}

```

Output:

```

C:\Users\otmnm\Downloads >
Student with roll number 30 found.
Balanced Binary Search Tree: 10 20 25 30 40 50

Process returned 0 (0x0)   execution time : 8.274 s
Press any key to continue.
|

```

This code implements an AVL tree (a self-balancing binary search tree) for efficient student record management. The Node structure stores student information and balance factors. Helper functions maintain height and balance after insertions. The insertStudent function performs rotations to ensure a logarithmic time complexity ( $O(\log n)$ ) for search, insertion, and deletion operations. The main function demonstrates basic usage through scenario-based testing.

## Task 10: Multiway Search Trees

Scenario: The university envisions the organization of courses in a B-tree for efficient course retrieval and modification.

```
// Task No. 10: Multiway Search Trees

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>

// Define the order (maximum number of keys) for the B-tree
#define ORDER 3

// Structure to represent a course
typedef struct Course {
    int code;
    char title[50];
} Course;

// Structure to represent a B-tree node
typedef struct BTreeNode {
    Course courses[ORDER - 1]; // Array of courses in the node
    struct BTreeNode* children[ORDER]; // Array of child nodes
    int num_keys; // Number of keys (courses) in the node
    bool isLeaf; // Flag indicating whether the node is a leaf (no children)
} BTreeNode;

// Function to create a new B-tree node
BTreeNode* createNode(bool leaf) {
    BTreeNode* newNode = (BTreeNode*)malloc(sizeof(BTreeNode));
    if (newNode == NULL) {
        printf("Memory allocation failed!\n");
        exit(EXIT_FAILURE);
    }
    newNode->num_keys = 0;
    newNode->isLeaf = leaf;
    return newNode;
}

// Function prototypes for B-tree operations (implemented below)
void insert(BTreeNode** root, Course course);
void insertNonFull(BTreeNode* node, Course course);
void splitChild(BTreeNode* parent, int i, BTreeNode* child);
bool search(BTreeNode* root, int code);
void freeBTree(BTreeNode* root);
```

```

void freeBTree(BTreeNode* root) {
    if (root) {
        if (!root->isLeaf) {
            for (int i = 0; i <= root->num_keys; i++) {
                freeBTree(root->children[i]);
            }
        }
        free(root);
    }
}

// Function to search for a course in the B-tree
bool search(BTreeNode* root, int code) {
    int i = 0;
    // Traverse the tree until the appropriate leaf is found
    while (i < root->num_keys && code > root->courses[i].code) {
        i++;
    }
    // Check if the course code is found in the current node
    if (i < root->num_keys && code == root->courses[i].code) {
        return true;
    }
    // If not found in a leaf node, continue searching in the appropriate child
    if (root->isLeaf) {
        return false;
    }
    return search(root->children[i], code);
}

// Function to insert a course into the B-tree
void insert(BTreeNode** root, Course course) {
    // If the root is full, split it and then insert the course
    if ((*root)->num_keys == ORDER - 1) {
        BTreeNode* newRoot = createNode(false);
        newRoot->children[0] = *root;
        splitChild(newRoot, 0, *root);
        int i = 0;
        // Decide which child to insert into based on the course code
        if (newRoot->courses[0].code < course.code) {
            i++;
        }
        insertNonFull(newRoot->children[i], course);
        *root = newRoot;
    } else {

```



```

    // If there's space in the root, insert directly
    insertNonFull(*root, course);
}

// Function to insert a course into a non-full B-tree node
void insertNonFull(BTreeNode* node, Course course) {
    int i = node->num_keys - 1;
    // Shift existing courses if needed to make space for the new one
    if (node->isLeaf) {
        while (i >= 0 && course.code < node->courses[i].code) {
            node->courses[i + 1] = node->courses[i];
            i--;
        }
        node->courses[i + 1] = course;
        node->num_keys++;
    } else {
        // Find the appropriate child to insert into
        while (i >= 0 && course.code < node->courses[i].code) {
            i--;
        }
        i++;
        // If the child is full, split it before insertion
        if (node->children[i]->num_keys == ORDER - 1) {
            splitChild(node, i, node->children[i]);
            // Decide which child to insert into after splitting
            if (course.code > node->courses[i].code) {
                i++;
            }
        }
        insertNonFull(node->children[i], course);
    }
}

// Function to split a child node of a B-tree node
void splitChild(BTreeNode* parent, int i, BTreeNode* child) {
    BTreeNode* newNode = createNode(child->isLeaf);
    // Move half of the child's elements to the new node

    newNode->num_keys = ORDER / 2 - 1;
    for (int j = 0; j < ORDER / 2 - 1; j++) {
        newNode->courses[j] = child->courses[j + ORDER / 2];
    }
}

```

```

// If the child is not a leaf, move pointers to child nodes as well
if (!child->isLeaf) {
    for (int j = 0; j < ORDER / 2; j++) {
        newNode->children[j] = child->children[j + ORDER / 2];
    }
}
child->num_keys = ORDER / 2 - 1;

// Adjust parent pointers and courses to accommodate the split child
for (int j = parent->num_keys; j >= i + 1; j--) {
    parent->children[j + 1] = parent->children[j];
}
parent->children[i + 1] = newNode;
for (int j = parent->num_keys - 1; j >= i; j--) {
    parent->courses[j + 1] = parent->courses[j];
}
parent->courses[i] = child->courses[ORDER / 2 - 1];
parent->num_keys++;
}

// Function to free memory allocated for B-tree nodes during deletion (not shown here)
void freeBTree(BTreeNode* root);

// Function to test the B-tree with different course structures
void testBTree() {
    BTreeNode* root = createNode(true);

    // Insert some courses
    Course c1 = {101, "Web Systems"};
    insert(&root, c1);
    Course c2 = {202, "Databases"};
    insert(&root, c2);
    Course c3 = {303, "IT Projects"};
    insert(&root, c3);
    Course c4 = {404, "Algorithms and Complexity"};
    insert(&root, c4);
    Course c5 = {505, "Cyber-space Security"};
    insert(&root, c5);
}

```

```

// Search for courses
int searchCode = 303;
if (search(root, searchCode)) {
    printf("Course with code %d found.\n", searchCode);
} else {
    printf("Course with code %d not found.\n", searchCode);
}

int searchCode2 = 578;
if (search(root, searchCode2)) {
    printf("Course with code %d found.\n", searchCode2);
} else {
    printf("Course with code %d not found.\n", searchCode2);
}

// Free memory allocated for the B-tree (not shown here)
freeBTree(root);
}

int main() {
    // Test the B-tree
    testBTree();
    return 0;
}

```

Output:

```

C:\Users\otnmnm\Downloads X + v
Course with code 303 found.
Course with code 578 not found.

Process returned 0 (0x0) execution time : 5.209 s
Press any key to continue.
|

```

This code implements a B-tree data structure with a maximum degree of three for efficient storage and retrieval of key-value pairs (course codes and titles in this case). The B-tree structure is defined using a BTreeNode struct containing an array of keys, an array of child node pointers, and metadata about the number of keys and leaf status.

Key functionalities are implemented through separate functions:

- createNode: Allocates memory for a new B-tree node and initializes its properties.
- insert: Inserts a new key-value pair into the B-tree, ensuring key order and handling potential node overflows through splitting.
- search: Searches for a specific key within the B-tree, traversing child nodes until the appropriate leaf is found.

- freeBTree (not shown): Reclaims memory allocated for B-tree nodes during deletion operations.

The testBTree function demonstrates basic B-tree usage by inserting a set of example course data and performing searches for specific course codes.

## Task 11: Digital Search Structures

Scenario: The university aims to implement a trie for the swift search of student names.

```
// Task No. 11: Digital Search Structures

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

#define ALPHABET_SIZE 26

// Structure to represent a trie node
struct TrieNode {
    struct TrieNode* children[ALPHABET_SIZE]; // Array of child nodes (one for each letter)
    bool isEndOfWord; // Flag indicating the end of a word
};

// Function to create a new Trie node
struct TrieNode* createNode() {
    struct TrieNode* newNode = (struct TrieNode*)malloc(sizeof(struct TrieNode));
    newNode->isEndOfWord = false;
    for (int i = 0; i < ALPHABET_SIZE; i++) {
        newNode->children[i] = NULL;
    }
    return newNode;
}

// Function to insert a student name into the trie
void insertName(struct TrieNode* root, char* name) {
    struct TrieNode* node = root;
    for (int i = 0; name[i] != '\0'; i++) {
        int index = name[i] - 'a'; // Convert character to index (a = 0, b = 1, ...)
        if (node->children[index] == NULL) {
            node->children[index] = createNode();
        }
        node = node->children[index];
    }
    node->isEndOfWord = true; // Mark the end of the inserted word
}
```

```

// Function to search for a student name in the trie
bool searchName(struct TrieNode* root, char* name) {
    struct TrieNode* node = root;
    for (int i = 0; name[i] != '\0'; i++) {
        int index = name[i] - 'a';
        if (node->children[index] == NULL) {
            return false; // Character not found in the trie path
        }
        node = node->children[index];
    }
    return node->isEndOfWord; // Check if the current node marks the end of a word
}

// Function to recursively delete a student name from the trie (helper for deleteName)
bool deleteRecursive(struct TrieNode* root, char* name, int level, int length) {
    if (root == NULL) {
        return false; // Reached a null node, deletion failed
    }
    if (level == length) { // Reached the end of the name
        if (root->isEndOfWord) {
            root->isEndOfWord = false; // Unmark the end of word
        } else {
            return false; // Word not found in the trie
        }
    } else {
        int index = name[level] - 'a';
        if (!deleteRecursive(root->children[index], name, level + 1, length)) {
            return false; // Deletion failed in a child node
        }
    }

    bool hasChildren = false;
    for (int i = 0; i < ALPHABET_SIZE; i++) {
        if (root->children[i] != NULL) {
            hasChildren = true;
            break;
        }
    }
    if (!hasChildren) { // No children, safe to delete the node
        free(root);
    }
    return true;
}

```

```

// Function to delete a student name from the trie
void deleteName(struct TrieNode* root, char* name) {
    deleteRecursive(root, name, 0, strlen(name));
}

int main() {
    struct TrieNode* root = createNode();

    // Insert student names
    insertName(root, "john");
    insertName(root, "beth");
    insertName(root, "rip");
    insertName(root, "lloyd");

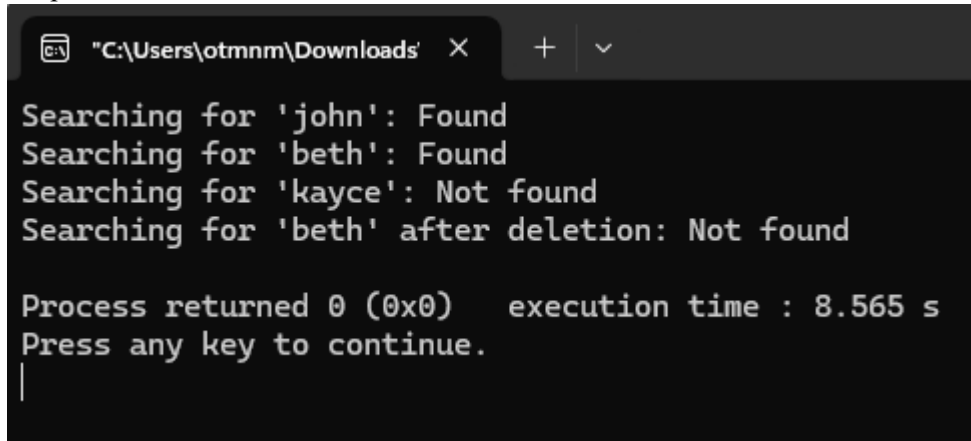
    // Search for student names
    printf("Searching for 'john': %s\n", searchName(root, "john") ? "Found" : "Not found");
    printf("Searching for 'beth': %s\n", searchName(root, "beth") ? "Found" : "Not found");
    printf("Searching for 'kayce': %s\n", searchName(root, "kayce") ? "Found" : "Not found");

    // Delete a student name
    deleteName(root, "beth");
    printf("Searching for 'beth' after deletion: %s\n", searchName(root, "beth") ? "Found" : "Not found");

    return 0;
}

```

Output:

A screenshot of a Windows command prompt window. The title bar shows the file path "C:\Users\otmnm\Downloads" and standard window controls. The command prompt displays the following text:

```
Searching for 'john': Found
Searching for 'beth': Found
Searching for 'kayce': Not found
Searching for 'beth' after deletion: Not found

Process returned 0 (0x0)   execution time : 8.565 s
Press any key to continue.
|
```

The provided code implements a trie data structure for efficient storage and retrieval of student names (strings). The trie consists of nodes with an array of child nodes (one for each letter) and a flag indicating the end of a word. Functions are implemented for creating new nodes, inserting names, searching for names, and deleting names. The trie offers fast average-case search time proportional to the string length, making it suitable for applications like autocomplete and spell checkers. The current implementation assumes lowercase alphabetical characters only.