Department Of Computer Science And Engineering

CSE 204 : Design And Analysis Of Algorithm

Project Title : OBE IMPLEMENTATION

Team Details:

Team Name : Team 404,Team Name Not Found

Team Project : Blooms Level

Team Leader : Tanishq Shreeyas – AP23110011165

Team Members:

Tanishq Shreeyas Sunkara	AP23110011165
Garlapati Eswar Rama Krishna	AP23110011173
Rapolu Balaji	AP23110011174
Raveloson Iary Finaritra	AP23110011176
Nambi Sandeep	AP23110011183
Manchi Phanindra Sitha Rama Raju	AP23110011214

SUBMITTED TO:

Designed By

GAVASKAR S,

ASSISTANT PROFESSOR(AD),

DEPARTMENT OF CSE,SRM UNIVERSITY - AP.

INDEX

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.

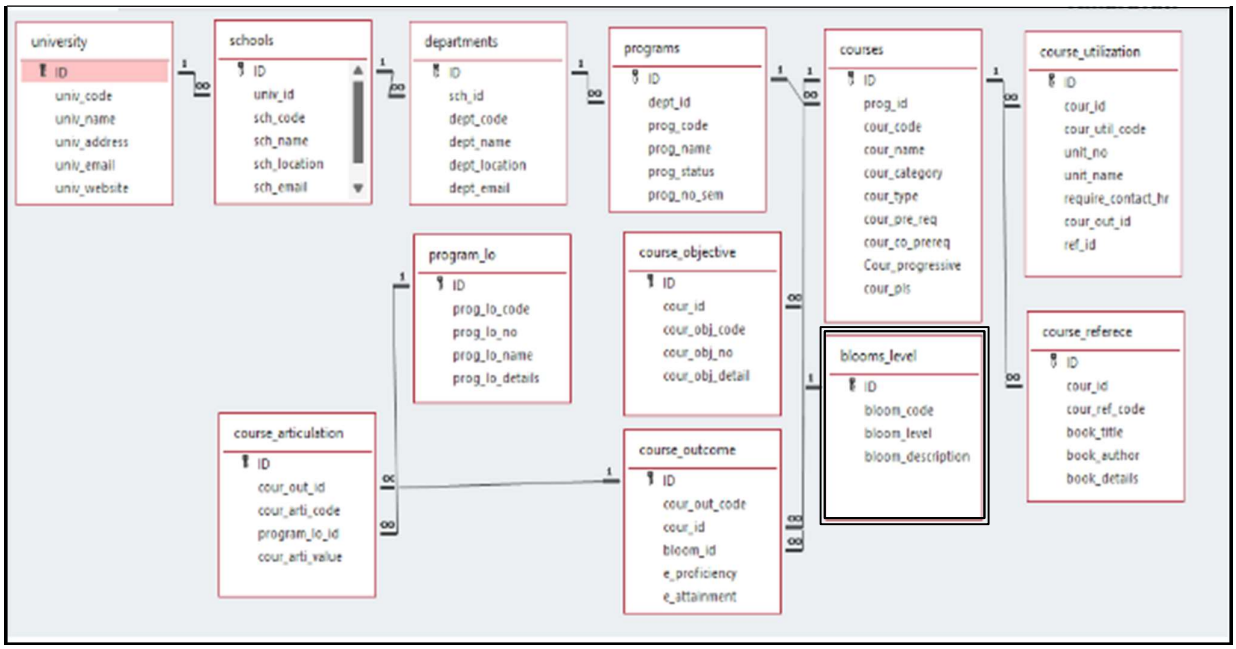
Features:

- **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:	Team_404_Team_Name_Not_Found_Blooms_Level_create
Update:	Team_404_Team_Name_Not_Found_Blooms_Level_update
Retrieve:	Team_404_Team_Name_Not_Found_Blooms_Level_retrive
Delete:	Team_404_Team_Name_Not_Found_Blooms_Level_delete
Sorting:	Team_404_Team_Name_Not_Found_Blooms_Level_bubble sort
Searching:	Team_404_Team_Name_Not_Found_Blooms_Level_linear search
Storing:	Team_404_Team_Name_Not_Found_Blooms_Level_Setting_storing

Comparison Functions:

Sorting Comparison

Team_404_Team_Name_Not_Found_blooms_level_Compare_sorting_bubble_sort

Searching Comparison

Team_404_Team_Name_Not_Found_blooms_level_Compare_Search_linear_search

Time Complexity:

Sorting Time Complexity: Team_404_Team_Name_Not_Found_blooms_level_complexity_sorting

Searching Time Complexity: Team_404_Team_Name_Not_Found_blooms_level_complexity_Search

Algorithm Details(pseudocode or steps)(both searching and Sorting):

For Searching: Team_404_Team_Name_Not_Found_blooms_level_linear_search_details

For Sorting: Team_404_Team_Name_Not_Found_blooms_level_bubble_sort_details

File name(for storing the details): File name to be used is:-blooms_level .txt

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:

1.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 **Quick 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 (Quick 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.

2.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.

3.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>
#define MAX 100
using namespace std;

class Blooms_Level{
  private:
    int id;
    string blooms_code;
    string blooms_level;
    string blooms_description;
    static Blooms_Level blooms_level_list[MAX];
    static int length;
  public:
    static void _404Error_blooms_level_create(){
      Blooms_Level blooms_level_element;
      cout<<"\nEnter ID: ";
      cin>>blooms_level_element.id;
      cout<<"Enter Blooms Code: ";
      cin>>blooms_level_element.blooms_code;
      cout<<"Enter Blooms Level: ";
      cin>>blooms_level_element.blooms_level;
      cout<<"Enter Blooms Desription: ";
      cin>>blooms_level_element.blooms_description;
      blooms_level_list[length]=blooms_level_element;
      length++;
      cout<<"Blooms Level Created\n\n";
```

```
}
static void _404Error_blooms_level_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>>blooms_level_list[i].blooms_code;
         cout<<"Update Blooms Level: ";
         cin>>blooms_level_list[i].blooms_level;
         cout<<"Update Blooms Desription: ";
         cin>>blooms_level_list[i].blooms_description;
         cout<<"Blooms Level Updated\n\n";
         return;
      }
   }
   cout<<"ID Not Found\n\n";
}
static void _404Error_blooms_level_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";
 }
 static void _404Error_blooms_level_delete(){
   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";
}
 static void _404Error_blooms_level_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){
           Blooms_Level temp=blooms_level_list[j];
           blooms_level_list[j]=blooms_level_list[j+1];
```

```
        blooms_level_list[j+1]=temp;
            }
          }
        }
        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){
              Blooms_Level temp=blooms_level_list[j];
              blooms_level_list[j]=blooms_level_list[j+1];
              blooms_level_list[j+1]=temp;
            }
          }
        }
        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){
              Blooms_Level temp=blooms_level_list[j];
              blooms_level_list[j]=blooms_level_list[j+1];
              blooms_level_list[j+1]=temp;
            }
          }
        }
        break;
    default:cout<<"Invalid Choice\n\n";
    }

}
```

```
static void _404Error_blooms_level_linear_search(){
        int choice;
        string search_value;
        bool found=false;
        cout<<"\nChoose Column To Search:\n1|Blooms Code\n2|Blooms Level\nChoose: ";
        cin>>choice;
        cout<<"Enter the value to search: ";
        cin>>search_value;
        switch(choice){
          case 1:
            for(int i=0;i<length;i++){
              if(blooms_level_list[i].blooms_code==search_value){
                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";
                found=true;
              }
            }
            if(!found){
              cout<<"Value Not Found\n\n";
            }
            break;
          case 2:
            for(int i=0;i<length;i++){
              if(blooms_level_list[i].blooms_level==search_value){
                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";
                found=true;
```

```
}
            }
            if(!found){
               cout<<"Value Not Found\n\n";
            }
            break;
         default:cout<<"Invalid Choice\n\n";
      }
   }
   static void _404Error_blooms_level_storing(){
      ofstream outFile("blooms_level.txt");
      if(outFile.is_open()){
         for(int i=0;i<length;i++){
            outFile<<blooms_level_list[i].id<<" "
                  <<blooms_level_list[i].blooms_code<<" "
                  <<blooms_level_list[i].blooms_level<<" "
                  <<blooms_level_list[i].blooms_description<<"\n";
         }
         outFile.close();
         cout<<"Data stored successfully.\n\n";
      }
      else{
         cout<<"Unable to open file for writing.\n\n";
      }
   }
   static void _404Error_blooms_level_compare_sorting_bubble_sort(){
      _404Error_blooms_level_complexity_sorting();
      _404Error_blooms_level_bubble_sort_details();
   }
   static void _404Error_blooms_level_compare_searching_linear_search(){
      _404Error_blooms_level_complexity_searching();
```

```
_404Error_blooms_level_linear_search_details();
    }
    static void _404Error_blooms_level_complexity_sorting(){
      cout<<"\nBubble Sort Time Complexity:\nO(n^2)\nΩ(n)\nΘ(n^2)\n";
      cout<<"\nQuick Sort Time Complexity:\nO(n^2)\nΩ(nlog(n))\nΘ(nlog(n))\n\n";
    }
    static void _404Error_blooms_level_complexity_searching(){
      cout<<"\nLinear Search Time Complexity:\nO(n)\nΩ(1)\nΘ(n)\n";
      cout<<"\nBinary Search Time Complexity:\nO(log(n))\nΩ(1)\nΘ(log(n))\n\n";
    }
    static void _404Error_blooms_level_bubble_sort_details(){
      cout<<"\nBubble Sort Algorithm:\nbubbleSort(array)\n  for i <- 1 to sizeOfArray - 1\n   swapped <- false\n for j <- 1 to sizeOfArray - 1 - i\n    if leftElement > rightElement\n      swap leftElement and rightElement\n swapped <- true\n   if swapped == false\n    break\nend bubbleSort\n";
      cout<<"\nQuick Sort Time Complexity:\nquickSort(array, leftmostIndex, rightmostIndex)\n  if (leftmostIndex < rightmostIndex)\n   pivotIndex <- partition(array,leftmostIndex, rightmostIndex)\n quickSort(array, leftmostIndex, pivotIndex - 1)\n   quickSort(array, pivotIndex, rightmostIndex)\npartition(array, leftmostIndex, rightmostIndex)\n  set rightmostIndex as pivotIndex\n storeIndex <- leftmostIndex - 1\n  for i <- leftmostIndex + 1 to rightmostIndex\n  if element[i] < pivotElement\n swap element[i] and element[storeIndex]\n   storeIndex++\n  swap pivotElement and element[storeIndex+1]\nreturn storeIndex + 1\n\n";
    }
    static void _404Error_blooms_level_linear_search_details(){
      cout<<"\nLinear Search Algorithm:\nLinearSearch(array, key)\n  for each item in the array\n   if item == value\n    return its index\n";
      cout<<"\nBinary Search Algorithm:\nbinarySearch(arr, x, low, high)\n   if low > high\n      return False\n else\n      mid = (low + high) / 2\n      if x == arr[mid]\n         return mid\n      else if x > arr[mid]\n         return binarySearch(arr, x, mid + 1, high)\n      else\n         return binarySearch(arr, x, low, mid - 1)\n\n";
    }
    static void _404Error_blooms_level_load(){
      ifstream inFile("blooms_level.txt");
      if(inFile.is_open()){
        length=0;
        while(!inFile.eof()){
          Blooms_Level blooms_level_element;
```

```
inFile>>blooms_level_element.id;
            inFile.ignore();
            getline(inFile,blooms_level_element.blooms_code);
            getline(inFile,blooms_level_element.blooms_level);
            getline(inFile,blooms_level_element.blooms_description);
            blooms_level_list[length]=blooms_level_element;
            length++;
          }
      inFile.close();
      cout<<"Data loaded successfully.\n";
      }
      else{
        cout<<"Unable to open file for reading.\n";
      }
    }
    static void _404Error_blooms_level_display_table(){
      if(length==0){
        cout<<"Blooms Level Table Empty\n\n";
        return;
      }
      cout<<"\nBlooms Levels Table:\n";
      cout<<setw(10)<<left<<"ID"
        <<setw(20)<<left<<"Blooms Code"
        <<setw(20)<<left<<"Blooms Level"
        <<setw(50)<<left<<"Blooms Description"<<"\n";
      cout<<"--------------------------------------------------------------------------------\n";
      for(int i=0;i<length;i++){
        cout<<setw(10)<<left<<blooms_level_list[i].id
          <<setw(20)<<left<<blooms_level_list[i].blooms_code
          <<setw(20)<<left<<blooms_level_list[i].blooms_level
          <<setw(50)<<left<<blooms_level_list[i].blooms_description << "\n";
```

```
}
        cout<<"--------------------------------------------------------------------------------\n";
    }
    static void _404Error_blooms_level_line(){
        cout<<"\n";
    }
};
Blooms_Level Blooms_Level::blooms_level_list[MAX];
int Blooms_Level::length=0;


int main(){
  int choice;
  Blooms_Level::_404Error_blooms_level_load();
  while(true){
    cout<<"Blooms Level\nChoose an option:\n1| Create\n2| Update\n3| Retrieve\n4| Delete\n5| Sorting\n6| Searching\n7| Storing\n8| Comparision Sorting\n9| Comparision Searching\n10| Time Complexity Sorting\n11| Time Complexity Searching\n12| Algorithm Details Sorting\n13| Algorithm Details Searching\n14| Display\n15|Exit\nChoice: ";
    cin>>choice;
    Blooms_Level::_404Error_blooms_level_line();
    switch(choice){
      case 1:Blooms_Level::_404Error_blooms_level_create();break;
      case 2:Blooms_Level::_404Error_blooms_level_update();break;
      case 3:Blooms_Level::_404Error_blooms_level_retrieve();break;
      case 4:Blooms_Level::_404Error_blooms_level_delete();break;
      case 5:Blooms_Level::_404Error_blooms_level_bubble_sort();break;
      case 6:Blooms_Level::_404Error_blooms_level_linear_search();break;
      case 7:Blooms_Level::_404Error_blooms_level_storing();break;
      case 8:Blooms_Level::_404Error_blooms_level_compare_sorting_bubble_sort();break;
      case 9:Blooms_Level::_404Error_blooms_level_compare_searching_linear_search();break;
      case 10:Blooms_Level::_404Error_blooms_level_complexity_sorting();break;
      case 11:Blooms_Level::_404Error_blooms_level_complexity_searching();break;
```

```
case 12:Blooms_Level::_404Error_blooms_level_bubble_sort_details();break;
        case 13:Blooms_Level::_404Error_blooms_level_linear_search_details();break;
        case 14:Blooms_Level::_404Error_blooms_level_display_table();break;
        case 15:return 0;
        default:cout<<"Invalid choice. Try again.\n";
      }
     Blooms_Level::_404Error_blooms_level_line();
   }
   Blooms_Level::_404Error_blooms_level_storing();
   return 0;
}
```

Comparison of Sorting Algorithms:

1. **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)

2. **Quick 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:

1. **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)

2. **Binary Search (Comparison Algorithm):**

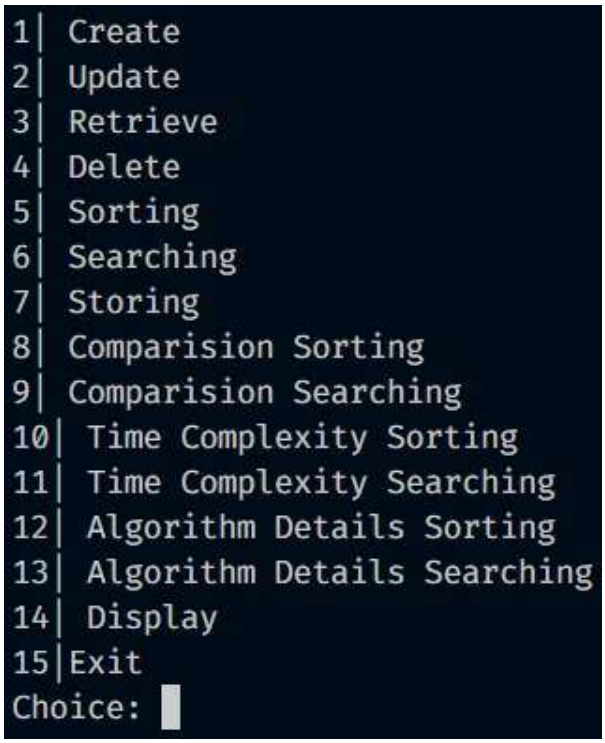
   Advantages: Highly efficient for sorted data with a time complexity of O(log n)O(\log n)O(logn), making it ideal for large datasets.

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

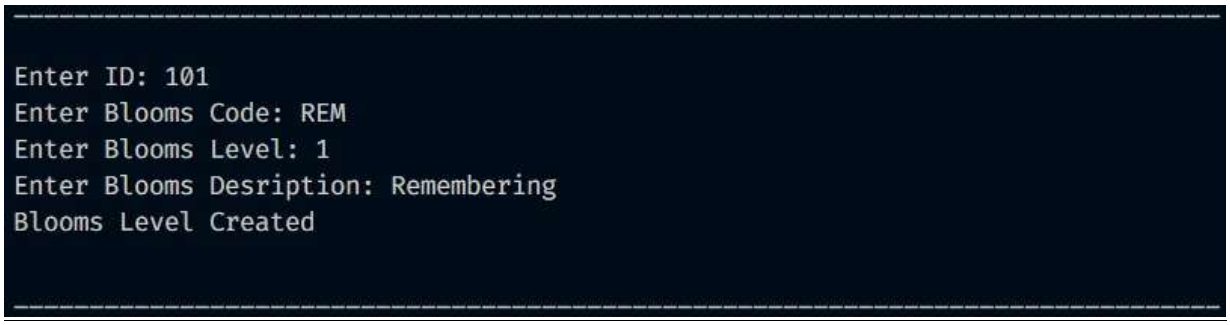
   Time Complexity: O(log n)O(\log n)O(logn) (when data is sorted)

SCREENSHOTS:

INTERFACE:



CREATE:



DISPLAY:

```
Blooms Levels Table:
ID        Blooms Code         Blooms Level        Blooms Description
--------------------------------------------------------------------------------
101       REM                 1                   Remembering
102       UND                 2                   Understanding
103       APP                 3                   Applying
104       ANA                 4                   Analyzing
105       EVAL                5                   Evaluating
106       CRE                 6                   Creating
--------------------------------------------------------------------------------
```

RETRIEVE:

```
--------------------------------------------------------------------------------

Enter ID to retrieve: 103
ID: 103
Blooms Code: APP
Blooms Level: 3
Blooms Description: Applying


--------------------------------------------------------------------------------
```

SORTING:

```
Blooms Levels Table:
ID        Blooms Code         Blooms Level        Blooms Description
--------------------------------------------------------------------------------
104       ANA                 4                   Analyzing
103       APP                 3                   Applying
106       CRE                 6                   Creating
105       EVAL                5                   Evaluating
101       REM                 1                   Remembering
102       UND                 2                   Understanding
--------------------------------------------------------------------------------
```

TIME COMPLEXITY SORTING:

```
Bubble Sort Time Complexity:
O(n^2)
Ω(n)
θ(n^2)

Quick Sort Time Complexity:
O(n^2)
Ω(nlog(n))
θ(nlog(n))
```

TIME COMPLEXITY SEARCHING:

```
Linear Search Time Complexity:
O(n)
Ω(1)
θ(n)

Binary Search Time Complexity:
O(log(n))
Ω(1)
θ(log(n))
```

UPDATE:

```
Enter ID to update: 101
Update Blooms Code: TRY
Update Blooms Level: 1
Update Blooms Desription: Trying
Blooms Level Updated
```

ALGORITHM DETAILS SORTING:

```
Bubble Sort Algorithm:
bubbleSort(array)
  for i <- 1 to sizeOfArray - 1
    swapped <- false
    for j <- 1 to sizeOfArray - 1 - i
      if leftElement > rightElement
        swap leftElement and rightElement
        swapped <- true
    if swapped == false
      break
end bubbleSort

Quick Sort Time Complexity:
quickSort(array, leftmostIndex, rightmostIndex)
  if (leftmostIndex < rightmostIndex)
    pivotIndex <- partition(array,leftmostIndex, rightmostIndex)
    quickSort(array, leftmostIndex, pivotIndex - 1)
    quickSort(array, pivotIndex, rightmostIndex)
partition(array, leftmostIndex, rightmostIndex)
  set rightmostIndex as pivotIndex
  storeIndex <- leftmostIndex - 1
  for i <- leftmostIndex + 1 to rightmostIndex
  if element[i] < pivotElement
    swap element[i] and element[storeIndex]
    storeIndex++
  swap pivotElement and element[storeIndex+1]
return storeIndex + 1
```

ALGORITHM DETAILS SEARCHING:

```
Linear Search Algorithm:
LinearSearch(array, key)
  for each item in the array
    if item == value
      return its index

Binary Search Algorithm:
binarySearch(arr, x, low, high)
    if low > high
        return False
    else
        mid = (low + high) / 2
        if x == arr[mid]
            return mid
        else if x > arr[mid]
            return binarySearch(arr, x, mid + 1, high)
        else
            return binarySearch(arr, x, low, mid - 1)
```

SEARCHING:

```
Choose Column To Search:
1|Blooms Code
2|Blooms Level
Choose: 1
Enter the value to search: ANA
ID: 104
Blooms Code: ANA
Blooms Level: 4
Blooms Description: Analyzing
```

STORING:

```
Data stored successfully.
```

DELETE:

```
Enter ID to delete: 101
Element Deleted
```

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.

3. **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. Quick 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 Quick 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.

SRM UNIVERSITY AP
Andhra Pradesh

THANK YOU