Discipline Core

ALGORITHMS

CS4003

Assignment 2

Sheikh Muhammed Tadeeb (AU19B1014)

❖ Problem Statement:

- Description:

  Searching is an algorithmic technique to search any item stored in the system using data structures like array, queue, and linked list, etc. In this exercise, you will implement the searching algorithm and compare the techniques in terms of the number of iterations required to search for any specific item. Finally, you will be commenting on the complexity trade-off for all the techniques.

- Instructions:

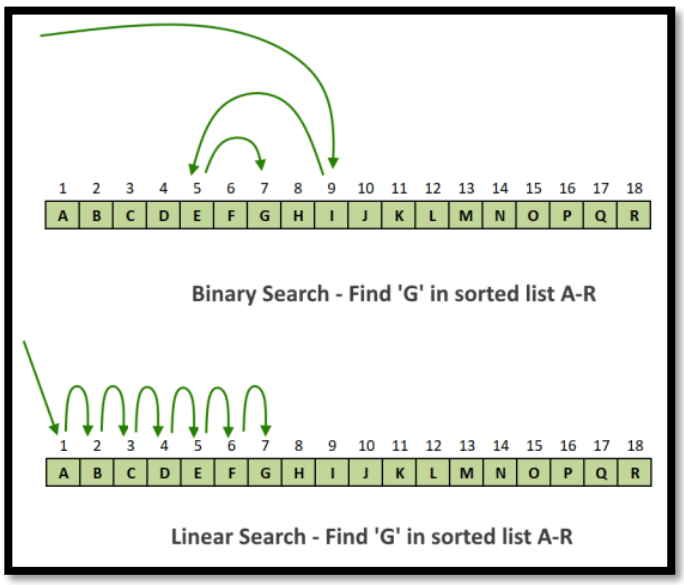
  Suppose the university scheduled a cultural event and students will be participating in various activities. You are selected as a lead drama artist by the university and given the task of presenting a drama during the event. You start your preparation by first searching the students to play a role in your skit, but to select any student his/her height should be exactly more than 5 feet's. This becomes a difficult task for you to select the specific student when they approach you and none of them are very sure of their exact height. To avoid the manual procedure of measuring each student height you as a computer science student decided to build a small application that will take student details like name and height as input and find the students who is exactly more than 5 feet's in height.

❖ Theory:

Searching Algorithms are designed to check for an element or retrieve an element from any data structure where it is stored. Based on the type of search operation, these algorithms are generally classified into two categories:

1. **Sequential Search:** In this, the list or array is traversed sequentially and every element is checked. For example: Linear Search.

2. **Interval Search:** These algorithms are specifically designed for searching in sorted data-structures. These type of searching algorithms are much more efficient than Linear Search as they repeatedly target the centre of the search structure and divide the search space in half. For Example: Binary Search.



❖ C++ Code:

```
1. #include <iostream>
2. #include <fstream>
3. #include<conio.h>
4. using namespace std;
5.
6. // Struct class.
7. class stud{
8.        public:
9.               string name;
10.              float height;
11.              stud* next;
12. };
13.
14. class search_algo
15. {
16.       protected:
17.              // Calling of all member functions
18.              void insert();
19.              void admin();
20.              stud* list_init(stud* arr[]);
21.              void display(stud* head);
22.              void unordered_linear_search(stud* head, float element);
23.              void ordered_linear_search(stud* head,float element);
24.              stud* mergeSort(stud* head);
25.              stud* merge(stud *L, stud *R);
26.              stud* mergeSort_for_ordered_linear_search(stud *head);
27.              stud* merge_for_ordered_linear_search(stud *L, stud *R);
28.              stud* binary_search(stud* head,float element,stud* arr[],int
    n);
29.              stud* middle(stud* start, stud* last);
30.              void Sorting_for_ordered_linear_search(stud *head);
31.
32.              // Calling of all the above functions in this function
33.              void call(){
34.                     stud* head;
35.                     int ch,n;
36.                     cout<<endl<<endl<<endl<<endl<<"\t# Enter the size of
    auxilary array:- ";
37.                     cin>>n;
38.                     float element;
39.                     /* To do indexing in Linked List we will make an
    auxiliary
40.                     pointer array having size same as linked list which will
    store
```

```
41.                     address of each corresponding node of linked list. */
42.                     stud *arr[n];
43.                     // Formatting the insert function.
44.                     cout<<endl<<endl<<"\t _____________________________
   Welcome to the competition,
   ____________________________________________________";
45.                     cout<<endl<<endl<<"\t\t\t\t\t  Register yourself here.
   ";
46.                     // Inserting data into the file.
47.                     insert();
48.                     // Creating a linked list.
49.                     head = list_init(arr);
50.                     // Taking the element to be seached.
51.                     cout<<endl<<endl<<endl<<endl<<"\t
   ____________________________________ Admin control
   ____________________________________________________";
52.                     // Calling the login function.
53.                     admin();
54.                     // Displaying the linked list.
55.                     char c;
56.               cout<<endl<<endl<<"\t# Do you want to see all records(y/n):-
   ";
57.               cin>>c;
58.               if (c == 'y' || c == 'Y'){
59.                     display(head);
60.                     }
61.                     // User Controlled system.
62.                     cout<<endl<<endl<<endl<<endl<<"\t # Enter the least
   height required for qualification:- ";
63.                     cin>>element;
64.                     // Giving Options
65.                     cout<<endl<<endl<<"\t# Enter your choice for searching:-
   ";
66.                     cout<<"\n\t\t\t\t1. Unordered linear search";
67.                     cout<<"\n\t\t\t\t2. Ordered linear search";
68.                     cout<<"\n\t\t\t\t3. Binary search"<<endl;
69.                     cout<<"\t Enter here: ";
70.             cin>>ch;
71.             // User controlled search.
72.                     switch(ch){
73.                           case 1:
74.                                 // A function to do linear search on
   Unsorted list
75.                                 unordered_linear_search(head,element);
76.                                 break;
77.                           case 2:
78.                                 // Sorting the list using merge sort
79.                                 Sorting_for_ordered_linear_search(head);
```

```
80.                                     // Displaying the linked list.
81.                                     char c;
82.                                   cout<<endl<<endl<<endl<<"\t# Do you want to see
   the sorted records(y/n):- ";
83.                                   cin>>c;
84.                                   if (c == 'y' || c == 'Y'){
85.                                     display(head);
86.                                     }
87.                                     // A function to do linear search on Sorted
   list
88.                                     ordered_linear_search(head,element);
89.                                     break;
90.                                 case 3:
91.                                     // Sorting the list using merge sort
92.                                     mergeSort(head);
93.                                     stud* new_data = head;
94.                                     int i = 0;
95.                                     // Inserting the address of corrosponding
   sorted nodes inside the array
96.                                     while(new_data != NULL){
97.                                           arr[i] = new_data;
98.                                           new_data = new_data->next;
99.                                           i++;
100.                                          }
101.                                          // Displaying the linked list.
102.                                    cout<<endl<<endl<<endl<<"\t# Do you want
   to see the sorted records(y/n):- ";
103.                                    cin>>c;
104.                                    if (c == 'y' || c == 'Y'){
105.                                      display(head);
106.                                      }
107.                                      // A function to do Binary search
   Sorted list.
108.                                      stud* val =
   binary_search(head,element,arr,n);
109.                                      // Used to display the search result
   of binary data.
110.                                      display(val);
111.                                      break;
112.                                }
113.                    }
114.      };
115.
116.      // Admin login function
117.      void search_algo :: admin(){
118.            int svr;
119.            string userName,userPassword="";
120.            char pw =' ';
```

```
121.              int loginAttempt = 0;
122.              while (loginAttempt < 2)
123.              {
124.                     cout<<"\n\n\n\n\t Please enter your user name: ";
125.                     cin>>userName;
126.                     cout<<"\n\t Please enter your user password: ";
127.                     while(pw!=13)
128.                        {
129.                                  pw = getch();
130.                                  if(pw!=13)
131.                                  {
132.                                  userPassword += pw;
133.                                  cout<<"*";
134.                                }
135.                              }
136.                     if ((userName == "Tadeeb" || userName == "Atul") &&
   (userPassword == "tadeeb" || userPassword == "atul"))
137.                     {
138.                                  cout<<"\n\n\n\t\t\t\t\t\t Welcome
   "<<userName<<" "<<"!\n";
139.                                  break;
140.                     }
141.                     else
142.                     {
143.                            cout<<"\n\n Invalid login attempt. Please try
   again.\n" << '\n';
144.                            loginAttempt++;
145.                     }
146.              }
147.              // This if just to check user attempts and is outside of if-
   else ladder.
148.              if (loginAttempt == 2)
149.              {
150.                     cout << "\n\t\t\t\tToo many login attempts! Sorry try
   after 10 minutes.";
151.                     exit(0);
152.              }
153.       }
154.
155.
156.       // Function to insert data into two files.
157.       void search_algo :: insert(){
158.              // declaring objects for class ofstream.
159.              ofstream file1,file2;
160.              // Opening the files.
161.              file1.open("name.txt", ios::app);
162.              file2.open("heights.txt", ios::app);
163.              // Defining user-controlled system.
```

```
164.            char ch;
165.            char name[90];
166.            float height;
167.            do{
168.                   cout<<endl<<endl<<"\t# Enter your first name:- ";
169.                   cin>>name;
170.                   cout<<endl;
171.                   file1<<endl<<name;
172.                   cout<<endl<<"\t# Enter your height(inches):- ";
173.                   cin>>height;
174.                   file2<<endl<<height;
175.                   cout<<endl<<endl<<endl<<"\t Any more students(y/n):- ";
176.                   cin>>ch;
177.            }while(ch == 'y' || ch == 'Y');
178.            // Closing the files
179.            file1.close();
180.            file2.close();
181.     }
182.
183.
184.     /* Function for creating a list
185.      and entering the values in it. */
186.     stud* search_algo :: list_init(stud* arr[]){
187.         // Declaring the pointers.
188.            stud* head, *p;
189.            // declaring objects for class ifstream.
190.            ifstream file1,file2;
191.            // Opening the files.
192.            file1.open("name.txt");
193.            file2.open("heights.txt");
194.            // Creating the head node.
195.          head = new stud;
196.            p = head;
197.            // At index 0 we are storing head address.
198.            arr[0] = p;
199.            // creating int i for inserting values inside the linked list.
200.            int i = 1;
201.            // While loop for inserting values in Linked list as well as
    auxiliary array.
202.            while(file1 && file2){
203.                   p->next = new stud;
204.                   arr[i] = p->next;
205.                   file1>>p->name;
206.                   file2>>p->height;
207.                   p = p->next;
208.                   i++;
209.            }
210.            // Closing the files.
```

```
211.            file1.close();
212.            file2.close();
213.            // Making the last nodes next as end point.
214.            p->next = NULL;
215.            return head;
216.      }
217.
218.      /* Function for Displaying the list */
219.      void search_algo :: display(stud* head){
220.            // Storing the address of head node.
221.          stud* p = head;
222.            // Formatted outputs
223.            cout<<endl<<endl<<endl<<"\t # The list of eligible students
    is:- ";
224.            cout<<endl<<endl<<"\t\t\t\t\t\t\t Name    Height";
225.            cout<<endl<<"\t\t\t\t\t\t\t_________________";
226.            while(p != NULL){
227.                  cout<<endl<<"\t\t\t\t\t\t |"<<p->name;
228.                  cout<<"\t "<<p->height<<"|";
229.                  p = p->next;
230.            }
231.      }
232.
233.      /* A function to do linear search
234.      on Unsorted list or unordered list. */
235.      void search_algo :: unordered_linear_search(stud* head,float element){

236.            // Declaring msg variable to check whether the value is present
    or not.
237.            int msg = 0;
238.            // Formatted outputs
239.            cout<<endl<<endl<<endl<<"\t # The list of eligible students
    is:- ";
240.            cout<<endl<<endl<<"\t\t\t\t\t\t\t Name    Height";
241.            cout<<endl<<"\t\t\t\t\t\t\t_________________";
242.            // Searching for the value
243.          for(stud* i = head ; i != NULL; i= i->next){
244.            // If value is found declare msg as 1 i.e. value is present.
245.                  if(i->height > element){
246.                  msg = 1;
247.                  cout<<endl<<"\t\t\t\t\t\t| "<<i->name<<"    "<<i-
    >height<<"|";
248.                  }
249.            }
250.            // If value is not found declare msg remains 0 i.e. value is
    not present.
251.            if(msg == 0){
252.                  cout<<endl<<"Element Not found";
```

```
253.            }
254.      }
255.
256.      /* A function to do linear search
257.      on Sorted list or Ordered list. */
258.      void search_algo :: ordered_linear_search(stud* head,float element){
259.            // Declaring msg variable to check whether the value is present
      or not.
260.            int msg = 0;
261.            // Formatted outputs
262.            cout<<endl<<endl<<endl<<"\t # The list of eligible students
      is:- ";
263.            cout<<endl<<endl<<"\t\t\t\t\t\t\t Name    Height";
264.            cout<<endl<<"\t\t\t\t\t\t_________________";
265.            // Searching for the value
266.            for(stud* i = head ; i != NULL; i = i->next){
267.                  // If value is found declare msg as 1 i.e. value is
      present.
268.                  if(i->height > element){
269.                        msg = 1;
270.                        cout<<endl<<"\t\t\t\t\t\t| "<<i->name<<"   "<<i-
      >height<<"|";
271.                  }
272.                  // Else is we reach a point after which there is no
      point in searching so we stop.
273.                  else if(i->height < element){
274.                        break;
275.                  }
276.            }
277.            // If value is not found declare msg remains 0 i.e. value is
      not present.
278.            if (msg == 0){
279.                  cout<<endl<<endl<<endl<<" The Element "<<element<<" is
      not found";
280.            }
281.      }
282.
283.      stud* search_algo :: mergeSort(stud *head) {
284.            // Base condition for stopping.
285.          if(!head || !head->next) return head;
286.          // getting middle node using these pointers fast and slow.
287.          stud *slow = head, *fast = head->next;
288.          while(fast && fast->next){
289.              slow = slow->next;
290.              fast = fast->next->next;
291.          }
292.          // Creating left and right head pointer for two sub-lists.
293.          stud *left = head, *right = slow->next;
```

```
294.          // Making the index of 1st sorted array as NULL.
295.          slow->next = NULL;
296.          // Reccursive call for breaking further sub-lists.
297.          left = mergeSort(left);
298.          right = mergeSort(right);
299.          return merge(left, right);
300.      }
301.
302.      stud* search_algo :: merge(stud *L, stud *R) {
303.          // If left node is not present we will return the right one and
    vice-versa.
304.             if(!L) return R;
305.          if(!R) return L;
306.          // Initially storing arbitarary address in pointer h.
307.          stud *h = NULL;
308.          // Creating a new linked list.
309.             if(L->height < R->height) {
310.              h = L;
311.              h->next = merge(L->next, R);
312.          }else {
313.              h = R;
314.              h->next = merge(L, R->next);
315.          }
316.          // Returning head address of the new linked-list.
317.          return h;
318.      }
319.
320.      // Using selection matric for sorting
321.      void search_algo ::  Sorting_for_ordered_linear_search(stud *head){
322.
323.          for (stud* key = head; key->next != NULL; key = key->next)
324.          {
325.              stud* min_idx = key;
326.                    // For finding the minimum index
327.              for (stud* key2 = key->next; key2 != NULL; key2 = key2->next){
328.                     if (key2->height >= min_idx->height){
329.                         min_idx = key2;
330.                           }
331.                    }
332.                    // Swapping
333.                    float temp = min_idx->height;
334.                    min_idx->height = key->height;
335.                    key->height = temp;
336.                    string temp2 ;
337.                    temp2 = min_idx->name;
338.                    min_idx->name = key->name;
339.                    key->name = temp2;
340.             }
```

```
341.      }
342.
343.      // For Binary search.
344.      stud* search_algo :: binary_search(stud* head,float element,stud*
   arr[],int n){
345.          // First and last index of auxiliary array.
346.             int low = 0;
347.          int high = n-1;
348.          // Declaring msg variable to check whether the value is present or
   not.
349.             int msg = 0;
350.             // Binary search starts here.
351.          while(low <= high){
352.             // Finding the middle element
353.             int mid = low + ( (high-low)/2 ) ;
354.             head = arr[mid];
355.             // Checking
356.             if(head->height == element){
357.                    msg = 1;
358.                    return head;
359.                    }else if (element > head->height){
360.                           low = mid + 1;
361.                    }else{
362.                           high = mid - 1;
363.                    }
364.             }
365.             // If element not found print this message.
366.             if (msg == 0){
367.                    cout<<endl<<endl<<" Element "<<element<<" not found";
368.             }
369.      }
370.
371.      // Single Inheritance
372.      class first : protected search_algo
373.      {
374.             public:
375.                    // Polymorphism. and automatic initialisation as it is a
   constructor.
376.                    first(){
377.                           cout<<endl<<endl<<"\t\t*******Search
   Algorithms*********";
378.                           call();
379.                    }
380.      };
381.
382.      // The main function
383.      int main(){
384.             // For changing output bg.
```

```
385.            system("Color C0");
386.            // creating an object of child class.
387.            first obj;
388.            return 0;
389.      }
```

❖ C++ Output:

```
        *******Search Algorithms*********

# Enter the size of auxilary array:- 1100

 _______________________________ Welcome to the competition, ____________________________________________________

                                  Register yourself here.

# Enter your first name:- Joy

# Enter your height(inches):- 5.8


 Any more students(y/n):- y

# Enter your first name:- Kamal

# Enter your height(inches):- 6.0


 Any more students(y/n):- n



 ____________________________________ Admin control ____________________________________________________


 Please enter your user name: Tadeeb

 Please enter your user password: ******

                                        Welcome Tadeeb !

# Do you want to see all records(y/n):- n



 # Enter the least height required for qualification:- 5.1

# Enter your choice for searching:-
                      1. Unordered linear search
                      2. Ordered linear search
                      3. Binary search
 Enter here: 1
```

```
 # The list of eligible students is:-

                                    Name    Height
                                   _____________
                                  | Aarika   5.3|
                                  | Adora   5.2|
                                  | Adore   5.3|
                                  | Adoree   5.4|
                                  | Adorne   5.5|
                                  | Adrea   5.6|
                                  | Adria   5.7|
                                  | Adriaens   5.8|
                                  | Adrian   5.9|
                                  | Adriana   6|
                                  | Adriane   6.1|
                                  | Adrianna   6.2|
                                  | Adrianne   6.3|
                                  | Adriena   6.4|
                                  | Adrienne   6.5|
                                  | Aeriel   6.6|
                                  | Aeriela   6.7|
                                  | Aeriell   6.8|
                                  | Afton   6.9|
                                  | Ag   7|
                                  | Alaine   5.2|
                                  | Alameda   5.3|
                                  | Alana   5.4|
                                  | Alanah   5.5|
                                  | Alane   5.6|
                                  | Alanna   5.7|
                                  | Alayne   5.8|
                                  | Alberta   5.9|
                                  | Albertina   6|
                                  | Albertine   6.1|
                                  | Albina   6.2|
                                  | Alfreda   5.2|
 # Enter your choice for searching:-
                        1. Unordered linear search
                        2. Ordered linear search
                        3. Binary search
  Enter here: 2


 # Do you want to see the sorted records(y/n):- n


  # The list of eligible students is:-

                                     Name    Height
                                    _____________
                                   | Saurabh   8.6|
                                   | Naman   8|
                                   | Cherilynn   7|
                                   | Charleen   7|
                                   | Cathie   7|
                                   | Carmelita   7|
                                   | Camila   7|
                                   | Cacilie   7|
                                   | Britteny   7|
                                   | Brigida   7|

# Enter your choice for searching:-
                        1. Unordered linear search
                        2. Ordered linear search
                        3. Binary search
 Enter here: 3


# Do you want to see the sorted records(y/n):- n


 # The list of eligible students is:-

                                     Name    Height
                                    _____________
                                    |Ailis  5.1|
                                    |Christyna      5.2|
                                    |Cherrita       5.2|
                                    |Charlotte      5.2|
                                    |Celle  5.2|
                                    |Ceil   5.2|
                                    |Cathy  5.2|
                                    |Cassaundra     5.2|
                                    |Carley         5.2|
                                    |Cairistiona    5.2|
                                    |Brittney       5.2|
                                    |Brigitta       5.2|
                                    |Breena         5.2|
                                    |Barbi  5.2|
                                    |Ayn    5.2|
                                    |Auguste        5.2|
                                    |Athena         5.2|
                                    |Ardelia        5.2|
                                    |Annissa        5.2|
                                    |Anetta         5.2|
                                    |Anastasie      5.2|
                                    |Amargo         5.2|
                                    |Alys   5.2|
```

❖ Time Complexity Chart:

Algorithm Name	Best Case	Average Case	Worst Case
Unsorted linear search	Ω(1)	Ø(n)	O(n)
Sorted linear search	Ω(1)	Ø(lesser than n)	O(n)
Binary search	Ω(1)	Ø(logn)	O(logn)

❖ Conclusion:

The following conclusion are drawn after doing the assignment:

1) Binary Search is divide and conquer approach to search an element from the list of sorted element. In Linked List we can do binary search but it has time complexity O(n) that is same as what we have for linear search which makes Binary Search inefficient to use in Linked List.

2) The main problem that binary search takes O(n) time in Linked List due to fact that in linked list we are not able to do indexing which led traversing of each element in Linked list take O(n) time.

3) The method implemented in the assignment through which binary search can be done with time complexity of O(log2n). This is done with the help of auxiliary array.

4) Auxiliary array helps in indexing of linked list through which one can traverse a node in O (1) complexity hence reducing the complexity of binary search to O(log2n) hence increasing efficiency of binary search in linked List