MIT World Peace University

Fundamental Data Structures
Second Year B. Tech, Semester 1

Searching and Sorting Algorithms

Practical Report

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A2, PA 20

November 27, 2022

Contents

1 Objectives

1. To understand use of an array of structures for maintaining records
2. To implement, analyze and compare linear and binary search.
3. To implement, analyze and compare selection, insertion sort and shell sort.

2 Problem Statements

Write a C program to create a student database using an array of structures. Apply searching (Linear and Binary Search) and sorting techniques(Insertion Sort, Selection Sort, Shell sort).

3 Theory

3.1 Searching

3.1.1 Linear Search

Linear Search is defined as a sequential search algorithm that starts at one end and goes through each element of a list until the desired element is found, otherwise the search continues till the end of the data set. It is the easiest searching algorithm

3.1.2 Binary Search

Binary search can be implemented only on a sorted list of items. If the elements are not sorted already, we need to sort them first. Binary Search is a searching algorithm for finding an element's position in a sorted array. In this approach, the element is always searched in the middle of a portion of an array.

It can be implemented in 2 ways:

- Iterative method
- Recursive Method

3.2 Sorting

3.2.1 Selection Sort

The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from the unsorted part and putting it at the beginning.

The algorithm maintains two subarrays in a given array.

- The subarray which already sorted.
- The remaining subarray was unsorted.

In every iteration of the selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.

3.2.2 Insertion Sort

Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

- This algorithm is one of the simplest algorithm with simple implementation
- Basically, Insertion sort is efficient for small data values
- Insertion sort is adaptive in nature, i.e. it is appropriate for data sets which are already partially sorted.

3.2.3 Shell sort

Shell sort is a generalized version of the insertion sort algorithm. It first sorts elements that are far apart from each other and successively reduces the interval between the elements to be sorted.

The interval between the elements is reduced based on the sequence used. Some of the optimal sequences that can be used in the shell sort algorithm are:

- Shell's original sequence: N/2 , N/4 , ..., 1
- Hibbard's increments: 1, 3, 7, 15, 31, 63, 127, 255, 511
- Pratt: 1, 2, 3, 4, 6, 9, 8, 12, 18, 27, etc

4 Platform

Operating System: Arch Linux x86-64
IDEs or Text Editors Used: Visual Studio Code
Compilers : gcc on linux for C

5 Input

- Array of Structures, so data of students
- Roll number, Marks and Name
- Which element to Search
- Which Sort to use.

6 Output

- Menu to choose what to do
- Index of the Searched element, and its respective data
- Table of the entire Data sorted by chosen Algorithm.

7 Test Conditions

1. Input at least five records.
2. Search roll no which is not present using linear search and binary search
3. Search roll no which is present at different locations using linear search and binary search.
4. Test three sorting methods for at least five records.

8 Code

8.1 Pseudo Code

8.1.1 Pseudo Code for Implementation of Linear Search

```
void linear_search(int a[], int n)
{
  //a is the array
  //n is the number to find
  int l, flag = 0;
  for(int i = 0;i < n;i++)
  {
    if(a[i] == n)
    {
      flag = 1;
      break;
    }
  }
  if(flag == 0)
  {
    return -1;
  }
}
```

8.1.2 Pseudo Code for Implementation of Binary Search

```
int binary_search_recursive(a[], int size, int low, int high, int key)
{
  int mid;
  if (low <= high)
  {
    mid = (low + high) / 2;
    if (a[mid] == key)
    {
      return mid;
    }
    else if (a[mid] > key)
    {
      return binary_search_recursive(a, size, low, mid - 1, key);
    }
    else if (a[mid] < key)
    {
      return binary_search_recursive(a, size, mid + 1, high, key);
    }
  }
  return -1;
}
```

8.1.3 Pseudo Code for Implementation of Bubble Sort

```
void Bubble_sort(int a[], int size, int key)
{
  for(int i = 0; i < size - 1; i++)
  {
    for(int j = 0; j < size - 2; j++)
    {
      if(a[j] > a[j + 1])
      {
        swap(a[j], a[j + 1])
      }
    }
  }
}
```

8.1.4 Pseudo Code for Implementation of Insertion Sort

```
void insertion_sort(int a[], int size, int key)
{
  for(i to size)
  {
    key = a[i]
    j = i - 1
    while(j >= 0 and a[j] > key)
    {
      a[j + 1] = a[j];
      j = j - 1;
    }
    a[j + 1] = key;
  }
}
```

8.1.5 Pseudo Code for Implementation of Selection Sort

```
void selection_sort(int arr[], int size, int key)
{
  for i to size
  {
    min_pos = i;
    for j = i + 1 to size - 1
    {
      if a[j] < a[min_pos]
      {
        min_pos = j;
      }
    }
  }
  if(min_pos != i)
  {
    swap(a[i], a[min_pos])
  }
}
```

8.2 C Implementation of Problem Statement

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

// linear search, binary search, selection sort, insertion sort, shell sort.

struct Student
{
    int roll_no;
    char name[100];
    int marks;
};

void accept_array(struct Student students[], int number_of_students)
{
    for (int i = 0; i < number_of_students; i++)
    {
        printf("Enter the roll number : ");
        scanf("%d", &students[i].roll_no);
        printf("\nEnter the Name: ");
        scanf("%s", students[i].name);
        printf("Enter the marks");
        scanf("%d", &students[i].marks);
        printf("\n");
    }
}

void display(struct Student students[], int number_of_students, int i)
{

    printf("-------------------------\n");
    printf("|\tNo|\tName|\tMarks|\n");
    for (; i < number_of_students; i++)
    {
        printf("|\t%d|\t%s|\t%d|\n", students[i].roll_no, students[i].name,
    students[i].marks);
    }
    printf("---------------------------\n");
}

void swap(struct Student *a, struct Student *b)
{
    struct Student temp = *a;
    *a = *b;
    *b = temp;
}

int linear_search(struct Student students[], int number_of_students, int key)
{
    int linear_search_result_key = 0; // 1
    int count = 0;                    // 1
    if (key < 0)                      // 1
    {
        return -1;
    }
    for (int i = 0; i < number_of_students; i++) // n+1
    {
        count++;
        if (key == students[i].roll_no)
        {
            return i;
        }
    }
    if (count >= number_of_students) // 1
    {
        return -1;
    } // n + 5
} // time complexity is O(n)

int binary_search_iterative(struct Student students[], int number_of_students, int
     key)
{
    int low = 0;
    int high = number_of_students - 1;
    while (low <= high)
    {
        int mid = (low + high) / 2;
        if (students[mid].roll_no == key)
        {
            return mid;
        }
        else if (students[mid].roll_no > key)
        {
            high = mid - 1;
        }
        else if (students[mid].roll_no < key)
        {
            low = mid + 1;
        }
    }
    return -1;
}

int binary_search_recursive(struct Student students[], int number_of_students, int
     low, int high, int key)
{
    int mid;
    if (low <= high)
    {
        mid = (low + high) / 2;
        if (students[mid].roll_no == key)
        {
            return mid;
        }
        else if (students[mid].roll_no > key)
        {
            return binary_search_recursive(students, number_of_students, low, mid
    - 1, key);
        }
        else if (students[mid].roll_no < key)
        {
            return binary_search_recursive(students, number_of_students, mid + 1,
    high, key);
        }
    }
    return -1;
```

```c
}

int selection_sort(struct Student students[], int number_of_students)
{
    int min;
    for (int i = 0; i < number_of_students - 1; i++)
    {
        // assign the minimum element equal to i
        min = i;
        // find the minimum element in the rest of the array
        for (int j = i + 1; j < number_of_students; j++)
        {
            if (students[j].marks < min)
            {
                min = j;
            }
        }

        // swap with the first element, coz its min.
        if (min != i)
        {
            swap(&students[i], &students[min]);
            // swap(students, i, min);
        }
    }
}

int insertion_sort(struct Student students[], int number_of_students)
{
    int j, key;
    for (int i = 0; i < number_of_students; i++)
    {
        key = students[i].marks;
        j = i - 1;
        while (j >= 0 && students[j].marks > key)
        {
            swap(&students[j + 1], &students[j]);
            j--;
        }
    }
}

int shell_sort(struct Student students[], int number_of_students)
{
    int gap = number_of_students / 2;
    int swapped;
    do
    {
        do
        {
            swapped = 0;
            for (int i = 0; i < number_of_students - gap; i++)
            {
                if (students[i].marks > students[i + gap].marks)
                {
                    swap(&students[i], &students[i + gap]);
                    swapped = 1;
                }
            }
        } while (swapped == 1);

    } while ((gap = gap / 2) >= 1);
}

int main()
{
    int choice = 0;
    int linear_search_key = 0, linear_search_result_key = 0, binary_search_key =
    0, binary_search_result_key = 0;
    int number_of_students = 3;
    struct Student students[3] =
        {
            {1, "a", 90},
            {2, "b", 100},
            {3, "c", 70}};

    accept_array(students, number_of_students);
    display(students, 3, 0);

    printf("\n\n Welcome to Assignment 2\n");
    printf("What do you want to do? \n");
    printf("1. Linear Search an element\n2. Binary Search (Iterative) an Element\
    n3. Binary Search (Recursive)\n4. Sort with Selection Sort\n5. Sort with
    Insertion Sort\n6. Sort with Shell Sort\n");
    scanf("%d", &choice);
    switch (choice)
    {
    case 1:
        printf("Enter The Roll number of the student : ");
        scanf("%d", &linear_search_key);
        linear_search_result_key = linear_search(students, 3, linear_search_key);
        if (linear_search_result_key < 0)
        {
            printf("\nRecords not found!\n");
        }
        else
        {
            linear_search_result_key++;
            printf("The Key was found at position : %d", linear_search_result_key)
    ;
            display(students, linear_search_result_key, linear_search_result_key -
     1);
        }
        printf("\n");
        break;
    case 2:
        printf("Enter The Roll number of the student : ");
        scanf("%d", &binary_search_key);
        binary_search_result_key = binary_search_iterative(students,
    number_of_students, binary_search_key);
        if (binary_search_result_key < 0)
        {
            printf("\nRecords not found!\n");
        }
        else
        {
            binary_search_result_key++;
            printf("The Key was found at position : %d", binary_search_result_key)
```

```c
;
            printf("\n");
            display(students, binary_search_result_key, binary_search_result_key - 1);
        }
        printf("\n");
        break;
    case 3:
        printf("Enter The Roll number of the student : ");
        scanf("%d", &binary_search_key);
        binary_search_result_key = binary_search_recursive(students, number_of_students, 0, number_of_students - 1, binary_search_key);
        if (binary_search_result_key < 0)
        {
            printf("\nRecords not found!\n");
        }
        else
        {
            binary_search_result_key++;
            printf("The Key was found at position : %d", binary_search_result_key);
            printf("\n");
            display(students, binary_search_result_key, binary_search_result_key - 1);
        }
        printf("\n");
        break;

    case 4:
        printf("\nHere are the Students sorted by their marks with Selection sort\n");
        selection_sort(students, number_of_students);
        display(students, number_of_students, 0);
        break;

    case 5:
        printf("\nHere are the Students sorted by their marks with Insertion sort\n");
        insertion_sort(students, number_of_students);
        display(students, number_of_students, 0);
        break;
    case 6:
        printf("\nHere are the Students sorted by their marks with Shell sort\n");
        shell_sort(students, number_of_students);
        display(students, number_of_students, 0);
        break;
    default:
        printf("\nWrong value entered. \n");
        break;
    }

    return 0;
}
```

Listing 1: Main.Cpp

8.3 C Output

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

```
|       No|     Name|   Marks|
|       1|      a|      90|
|       2|      b|      100|
|       3|      c|      70|
---------------------------


 Welcome to Assignment 2
What do you want to do?
1. Linear Search an element
2. Binary Search (Iterative) an Element
3. Binary Search (Recursive)
4. Sort with Selection Sort
5. Sort with Insertion Sort
6. Sort with Shell Sort
1
Enter The Roll number of the student : 2
The Key was found at position : 2
--------------------------
|       No|     Name|   Marks|
|       2|      b|      100|
---------------------------

2
Enter The Roll number of the student : 2
The Key was found at position : 2
--------------------------
|       No|     Name|   Marks|
|       2|      b|      100|
---------------------------

4

Here are the Students sorted by their marks with Selection sort
--------------------------
|       No|     Name|   Marks|
|       3|      c|      70|
|       1|      a|      90|
|       2|      b|      100|
---------------------------

5

Here are the Students sorted by their marks with Insertion sort
--------------------------
|       No|     Name|   Marks|
|       3|      c|      70|
|       1|      a|      90|
|       2|      b|      100|
---------------------------

6

Here are the Students sorted by their marks with Shell sort
--------------------------
|       No|     Name|   Marks|
|       3|      c|      70|
|       1|      a|      90|
|       2|      b|      100|
--------------------------
```

Listing 2: Outpute

9 Time Complexity

- Time Complexity of Linear Search is: Ω(1), O(n)
- Time Complexity of Binary Search is: $\Omega(1)$, $O(\log(n))$
- Time Complexity of Bubble Sort is: $\Omega(n)$, $O(n^2)$
- Time Complexity of Insertion Sort is: $\Omega(n)$, $O(n^2)$
- Time Complexity of Selection Sort is: $\Omega(n^2)$, $O(n^2)$
- Time Complexity of Shell Sort is: $\Omega(n\log(n))$, $O(n^2)$

10 Conclusion

Thus, implemented different searching and sorting methods on the student database. This System is able to perform searching and sorting under different cases.

11 FAQs

1. **What is the meaning of database? How to maintain in C ?**
   A Database is simple a collection of data kept securely and readily accessible by the CPU, program or the user, to read or write to as many times as needed. It can be maintained in C is a few ways:
   - Using a Simple Structure, and then creating an array of structure objects.
   - Using File IO operations in C, to read to and write from files.
   - Arrays, linked lists, and other data structures, which could be created at runtime or compile time for ready processing or be transferred to files to act as long term databases.
2. **What are the applications of searching and sorting?**
   Here are some Applications of Searching Algorithms:
   (a) Find an element in a sorted array
   (b) To find if n is a square of an integer
   (c) Find the first value greater than or equal to x in a given array of sorted integers
   (d) Find the frequency of a given target value in an array of integers
   (e) Find the peak of an array which increases and then decreases
   (f) A sorted array is rotated n times. Search for a target value in the array.
   (g) Dictionary
   (h) Debugging a linear piece of code
   (i) Figuring out resource requirements for a large system
   (j) Find values in sorted collection

(k) Semiconductor test programs

(l) Numerical solutions to an equation

Here are some Applications of Sorting Algorithms:

(a) Sort a list of names.

(b) Organize an MP3 library.

(c) Display Google PageRank results.

(d) List RSS news items in reverse chronological order.

(e) Find the median.

(f) Find the closest pair.

(g) Binary search in a database.

(h) Identify statistical outliers.

(i) Find duplicates in a mailing list.

(j) Data compression.

(k) Computer graphics.

(l) Computational biology.

(m) Supply chain management.

(n) Load balancing on a parallel computer

Sorting Algorithms in Particular have some unique features that help them to be used in niche fields of science and Real life Application. Some are :

(a) Selection Sort - Selection sort does not require a lot of memory so it can be used where memory is a constraint. Since memory is not a big problem nowadays, selection sort is rarely used.

(b) Insertion Sort - Insertion sort is used in sort() function in java along with quick sort. When the data to be sorted is reduced to a small size, insertion sort works great.

(c) Shell Sort - Shell sort is used when calling a stack is overhead. It is used in C standard libraries. Also shell sort is used in linux kernels.

(d) Merge Sort - Variant of merge sort i.e. tim sort, is used in standard sorting algorithm of python. It can be used for sorting linked lists.

(e) Heap Sort - Heap sort is used in the implementation of priority queues. It is also used in embedded systems and systems concerned with security.

(f) Quick Sort - It is used in traditional java sorting. It is also used in event driven simulation.

(g) Radix Sort - Radix sort can be used where data is to be sorted in lexicographical order like strings or numbers. It can be used when numbers are in a large range and sorting them using counting sort will take more time.

3. **Compare and contrast linear search and binary search?**

Linear Search	**Binary Search**
In linear search input data need not to be in sorted.	In binary search input data need to be in sorted order.
It is also called sequential search.	It is also called half-interval search.
The time complexity of linear search O(n).	The time complexity of binary search O(log n).
Multidimensional array can be used.	Only single dimensional array is used.
Linear search performs equality comparisons	Binary search performs ordering comparisons
It is less complex.	It is more complex.
It is very slow process.	It is very fast process.

4. **Compare and contrast bubble, selection, insertion sort and shell sort.**

(a) *Bubble Sort*:
Time Complexity:
Best Case Sorted array as input. Or almost all elements are in proper place. [O(N)]. O(1) swaps.
Worst Case: Reversely sorted / Very few elements are in proper place. [O(N2)] . O(N2) swaps.
Average Case: [O(N2)] . O(N2) swaps.

Space Complexity: A temporary variable is used in swapping [auxiliary, O(1)]. Hence it is In-Place sort.
Advantage:

- It is the simplest sorting approach.
- Best case complexity is of O(N) [for optimized approach] while the array is sorted.
- Using optimized approach, it can detect already sorted array in first pass with time complexity of O(N).
- Stable sort: does not change the relative order of elements with equal keys.
- In-Place sort.

Disadvantage:
Bubble sort is comparatively slower algorithm.

(b) *Selection Sort*:
Time Complexity:
Best Case [O(N2)]. And O(1) swaps.
Worst Case: Reversely sorted, and when the inner loop makes a maximum comparison. [O(N2)] . Also, O(N) swaps.
Average Case: [O(N2)] . Also O(N) swaps.

Space Complexity: [auxiliary, O(1)].
In-Place sort.(When elements are shifted instead of being swapped (i.e. temp=a[min], then shifting elements from ar[i] to ar[min-1] one place up and then putting a[i]=temp). If swapping is opted for, the algorithm is not In-place.)

Advantage:

- It can also be used on list structures that make add and remove efficient, such as a linked list. Just remove the smallest element of unsorted part and end at the end of sorted part.
- The number of swaps reduced. O(N) swaps in all cases.
- In-Place sort.

Disadvantage:
Time complexity in all cases is O(N2), no best case scenario.

(c) *Insertion Sort*:
Time Complexity:
Best Case Sorted array as input, [O(N)]. And O(1) swaps.
Worst Case: Reversely sorted, and when inner loop makes maximum comparison, [O(N2)] . And O(N2) swaps.
Average Case: [O(N2)] . And O(N2) swaps.

Space Complexity: [auxiliary, O(1)]. In-Place sort.

Advantage:

- It can be easily computed.
- Best case complexity is of O(N) while the array is already sorted. Number of swaps reduced than bubble sort.
- For smaller values of N, insertion sort performs efficiently like other quadratic sorting algorithms. Stable sort.
- Adaptive: total number of steps is reduced for partially sorted array. In-Place sort.

Disadvantage:
It is generally used when the value of N is small. For larger values of N, it is inefficient.

(d) Shell Sort:
Time Complexity:
Best Case : When the given array list is already sorted the total count of comparisons of each interval is equal to the size of the given array. So best case complexity is O(n log(n))
Worst Case: The worst-case complexity for shell sort is O(n2)
Average Case: The shell sort Average Case Complexity depends on the interval selected by the programmer. O(n log(n)2).

Space Complexity: The space complexity of the shell sort is O(1).
Advantage:

- Replacement for insertion sort, where it takes long time to complete given task.
- To call stack overhead we use shell sort.
- when recursion exceeds a particular limit we use shell sort.
- For medium to large-sized datasets.
- In insertion sort to reduce the number of operations .