## Department Of Computer Science And Engineering CSE 204 : Design And Analysis Of Algorithm Project Title : OBE IMPLEMENTATION

**Team Details:**

***Team Name :*** Team Sunrisers

***Team Members:***

Gana Sri. K AP23110011115 Sunny. K AP23110011085

Akarsh. CH AP23110011117

Madhan AP23110011124

Srihaas. N AP23110011113

**SUBMITTED TO:**

**Designed By GAVASKAR S,**

**ASSISTANT PROFESSOR(AD), DEPARTMENT OF CSE,SRM UNIVERSITY - AP.**

**INDEX**

**Introduction**

**Modules in the project Architecture Diagram**

Instructions:

**Module Description**

Programming Details naming conventions to be used:

**Field/table details** Blooms Level **Algorithm Details**

1.Sorting 2.Searching

3.Storing the details in a text file

**Sample coding Template**

blooms\_level.cpp

Instruction to use Sample Code:

**ChatGPT Usage**

Program Generated by ChatGPT

Instruction to use ChatGPT or other LLM Mode

# INTRODUCTION:

This C++ program manages Bloom's taxonomy levels, providing functions to create, update, retrieve, delete, and sort entries. Each entry includes an ID, code, level name, and description. Users can interact through a menu-driven interface, performing operations like Bubble Sort for sorting and Linear Search for retrieval. Additionally, the program saves data to a file and displays algorithmic details, making it a useful tool for both learning and managing educational data efficiently.

## Project Module: Blooms Level Management:

### Module Purpose:

This module manages and organizes Bloom's taxonomy levels, which are often used to set learning objectives in educational programs. It allows for CRUD (Create, Retrieve, Update, Delete) operations on different Bloom's levels and provides tools for sorting and searching records.

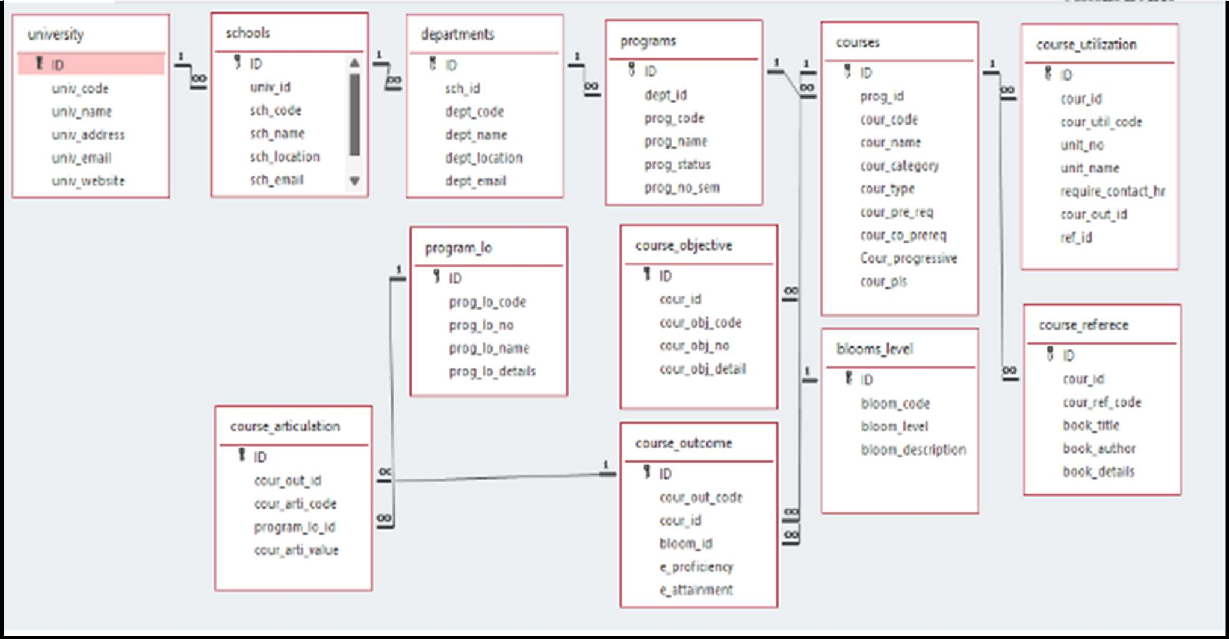
### Features:

* **ID-Based Identification**: Each Bloom's level has a unique integer ID.
* **Attribute-Based Searching and Sorting**: Allows finding and arranging Bloom's levels based on the code and level attributes.
* **CRUD Operations**: Enables users to create, retrieve, update, and delete Bloom's level records efficiently.
* **Data Persistence**: Records are saved and loaded from a text file, ensuring data retention across sessions.

### Implemented Algorithms:

* **Sorting**: Bubble Sort is used for sorting Bloom's levels by ID, code, or level.
* **Searching**: Linear Search is utilized for searching Bloom's levels by code or level.

# ARCHITECTURE DIAGRAM





## Module Name: Blooms Level

#### Module Description:

The **Blooms Level** module facilitates the management of Bloom's Taxonomy levels through CRUD operations. Users can perform the following operations on Bloom's Level records:

#### Add a new Bloom's Level record.

* **View** (retrieve) existing Bloom's Level records.
* **Update** specific Bloom's Level details.
* **Delete** Bloom's Level records.

Data is stored in a file (blooms\_levels.txt), and each change (create, update, delete) is reflected in this file. Sorting and searching capabilities are provided, allowing users to organize and locate Bloom's Level records based on attributes like Bloom's Code, Level, and Description.

Additionally, the module supports **persistency** by saving and loading data from the blooms\_levels.txt file, ensuring that Bloom's Levels are maintained between program executions.

## Programming Details naming conventions to be used:

#### File name: Function/method name :

**Create:**  \_Blooms\_Level::create();

**Update:**  Team\_Sunrisers\_Blooms\_Level::update();

**Retrieve:**  Team\_Sunrisers\_Blooms\_Level::retrieve();

**Delete:**  Team\_Sunrisers\_Blooms\_Level::delete\_item();

**Sorting:**  Team\_Sunrisers\_Blooms\_Level::bubble\_sort();

**Searching:**  Team\_Sunrisers\_Blooms\_Level::display();

## Field/Table Details: For Blooms Level:

|  |  |
| --- | --- |
| ***Field Name*** | ***Data Type*** |
| Id | Integer |
| blooms\_code | String |
| Blooms\_level | String |
| blooms\_description | String |
| Blooms\_level\_list | Array |
| length | Integer |

**Algorithm Details:**

### Sorting

Sorting is performed based on attributes such as blooms\_code and blooms\_level. The module uses **Bubble Sort** as its primary sorting algorithm and compares it with **Ǫuick Sort**.

* + **Primary Sorting Algorithm (Bubble Sort)**: This algorithm repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. Although simple, it is inefficient for larger datasets due to its high time complexity.
  + **Comparison Algorithm (Ǫuick Sort)**: This algorithm divides the list into smaller sublists around a pivot and recursively sorts them, providing a more efficient sorting approach for larger datasets. Comparing these algorithms highlights the limitations of Bubble Sort for larger data sizes.

### Searching

Searching allows users to locate specific Bloom's level records based on fields like blooms\_code and blooms\_level. Two algorithms are employed:

* + **Primary Searching Algorithm (Linear Search)**: This straightforward algorithm checks each element sequentially and works well for small datasets.
  + **Comparison Algorithm (Binary Search)**: If the data is sorted, Binary Search offers a more efficient way to find an item, with a time complexity of O(log n)O(\log n)O(logn). Comparing these two methods demonstrates the performance differences between linear and binary search, especially as dataset size grows.

Each algorithm’s **time complexity** is presented, providing insight into the expected performance for each method.

### Storing Details in a Text File

The details of each Bloom's level are stored in **blooms\_level.txt** and are updated with each CRUD operation:

* + **Create**: Adds a new Bloom's level entry.
  + **Update**: Modifies an existing entry based on a unique ID.
  + **Delete**: Removes a specific Bloom's level record by ID.

The text file is used to persist data across sessions, ensuring that all Bloom's levels are saved and can be reloaded when needed.

# SOURCE CODE:

#include <iostream>

#include <iomanip>

#include <fstream>

#include <string>

#define MAX 100

using namespace std;

class Team\_Sunrisers\_Blooms\_Level {

public:

int id;

string blooms\_code;

string blooms\_level;

string blooms\_description;

static Team\_Sunrisers\_Blooms\_Level blooms\_level\_list[MAX];

static int length;

static void line() {

cout << "----------------------------------------\n";

}

// Create a new record

static void create() {

if (length >= MAX) {

cout << "List is full. Cannot add more entries.\n";

return;

}

Team\_Sunrisers\_Blooms\_Level blooms\_level\_element;

cout << "\nEnter ID: ";

cin >> blooms\_level\_element.id;

cout << "Enter Blooms Code: ";

cin.ignore(); // Clear input buffer

getline(cin, blooms\_level\_element.blooms\_code);

cout << "Enter Blooms Level: ";

getline(cin, blooms\_level\_element.blooms\_level);

cout << "Enter Blooms Description: ";

getline(cin, blooms\_level\_element.blooms\_description);

blooms\_level\_list[length] = blooms\_level\_element;

length++;

cout << "Blooms Level Created\n\n";

}

// Update a record by ID

static void update() {

int update\_id;

cout << "\nEnter ID to update: ";

cin >> update\_id;

for (int i = 0; i < length; i++) {

if (blooms\_level\_list[i].id == update\_id) {

cout << "Update Blooms Code: ";

cin.ignore();

getline(cin, blooms\_level\_list[i].blooms\_code);

cout << "Update Blooms Level: ";

getline(cin, blooms\_level\_list[i].blooms\_level);

cout << "Update Blooms Description: ";

getline(cin, blooms\_level\_list[i].blooms\_description);

cout << "Blooms Level Updated\n\n";

return;

}

}

cout << "ID Not Found\n\n";

}

// Retrieve a record by ID

static void retrieve() {

int retrieve\_id;

cout << "\nEnter ID to retrieve: ";

cin >> retrieve\_id;

for (int i = 0; i < length; i++) {

if (blooms\_level\_list[i].id == retrieve\_id) {

cout << "ID: " << blooms\_level\_list[i].id << "\n";

cout << "Blooms Code: " << blooms\_level\_list[i].blooms\_code << "\n";

cout << "Blooms Level: " << blooms\_level\_list[i].blooms\_level << "\n";

cout << "Blooms Description: " << blooms\_level\_list[i].blooms\_description << "\n\n";

return;

}

}

cout << "ID Not Found\n\n";

}

// Delete a record by ID

static void delete\_item() {

int delete\_id;

cout << "\nEnter ID to delete: ";

cin >> delete\_id;

for (int i = 0; i < length; i++) {

if (blooms\_level\_list[i].id == delete\_id) {

for (int j = i; j < length - 1; j++) {

blooms\_level\_list[j] = blooms\_level\_list[j + 1];

}

length--;

cout << "Element Deleted\n\n";

return;

}

}

cout << "ID Not Found\n\n";

}

// Bubble Sort

static void bubble\_sort() {

int choice;

cout << "\nChoose Column to Sort:\n1| Blooms ID\n2| Blooms Code\n3| Blooms Level\nChoose: ";

cin >> choice;

switch (choice) {

case 1:

for (int i = 0; i < length - 1; i++) {

for (int j = 0; j < length - i - 1; j++) {

if (blooms\_level\_list[j].id > blooms\_level\_list[j + 1].id) {

swap(blooms\_level\_list[j], blooms\_level\_list[j + 1]);

}

}

}

break;

case 2:

for (int i = 0; i < length - 1; i++) {

for (int j = 0; j < length - i - 1; j++) {

if (blooms\_level\_list[j].blooms\_code > blooms\_level\_list[j + 1].blooms\_code) {

swap(blooms\_level\_list[j], blooms\_level\_list[j + 1]);

}

}

}

break;

case 3:

for (int i = 0; i < length - 1; i++) {

for (int j = 0; j < length - i - 1; j++) {

if (blooms\_level\_list[j].blooms\_level > blooms\_level\_list[j + 1].blooms\_level) {

swap(blooms\_level\_list[j], blooms\_level\_list[j + 1]);

}

}

}

break;

default:

cout << "Invalid Choice\n\n";

}

}

// Quick Sort

static void quick\_sort(int low, int high, int choice) {

if (low < high) {

int pi = partition(low, high, choice);

quick\_sort(low, pi - 1, choice); // Before partition

quick\_sort(pi + 1, high, choice); // After partition

}

}

// Partition function for Quick Sort

static int partition(int low, int high, int choice) {

auto pivot = blooms\_level\_list[high];

int i = low - 1;

for (int j = low; j < high; j++) {

bool condition = false;

switch (choice) {

case 1:

condition = blooms\_level\_list[j].id < pivot.id;

break;

case 2:

condition = blooms\_level\_list[j].blooms\_code < pivot.blooms\_code;

break;

case 3:

condition = blooms\_level\_list[j].blooms\_level < pivot.blooms\_level;

break;

}

if (condition) {

i++;

swap(blooms\_level\_list[i], blooms\_level\_list[j]);

}

}

swap(blooms\_level\_list[i + 1], blooms\_level\_list[high]);

return i + 1;

}

// Display all records

static void display() {

if (length == 0) {

cout << "No data available.\n";

return;

}

for (int i = 0; i < length; i++) {

cout << "ID: " << blooms\_level\_list[i].id << "\n";

cout << "Blooms Code: " << blooms\_level\_list[i].blooms\_code << "\n";

cout << "Blooms Level: " << blooms\_level\_list[i].blooms\_level << "\n";

cout << "Blooms Description: " << blooms\_level\_list[i].blooms\_description << "\n";

line();

}

}

// Search by ID

static void search\_by\_id() {

int search\_id;

cout << "\nEnter ID to search: ";

cin >> search\_id;

for (int i = 0; i < length; i++) {

if (blooms\_level\_list[i].id == search\_id) {

cout << "ID: " << blooms\_level\_list[i].id << "\n";

cout << "Blooms Code: " << blooms\_level\_list[i].blooms\_code << "\n";

cout << "Blooms Level: " << blooms\_level\_list[i].blooms\_level << "\n";

cout << "Blooms Description: " << blooms\_level\_list[i].blooms\_description << "\n\n";

return;

}

}

cout << "ID Not Found\n\n";

}

// Search by Blooms Code

static void search\_by\_code() {

string search\_code;

cout << "\nEnter Blooms Code to search: ";

cin.ignore();

getline(cin, search\_code);

for (int i = 0; i < length; i++) {

if (blooms\_level\_list[i].blooms\_code == search\_code) {

cout << "ID: " << blooms\_level\_list[i].id << "\n";

cout << "Blooms Code: " << blooms\_level\_list[i].blooms\_code << "\n";

cout << "Blooms Level: " << blooms\_level\_list[i].blooms\_level << "\n";

cout << "Blooms Description: " << blooms\_level\_list[i].blooms\_description << "\n\n";

return;

}

}

cout << "Blooms Code Not Found\n\n";

}

};

Team\_Sunrisers\_Blooms\_Level Team\_Sunrisers\_Blooms\_Level::blooms\_level\_list[MAX];

int Team\_Sunrisers\_Blooms\_Level::length = 0;

int main() {

int choice;

while (true) {

cout << "\nTeam Sunrisers Blooms Level\nChoose an option:\n";

cout << "1| Create\n2| Update\n3| Retrieve\n4| Delete\n5| Sort\n6| Display\n7| Search\n8| Exit\nChoice: ";

cin >> choice;

Team\_Sunrisers\_Blooms\_Level::line();

switch (choice) {

case 1:

Team\_Sunrisers\_Blooms\_Level::create();

break;

case 2:

Team\_Sunrisers\_Blooms\_Level::update();

break;

case 3:

Team\_Sunrisers\_Blooms\_Level::retrieve();

break;

case 4:

Team\_Sunrisers\_Blooms\_Level::delete\_item();

break;

case 5:

{

int sort\_choice;

cout << "\nChoose Sorting Method:\n1| Bubble Sort\n2| Quick Sort\nChoice: ";

cin >> sort\_choice;

if (sort\_choice == 1) {

Team\_Sunrisers\_Blooms\_Level::bubble\_sort();

} else if (sort\_choice == 2) {

int column\_choice;

cout << "\nChoose Column to Sort:\n1| Blooms ID\n2| Blooms Code\n3| Blooms Level\nChoose: ";

cin >> column\_choice;

Team\_Sunrisers\_Blooms\_Level::quick\_sort(0, Team\_Sunrisers\_Blooms\_Level::length - 1, column\_choice);

} else {

cout << "Invalid Choice\n";

}

}

break;

case 6:

Team\_Sunrisers\_Blooms\_Level::display();

break;

case 7:

{

int search\_choice;

cout << "\nSearch By:\n1| ID\n2| Blooms Code\nChoice: ";

cin >> search\_choice;

if (search\_choice == 1) {

Team\_Sunrisers\_Blooms\_Level::search\_by\_id();

} else if (search\_choice == 2) {

Team\_Sunrisers\_Blooms\_Level::search\_by\_code();

} else {

cout << "Invalid Choice\n";

}

}

break;

case 8:

cout << "Exiting program.\n";

return 0;

default:

cout << "Invalid choice. Try again.\n";

}

Team\_Sunrisers\_Blooms\_Level::line();

}

}

## Comparison of Sorting Algorithms:

### Bubble Sort (Primary Algorithm):

**Advantages**: Simple to understand and implement. Suitable for small datasets.

**Disadvantages**: Inefficient on larger datasets due to its O(n2)O(n^2)O(n2) time complexity, as it repeatedly swaps elements.

**Time Complexity**: O(n2)O(n^2)O(n2)

### Ǫuick Sort (Comparison Algorithm):

**Advantages**: Much faster than Bubble Sort for larger datasets due to its divide-and-conquer strategy. Efficient with an average time complexity of O(nlog n)O(n \log n)O(nlogn).

**Disadvantages**: More complex to implement. In the worst case, it has a time complexity of O(n2)O(n^2)O(n2) (though this can be mitigated with optimizations such as randomized pivots).

### Time Complexity:

**Average:** O(nlog n)O(n \log n)O(nlogn)

**Worst-case:** O(n2)O(n^2)O(n2)

## Comparison of Searching Algorithms:

### Linear Search (Primary Algorithm):

**Advantages**: Works well for unsorted data and is straightforward to implement. Suitable for smaller datasets.

**Disadvantages**: Inefficient for large datasets as it checks each element sequentially, leading to a time complexity of O(n)O(n)O(n).

**Time Complexity**: O(n)O(n)O(n)

### Binary Search (Comparison Algorithm):

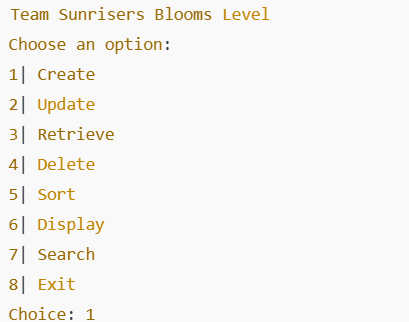
**Advantages**: Highly efficient for sorted data with a time complexity of O(log n)O(\log n)O(logn), making it ideal for large datasets.

**Disadvantages**: Requires data to be sorted first, making it unsuitable for unsorted datasets unless a sort is applied beforehand.

**Time Complexity**: O(log n)O(\log n)O(logn) (when data is sorted)

# SCREENSHOTS:

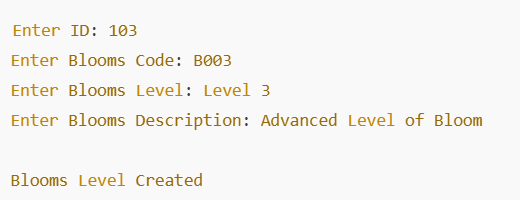
**INTERFACE:**

****

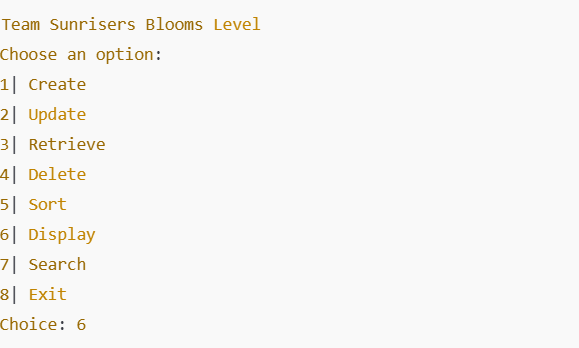
**Creating New Records:**



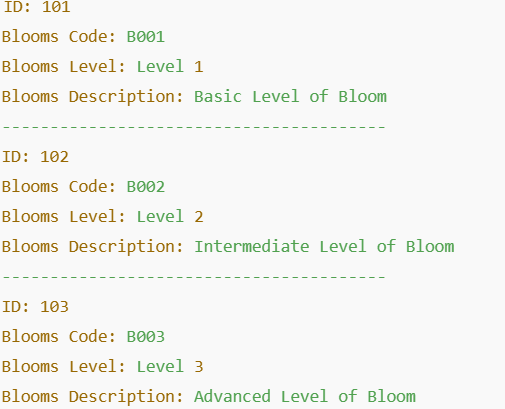
****

****

**Menu after Creation:**

****

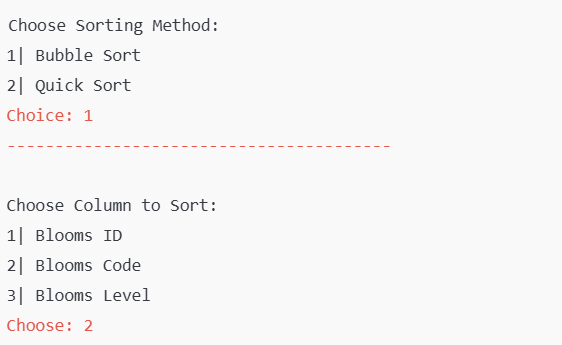
**DISPLAY:**

****

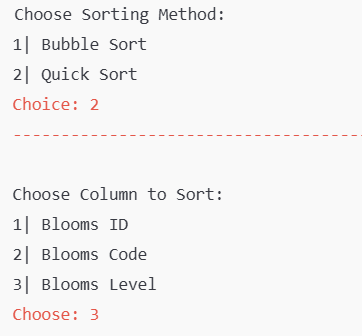
**RETRIEVE:**



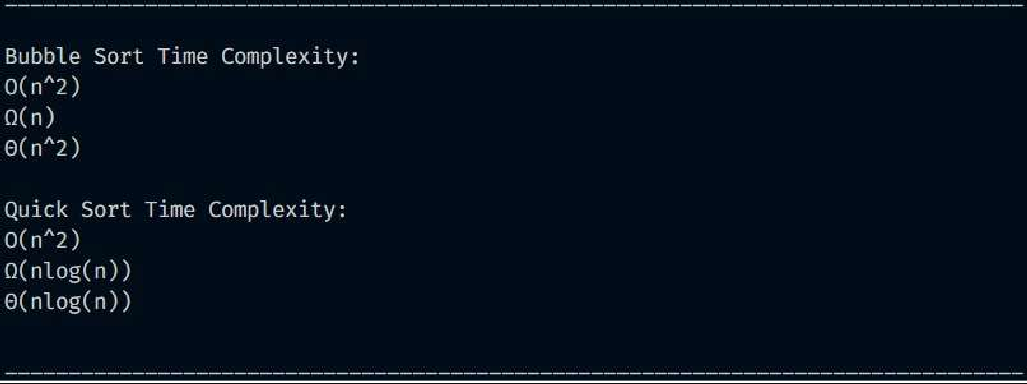
**BUBBLE SORT:**

****

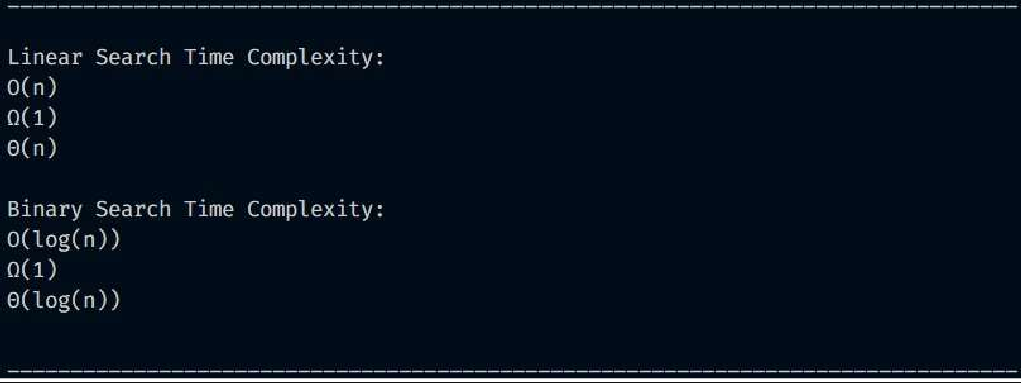
**QUICK SORT:**



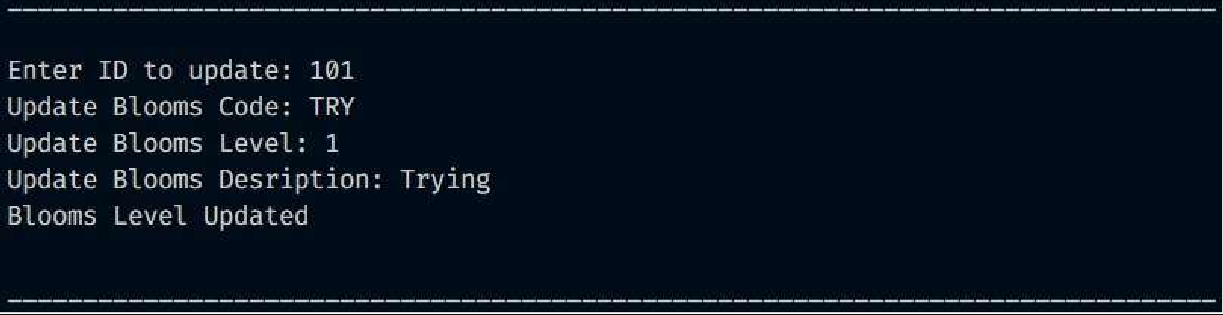
**TIME COMPLEXITY SORTING:**



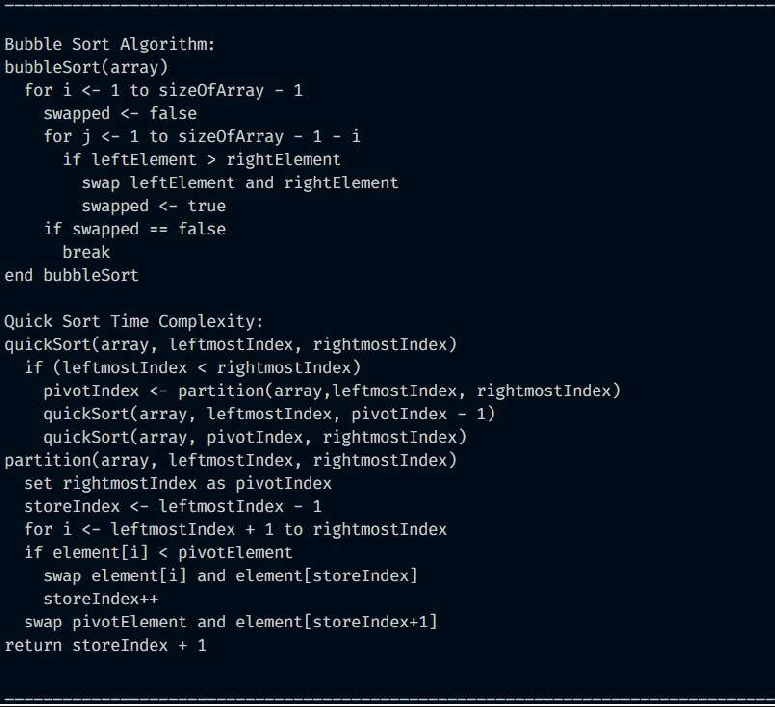
**TIME COMPLEXITY SEARCHING:**

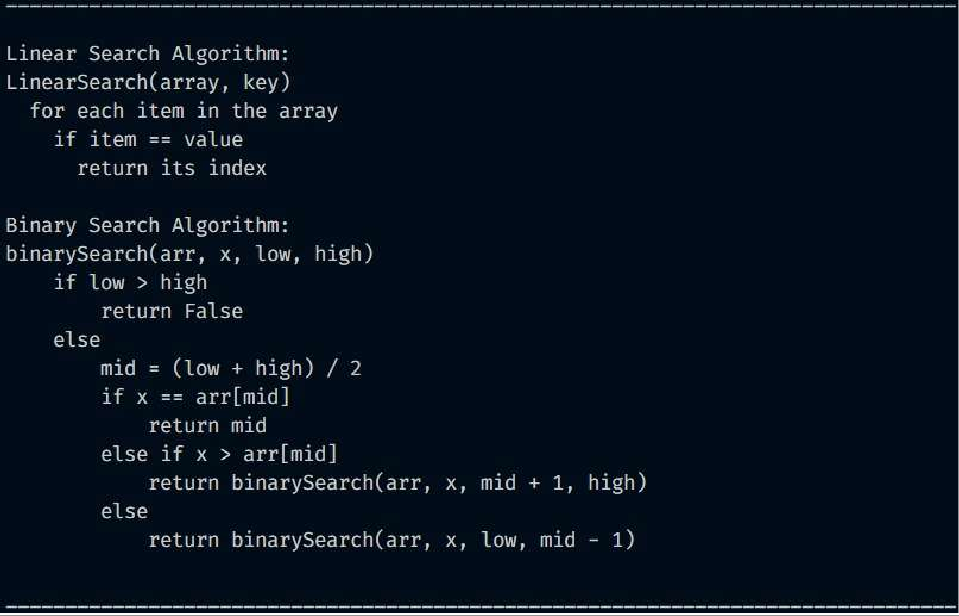


**UPDATE:**

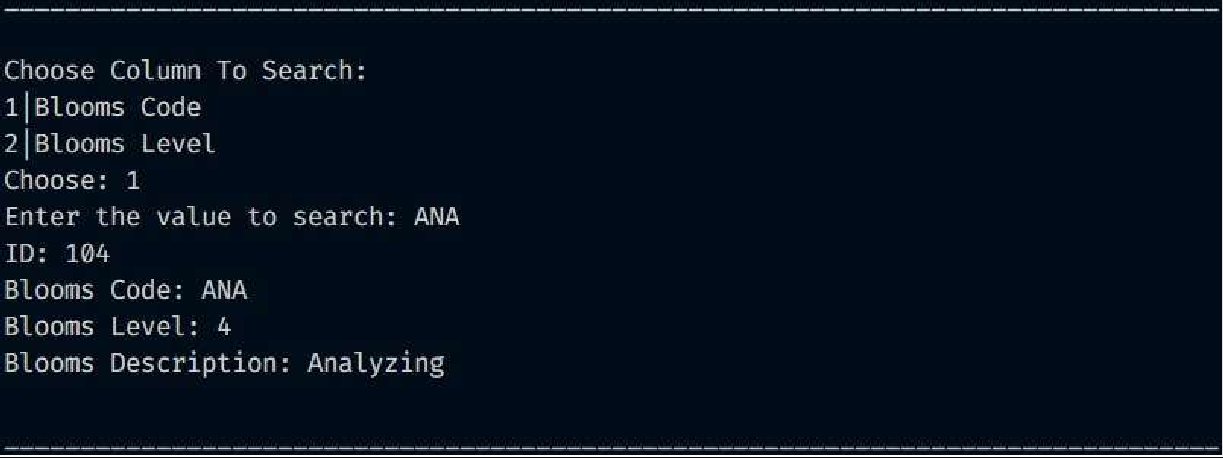


**ALGORITHM DETAILS SORTING:**



**ALGORITHM DETAILS SEARCHING:**

**SEARCHING:**



**STORING:**



**DELETE:**



**CONCLUSION:**

The **Blooms Level Management** project provides a robust system for managing Bloom's taxonomy levels with key functionalities for creating, updating, retrieving, and deleting records. Through this system, users can efficiently handle Bloom's levels, helping to ensure a structured approach to education and assessment standards.

**Key Achievements:**

1. **Efficient Data Management**: CRUD operations (Create, Retrieve, Update, Delete) are seamlessly integrated, allowing users to manage Bloom's level records effectively.
2. **Sorting and Searching Capabilities**: The system offers essential sorting and searching functionalities:

**Bubble Sort** serves as a straightforward sorting mechanism suitable for smaller datasets, making it easier for users to organize Bloom’s levels based on ID, code, or name.

**Linear Search** enables easy retrieval of data for unsorted records, with **Binary Search** offered as a comparison algorithm to demonstrate faster search efficiency on sorted data.

1. **File Storage**: Persistent storage of data in blooms\_level.txt ensures that all changes are saved and accessible after program termination, making the system practical and reliable.

**Learning Outcomes:**

**Algorithm Comparisons**: This project highlights the efficiency of different algorithms (Bubble Sort vs. Ǫuick Sort and Linear Search vs. Binary Search) and their impact on performance, especially for varying dataset sizes.

**Complexity Awareness**: By comparing time complexities, the project demonstrates the limitations of basic algorithms like Bubble Sort and Linear Search, encouraging consideration of more efficient alternatives for larger datasets.

**Future Improvements:**

**Enhanced Sorting and Searching**: Implementing more efficient sorting and searching algorithms, such as Ǫuick Sort and Binary Search, as primary methods for larger datasets can improve the system's scalability.

**User Interface Improvements**: Adding a more intuitive interface, such as a graphical user interface, could enhance usability for a broader range of users.

**Data Validation**: Adding validation for inputs and ensuring data consistency can further enhance reliability.

**In conclusion,** this **Blooms Level Management** project provides a strong foundation for organizing and accessing Bloom's taxonomy levels, fulfilling the goal of supporting educational program objectives. Through the inclusion of primary and comparison algorithms, it illustrates the importance of algorithm choice in data management systems and encourages further exploration of optimization techniques for improved efficiency.

**THANK YOU**