Searching and Sorting --

Design a program in cpp to maintain a student database that performs the following tasks:

1. Add and store student details (ID, Name, CGPA) using dynamically allocated memory.

2. Expand the student list using realloc() as new entries are added.

3. Implement Linear Search to find student records by ID.

4. Implement Sorting Algorithms (Bubble Sort) to sort student records by:

o Name (Alphabetically)

o CGPA (Ascending/Descending)

……

```
#include <iostream>
#include <cstring>   // for strcmp()
#include <cstdlib>   // for malloc(), realloc(), free()

using namespace std;

struct Student {
    int id;
    char name[50];
    float cgpa;
};

// Function to add a new student
void addStudent(Student* &students, int &count) {
    students = (Student*) realloc(students, (count + 1) * sizeof(Student)); // 
expand memory

    cout << "\nEnter ID: ";
    cin >> students[count].id;
```

```
    cout << "Enter Name: ";
    cin >> students[count].name;
    cout << "Enter CGPA: ";
    cin >> students[count].cgpa;

    count++;
    cout << "Student added successfully!\n";
}

// Function to display all students
void displayStudents(Student* students, int count) {
    cout << "\n--- Student List ---\n";
    for (int i = 0; i < count; i++) {
        cout << "ID: " << students[i].id
             << "\tName: " << students[i].name
             << "\tCGPA: " << students[i].cgpa << endl;
    }
}

// Function for Linear Search
void searchByID(Student* students, int count, int key) {
    for (int i = 0; i < count; i++) {
        if (students[i].id == key) {
            cout << "\nStudent Found:\n";
            cout << "ID: " << students[i].id
                 << "\tName: " << students[i].name
```

```
            << "\tCGPA: " << students[i].cgpa << endl;
            return;
        }
    }
    cout << "\nStudent with ID " << key << " not found.\n";
}

// Bubble Sort by Name
void sortByName(Student* students, int count) {
    for (int i = 0; i < count - 1; i++) {
        for (int j = 0; j < count - i - 1; j++) {
            if (strcmp(students[j].name, students[j + 1].name) > 0) {
                swap(students[j], students[j + 1]);
            }
        }
    }
    cout << "\nSorted by Name Alphabetically!\n";
}

// Bubble Sort by CGPA (Ascending)
void sortByCGPA(Student* students, int count, bool ascending = true) {
    for (int i = 0; i < count - 1; i++) {
        for (int j = 0; j < count - i - 1; j++) {
            bool condition = ascending
                ? (students[j].cgpa > students[j + 1].cgpa)
                : (students[j].cgpa < students[j + 1].cgpa);
```

```
            if (condition) {
                swap(students[j], students[j + 1]);
            }
        }
    }
    cout << "\nSorted by CGPA (" << (ascending ? "Ascending" : "Descending") << ")!\n";
}

int main() {
    Student* students = NULL;
    int count = 0, choice;

    do {
        cout << "\n--- Student Database Menu ---\n";
        cout << "1. Add Student\n2. Display Students\n3. Search by ID\n4. Sort by Name\n5. Sort by CGPA Asc\n6. Sort by CGPA Desc\n7. Exit\n";
        cout << "Enter choice: ";
        cin >> choice;

        switch (choice) {
            case 1: addStudent(students, count); break;
            case 2: displayStudents(students, count); break;
            case 3: {
                int key;
                cout << "Enter ID to search: ";
                cin >> key;
```

```
            searchByID(students, count, key);
            break;
        }
        case 4: sortByName(students, count); break;
        case 5: sortByCGPA(students, count, true); break;
        case 6: sortByCGPA(students, count, false); break;
        case 7: cout << "Exiting...\n"; break;
        default: cout << "Invalid choice!\n";
      }
   } while (choice != 7);

   free(students);
   return 0;
}
```

Design and implement a maze navigation system that enables an agent to find an **optimal path** from a **starting point** to a **goal** using **AI search algorithms**. The system should be capable of solving both **static** and **dynamic** mazes and should **visualize** the pathfinding process.

(Use only one: BFS or DFS)

……..

```
#include <iostream>
#include <queue>
#include <vector>
#include <map>
#include <algorithm>
```

```
using namespace std;


vector<pair<int,int>> bfs(vector<vector<int>>& maze, pair<int,int> start,
pair<int,int> goal) {
    int r = maze.size(), c = maze[0].size();
    vector<vector<bool>> vis(r, vector<bool>(c, false));
    queue<pair<int,int>> q;
    map<pair<int,int>, pair<int,int> > parent; // note space before '>'


    vector<pair<int,int>> dirs = {{-1,0},{1,0},{0,-1},{0,1}};


    q.push(start);
    vis[start.first][start.second] = true;


    while(!q.empty()) {
        auto cur = q.front(); q.pop();
        if (cur == goal) {
            vector<pair<int,int>> path;
            for (auto it = goal; it != start; it = parent[it]) path.push_back(it);
            path.push_back(start);
            reverse(path.begin(), path.end());
            return path;
        }
        for (auto dir : dirs) {
            int nx = cur.first + dir.first, ny = cur.second + dir.second;
            if (nx>=0 && nx<r && ny>=0 && ny<c && !vis[nx][ny] &&
maze[nx][ny]==0) {
```

```
            q.push({nx,ny});
            vis[nx][ny] = true;
            parent[{nx,ny}] = cur;
        }
    }
  }
  return {};
}

void printMaze(vector<vector<int>>& maze, vector<pair<int,int>>& path) {
  for (int i=0;i<maze.size();i++) {
    for (int j=0;j<maze[0].size();j++) {
      if (find(path.begin(), path.end(), make_pair(i,j)) != path.end()) cout<<"• ";
      else if (maze[i][j]==1) cout<<"█ ";
      else cout<<"  ";
    }
    cout<<'\n';
  }
}

int main() {
  vector<vector<int>> maze = {
    {0,1,0,0,0},
    {0,1,0,1,0},
    {0,0,0,1,0},
    {0,1,0,0,0},
```

```
        {0,0,0,1,0}
    };
    pair<int,int> start={0,0}, goal={4,4};
    auto path = bfs(maze,start,goal);
    if(!path.empty()) {
        cout<<"Path found:\n";
        for(auto p:path) cout<<"("<<p.first<<","<<p.second<<") ";
        cout<<"\n\nMaze Visualization:\n";
        printMaze(maze,path);
    } else cout<<"No path found.\n";
}
```

//

Heap Sort -- Design and implement the Heap Sort algorithm to efficiently sort an array of integers in ascending order. The implementation should be optimized for time and space complexity and should clearly demonstrate the working principles of heap data structures (min-heap or max-heap as applicable)

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

void heapify(vector<int>& a, int n, int i) {
    int largest = i, l = 2*i+1, r = 2*i+2;
    if (l < n && a[l] > a[largest]) largest = l;
    if (r < n && a[r] > a[largest]) largest = r;
    if (largest != i) {
        swap(a[i], a[largest]);
        heapify(a, n, largest);
    }
```

```
}

void heapSort(vector<int>& a) {
    int n = a.size();
    for (int i = n/2 - 1; i >= 0; i--) heapify(a, n, i); // Build max heap
    for (int i = n-1; i > 0; i--) {                    // Extract elements
        swap(a[0], a[i]);
        heapify(a, i, 0);
    }
}

int main() {
    int n;
    cout << "Enter number of elements: ";
    cin >> n;
    vector<int> a(n);
    cout << "Enter " << n << " integers: ";
    for (int i = 0; i < n; i++) cin >> a[i];

    heapSort(a);

    cout << "Sorted array: ";
    for (int x : a) cout << x << " ";
    return 0;
}
```

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

// Merge two sorted halves
void merge(vector<int>& orders, int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;
    vector<int> L(n1), R(n2);

    for (int i = 0; i < n1; i++) L[i] = orders[left + i];
    for (int j = 0; j < n2; j++) R[j] = orders[mid + 1 + j];

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

    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) orders[k++] = L[i++];
        else orders[k++] = R[j++];
    }
```

```
    while (i < n1) orders[k++] = L[i++];
    while (j < n2) orders[k++] = R[j++];
}

// Recursive function implementing Divide and Conquer
void mergeSort(vector<int>& orders, int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        mergeSort(orders, left, mid);
        mergeSort(orders, mid + 1, right);
        merge(orders, left, mid, right);
    }
}

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

    vector<int> deliveryTime(n);
    cout << "Enter delivery times (in minutes): ";
    for (int i = 0; i < n; i++)
        cin >> deliveryTime[i];

    mergeSort(deliveryTime, 0, n - 1);
```

```
    cout << "\nOrders sorted by delivery time (quickest first): ";
    for (int t : deliveryTime)
        cout << t << " ";

    cout << endl;
    return 0;
}
```

Greedy Algorithm (Fractional Knapsack) -

Maximize Profit by Shipping Partial Orders (Fractional Knapsack)

Problem Statement: You run a shipping company and need to load a truck with parcels of different weights and profits. The truck has a limited weight capacity. Write a program to choose parcels (even partially) to maximize profit using the Fractional Knapsack strategy.

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

struct Parcel { double w, p; };

bool cmp(Parcel a, Parcel b) { return (a.p / a.w) > (b.p / b.w); }

double knapsack(vector<Parcel>& v, double cap) {
    sort(v.begin(), v.end(), cmp);
    double profit = 0;
    for (auto &x : v) {
        if (cap >= x.w) { cap -= x.w; profit += x.p; }
```

```
        else { profit += x.p * (cap / x.w); break; }
    }
    return profit;
}

int main() {
    int n; double cap;
    cout << "Enter number of parcels: "; cin >> n;
    vector<Parcel> v(n);
    cout << "Enter weight and profit:\n";
    for (int i = 0; i < n; i++) cin >> v[i].w >> v[i].p;
    cout << "Enter truck capacity: "; cin >> cap;
    cout << "Maximum Profit = " << knapsack(v, cap);
}
```

String Processing: Naïve String Matching -

Given:

l A text string text of length n.

l A pattern string pattern of length m.

Objective:

Find all starting indices i in the text such that the substring text[i:i+m] is exactly equal to the pattern pattern, using the Naive String Matching Algorithm approach.

Constraints:

$l\ 0 \le m \le n$

l Characters in text and pattern can be any valid characters (e.g., a–z, A–Z, digits, etc.)

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

```
void naiveStringMatch(string text, string pattern) {
    int n = text.length(), m = pattern.length();
    bool found = false;

    for (int i = 0; i <= n - m; i++) {
        int j;
        for (j = 0; j < m; j++)
            if (text[i + j] != pattern[j]) break;

        if (j == m) {
            cout << "Pattern found at index " << i << endl;
            found = true;
        }
    }
    if (!found) cout << "Pattern not found." << endl;
}

int main() {
    string text, pattern;
    cout << "Enter text: ";
    getline(cin, text);
    cout << "Enter pattern: ";
    getline(cin, pattern);

    naiveStringMatch(text, pattern);
    return 0;
}
```

Implement stack as an abstract data type using singly linked list and use this ADT in cpp for conversion of infix expression to postfix,

```
#include <iostream>
#include <cctype>
```

```
using namespace std;

// Node structure for stack (linked list)
struct Node {
    char data;
    Node* next;
};

// Stack ADT functions
class Stack {
    Node* top;
public:
    Stack() { top = nullptr; }

    void push(char x) {
        Node* t = new Node;
        t->data = x;
        t->next = top;
        top = t;
    }

    char pop() {
        if (!top) return '\0';
        char x = top->data;
        Node* t = top;
        top = top->next;
```

```
        delete t;
        return x;
    }

    char peek() { return top ? top->data : '\0'; }

    bool isEmpty() { return top == nullptr; }
};

// Utility: precedence of operators
int prec(char c) {
    if (c == '^') return 3;
    if (c == '*' || c == '/') return 2;
    if (c == '+' || c == '-') return 1;
    return -1;
}

// Convert infix to postfix
string infixToPostfix(string infix) {
    Stack s;
    string post;
    for (char c : infix) {
        if (isalnum(c)) post += c;              // operand → output
        else if (c == '(') s.push(c);           // push '('
        else if (c == ')') {                    // pop until '('
            while (!s.isEmpty() && s.peek() != '(')
```

```
            post += s.pop();
        s.pop();
      }
      else {                              // operator
        while (!s.isEmpty() && prec(c) <= prec(s.peek()))
            post += s.pop();
        s.push(c);
      }
   }
   while (!s.isEmpty()) post += s.pop();          // pop remaining
   return post;
}

// Main
int main() {
   string infix;
   cout << "Enter infix expression: ";
   cin >> infix;
   cout << "Postfix expression: " << infixToPostfix(infix);
   return 0;
}
```

Implement Circular Queue using Array. Perform following operations on it.

a) Insertion (Enqueue)

b) Deletion (Dequeue)

c) Display

(Note: Handle queue full condition by considering a fixed size of a queue.)

```
#include <iostream>
using namespace std;
#define SIZE 5  // fixed queue size

class CircularQueue {
   int items[SIZE], front, rear;
public:
   CircularQueue() {
      front = rear = -1;
   }

   bool isFull() {
      return (front == 0 && rear == SIZE - 1) || (front == rear + 1);
   }

   bool isEmpty() {
      return front == -1;
   }

   void enqueue(int value) {
      if (isFull())
         cout << "Queue is Full!\n";
      else {
```

```
        if (front == -1) front = 0;
        rear = (rear + 1) % SIZE;
        items[rear] = value;
        cout << "Inserted: " << value << endl;
    }
}

void dequeue() {
    if (isEmpty())
        cout << "Queue is Empty!\n";
    else {
        cout << "Deleted: " << items[front] << endl;
        if (front == rear)  // only one element
            front = rear = -1;
        else
            front = (front + 1) % SIZE;
    }
}

void display() {
    if (isEmpty())
        cout << "Queue is Empty!\n";
    else {
        cout << "Queue elements: ";
        int i = front;
        while (true) {
```

```
            cout << items[i] << " ";
            if (i == rear) break;
            i = (i + 1) % SIZE;
        }
        cout << endl;
    }
  }
};

int main() {
  CircularQueue q;
  int choice, value;
  cout << "Circular Queue Implementation\n";

  do {
    cout << "\n1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\nEnter choice: ";
    cin >> choice;
    switch (choice) {
      case 1: cout << "Enter value: "; cin >> value; q.enqueue(value); break;
      case 2: q.dequeue(); break;
      case 3: q.display(); break;
      case 4: cout << "Exiting...\n"; break;
      default: cout << "Invalid choice!\n";
    }
  } while (choice != 4);
```

```
    return 0;
}
```

Implement stack as an abstract data type using singly linked list and use this ADT in cpp for conversion of infix expression to postfix

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

// Node for linked list
struct Node {
    char data;
    Node* next;
};

// Stack ADT using singly linked list
class Stack {
    Node* top;
public:
    Stack() { top = NULL; }

    void push(char value) {
        Node* newNode = new Node;
        newNode->data = value;
        newNode->next = top;
        top = newNode;
```

```
    }

    char pop() {
        if (isEmpty()) return '\0';
        char value = top->data;
        Node* temp = top;
        top = top->next;
        delete temp;
        return value;
    }

    char peek() {
        return (isEmpty()) ? '\0' : top->data;
    }

    bool isEmpty() {
        return top == NULL;
    }
};

// Function to check operator precedence
int precedence(char op) {
    if (op == '^') return 3;
    if (op == '*' || op == '/') return 2;
    if (op == '+' || op == '-') return 1;
    return 0;
```

```
}

// Function to convert infix to postfix
string infixToPostfix(string infix) {
    Stack s;
    string postfix = "";
    for (char ch : infix) {
        if (isalnum(ch))
            postfix += ch;  // operand directly added
        else if (ch == '(')
            s.push(ch);
        else if (ch == ')') {
            while (!s.isEmpty() && s.peek() != '(')
                postfix += s.pop();
            s.pop(); // remove '('
        } else {
            while (!s.isEmpty() && precedence(s.peek()) >= precedence(ch))
                postfix += s.pop();
            s.push(ch);
        }
    }
    while (!s.isEmpty())
        postfix += s.pop();
    return postfix;
}
```

```
int main() {
    string infix;
    cout << "Enter infix expression: ";
    cin >> infix;

    string postfix = infixToPostfix(infix);
    cout << "Postfix expression: " << postfix << endl;
    return 0;
}
```