# Data Structure and Algorithms

Deepali Londhe
Information Technology,
PICT, Pune

1

## Unit- I Introduction (06 Hrs)

2

- Introduction to Data Structures: Concept of data, Data object, Data structure, Concept of Primitive and non-primitive, linear and Nonlinear, static and dynamic, persistent and ephemeral data structures
- Definition of ADT, Array: Single and multidimensional array address calculation, recursion.
- Searching and sorting: Need of searching and sorting, Concept of internal and external sorting, sort stability
- Searching methods: Linear and binary search algorithms, Fibonacci Series.
- Sorting methods: Bubble, insertion, Quick, Merge, shell and comparison of all sorting methods.
- Case Studies  Set Operation, String Operation

DSA Unit-1.3 Searching

2

# Contents

3

| Section | Contents |
|---|---|
| DSA Unit-I.1 | Introduction to Data Structures, its types |
| DSA Unit-I.2 | Definition of ADT, Array |
| DSA Unit-I.3 | Searching and sorting- Searching |
| DSA Unit-I.4 | **Sorting Methods** |

DSA Unit-1.3 Searching

3

# Agenda

4

- Searching and sorting
- Need of searching and sorting
- Concept of internal and external sorting
- Sort stability
- Searching methods: Linear and binary search algorithms
- Fibonacci Series.

DSA Unit-1.3 Searching

4

# Outcomes

5

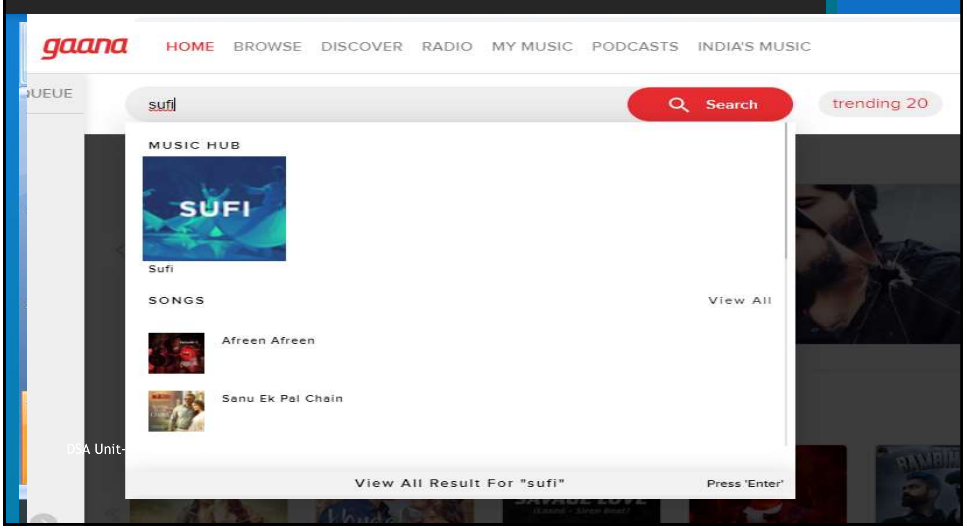
- What is searching?
- List different Searching Methods
- Explain Sequential Search with Algorithm
- Explain Binary Search with Algorithm
- Discuss efficiency of Sequential and Binary Search

DSA Unit-1.3 Searching

5

# Searching

6




6

## Searching..

7

- Storage and Retrieval of Information
- Table Look-Up
- We are concerned with the process of collecting information in a computer's memory, in such a way that the information can subsequently be recovered as quickly as possible.
- We want to get search information as fast as possible.

DSA Unit-1.3 Searching

7

## Searching..

8

- Locating the element in the given list.
- List can be represented using
  - Array
  - Linked list
  - Tree
  - Heap
  - File
- Search key in sequentially by comparing to every record. If found then search successful otherwise unsuccessful.
- This is called sequential search.

DSA Unit-1.3 Searching

8

## Searching and Sorting

9

- Searching- We shall suppose that a set of *N records has been stored, and the* problem is to locate the appropriate one.
- Sorting- We assume that each record includes a special field called its *key; this terminology is* especially appropriate, because many people spend a great deal of time every day searching for their keys.
- We generally require the *N keys to be distinct, so* that each key uniquely identifies its record.
- The collection of all records is called a *table or file, where the word "table" is usually used to indicate a small file, and* "file" is usually used to indicate a large table.
- A large file or a group of files is frequently called a *database*.

DSA Unit-1.3 Searching

9

## Searching and Sorting..

10

- Searching and sorting are often closely related to each other.
- For example, consider the following problem:
- *Given two sets of numbers, A = {a1, a2, . . ., am} and B = {b1, b2, . . ., bn}, determine whether or not A ⊆ B.*
- ***Three solutions*:**

1. **Compare each *ai sequentially with the bj's until finding a match.***
2. **Sort the *a's and b's, then make one sequential pass through both files,*** checking the appropriate condition.
3. **Enter the *bj's in a hash table, then search for each of the ai.***

Sol-[illegible] *the internal* memory size

DSA Unit-1.3 Searching

10

## Sequential Searching

11

- Begin at the beginning, and go on till you find the right key; then stop.

**Algorithm S** (*Sequential search*). Given a table of records $R_1, R_2, \ldots, R_N$, whose respective keys are $K_1, K_2, \ldots, K_N$, this algorithm searches for a given argument $K$. We assume that $N \geq 1$.

**S1.** [Initialize.] Set $i \leftarrow 1$.

**S2.** [Compare.] If $K = K_i$, the algorithm terminates successfully.

**S3.** [Advance.] Increase $i$ by 1.

**S4.** [End of file?] If $i \leq N$, go back to S2. Otherwise the algorithm terminates unsuccessfully. ▮

- Algorithm can terminate in two different ways, *successfully* (having located the desired key) or *unsuccessfully (having established that the* given argument is not present in the table)



11

## Sequential Searching flow

12

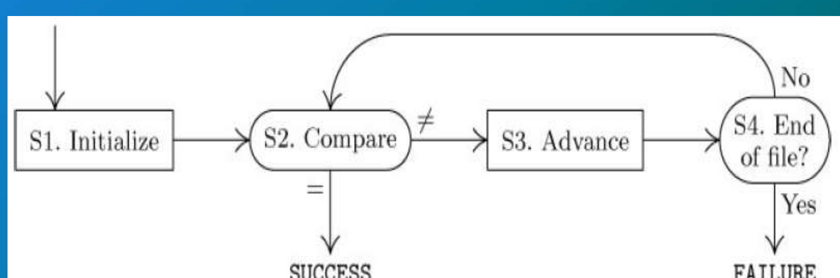



**Fig. 1.** Sequential or "house-to-house" search.



12

# Sequential Search: Algorithm

13

- Seq_search(a,n,k)

  //Search for key k in array a of n elements

```
i=0;
While (a[i]≠k && i<n)
  i=i+1;
If (i<n)
  print(Number found)
```





# Sequential Search: Analysis

14

- Seq_search(a,n,k)
  - Search for key k in array a of n elements

| | cost |
|---|---|
| i=0; | 1 |
| While (a[i]≠k && i<n) | n+1, n+1 |
|   i=i+1; | n |
| If (i<n) | 1 |
|   print(Number found) | 1 |

- Total cost
  - 1+2n+2+n+2=3n+5
- Time complexity: O(n)

## Different Cases

15

- The total cost of sequential search is 3n + 5
  - But is it always exactly 3n + 5 instructions?
  - The last assignment does not always execute
    - But does one assignment really matter?
  - How many times will the loop actually execute?
    - that depends
  - If searchID is found at index 0: _____ iterations
    - best case
  - If searchID is found at index n-1:_____ iterations
    - worst case
  - If searchID is found at index n/2:_____ iterations
    - Average case

DSA Unit-1.3 Searching

15

| Statement | s/e | Frequency | Total steps |
|---|---|---|---|
| `int sequentialSearch(···)` | 0 | 0 | 0 |
| `{` | 0 | 0 | 0 |
| `int i;` | 1 | 1 | 1 |
| `for (i = 0; i < n && x != a[i]; i++);` | 1 | 1 | 1 |
| `if (i == n) return -1;` | 1 | 1 | 1 |
| `else return i;` | 1 | 1 | 1 |
| `}` | 0 | 0 | 0 |
| Total | | | 4 |

**TABLE 1.1** Best-case step count for Figure 1.3

16

Best case: O(1)

| Statement | s/e | Frequency | Total steps |
|---|---|---|---|
| `int sequentialSearch(···)` | 0 | 0 | 0 |
| `{` | 0 | 0 | 0 |
| `int i;` | 1 | 1 | 1 |
| `for (i = 0; i < n && x != a[i]; i++);` | 1 | $n+1$ | $n+1$ |
| `if (i == n) return -1;` | 1 | 1 | 1 |
| `else return i;` | 1 | 0 | 0 |
| `}` | 0 | 0 | 0 |
| Total | | | $n+3$ |

**TABLE 1.2** Worst-case step count for Figure 1.3

Worst case: O(n)

| Statement | s/e | Frequency | Total steps |
|---|---|---|---|
| `int sequentialSearch(···)` | 0 | 0 | 0 |
| `{` | 0 | 0 | 0 |
| `int i;` | 1 | 1 | 1 |
| `for (i = 0; i < n && x != a[i]; i++);` | 1 | $j+1$ | $j+1$ |
| `if (i == n) return -1;` | 1 | 1 | 1 |
| `else return i;` | 1 | 1 | 1 |
| `}` | 0 | 0 | 0 |
| Total | | | $j+4$ |

**TABLE 1.3** Step count for Figure 1.3 when `x = a[j]`

Average case:
O(n/2) ~ O(n)

16

## Searching an Ordered Table

17

- Find the name of the person whose number is 865-7923 in Telephone Directory
- Sequential search
- Much easier to find an entry by the party's name, instead of by number
- When a large file must be searched, sequential scanning is almost out of the question, but an ordering relation simplifies the job enormously.

DSA Unit-1.3 Searching

17

## Binary Search

18

- Binary search can be a more efficient algorithm for searching
- It works only on sorted arrays like this
  - **Compare the element in the middle**
  - **if that's the target, quit and report success**
  - **if the key is smaller, search the array to the left**
  - **otherwise search the array to the right**
- This process repeats until the target is found or there is nothing left to search
- Each comparison narrows search by half, lg *N comparisons, we will have found the key or we will have established* that it is not present. This procedure is sometimes known as "logarithmic search" or "bisection," but it is most commonly called *binary search*.

DSA Unit-1.3 Searching

18

## Binary Search Algorithm

19

**Algorithm B** (*Binary search*). Given a table of records $R_1, R_2, \ldots, R_N$ whose keys are in increasing order $K_1 < K_2 < \cdots < K_N$, this algorithm searches for a given argument $K$.

**B1.** [Initialize.] Set $l \leftarrow 1$, $u \leftarrow N$.

**B2.** [Get midpoint.] (At this point we know that if $K$ is in the table, it satisfies $K_l \le K \le K_u$. A more precise statement of the situation appears in exercise 1 below.) If $u < l$, the algorithm terminates unsuccessfully. Otherwise, set $i \leftarrow \lfloor (l+u)/2 \rfloor$, the approximate midpoint of the relevant table area.

**B3.** [Compare.] If $K < K_i$, go to B4; if $K > K_i$, go to B5; and if $K = K_i$, the algorithm terminates successfully.

**B4.** [Adjust $u$.] Set $u \leftarrow i - 1$ and return to B2.

**B5.** [Adjust $l$.] Set $l \leftarrow i + 1$ and return to B2. ▮

19

## Binary Search..

20

two pointer
*l-current lower and u*-upper
limits for the search

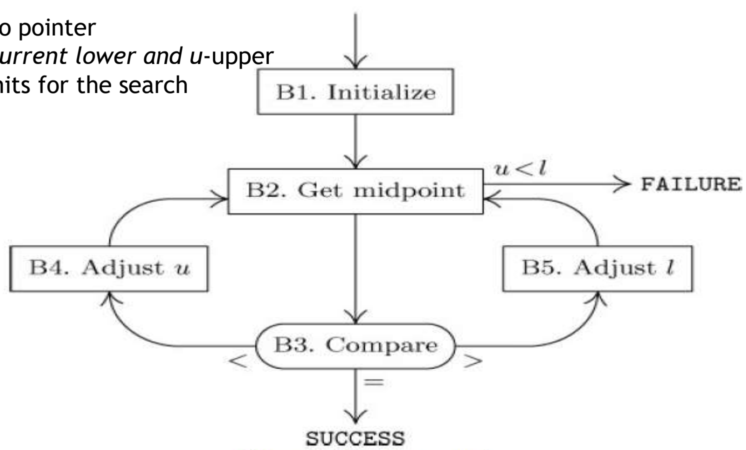



**Fig. 3.** Binary search.

20

## Binary Search Hiten

21

| Data | reference | pass 1 | pass 2 |
|---|---|---|---|
| Bina | a[0] | low | |
| Chinu | a[1] | | |
| Deepak | a[2] | | |
| Evleen | a[3] | | |
| Fatima | a[4] | mid | |
| Ganesh | a[5] | | low |
| Hiten | a[6] | | mid |
| Irra | a[7] | | |
| Jeet | a[8] | high | high |

DSA Unit-1.3 Searching

21

## Binary Search Example

22

a) Searching for 653:

[061 087 154 170 275 426 503 <u>509</u> 512 612 653 677 703 765 897 908]
061 087 154 170 275 426 503 509 [512 612 653 <u>677</u> 703 765 897 908]
061 087 154 170 275 426 503 509 [512 <u>612</u> 653] 677 703 765 897 908
061 087 154 170 275 426 503 509 512 612 [<u>653</u>] 677 703 765 897 908

b) Searching for 400:

[061 087 154 170 275 426 503 <u>509</u> 512 612 653 677 703 765 897 908]
[061 087 154 <u>170</u> 275 426 503] 509 512 612 653 677 703 765 897 908
061 087 154 170 [275 <u>426</u> 503] 509 512 612 653 677 703 765 897 908
061 087 154 170 [<u>275</u>] 426 503 509 512 612 653 677 703 765 897 908
061 087 154 170 275] [426 503 509 512 612 653 677 703 765 897 908

22

# Binary Search algorithm

23

```
int binary_search(int key, int arr[], int size)
{  int low = 0, high = size, mid;
   while(low<=high)
   {        mid = (low + high) / 2;
            if(arr[mid] < Key)
                    low = mid + 1;
            else
                if(arr[mid] > val)
                    high = mid - 1;
                else
                    return mid;
   }
   return -1;
}
```

DSA Unit-1.3 Searching

23

# No of Comparisons

24

- 16
- 50
- 256
- 1000
- 10000
- 100000
- 1000000

DSA Unit-1.3 Searching

24

# How fast is Binary Search?

25

- Best case: 1
- Worst case: when target is not in the array
- At each pass, the "live" portion of the array is narrowed to half the previous size.
- The series proceeds like this:
  - n , n/2, n/4, n/8, ...
- Each term in the series represents one comp-arison. How long does it take to get to 1?
  - This will be the number of comparisons

DSA Unit-1.3 Searching

25

# Binary Search..

26

- Could start at 1 and double until we get to n

| | |
|---|---|
| 1 | $2^0$ |
| 2 | $2^1$ |
| 4 | $2^2$ |
| 8 | $2^3$ |
| 16 | $2^4$ |
| : | : |
| K>=n | $2^c$ >=n |

1, 2, 4, 8, 16, ... , *k* >= *n* or
20, 21, 22, 23, 24, ... , 2*c* >= *n*

- The length of this series is c+1
- The question is
  - 2 to what power *c* is greater than or equal to n?
    - if n is 8, *c* is 3
    - if n is 1024, *c* is 10
    - if n is 16,777,216, *c* is 24
- Binary search is O( log n ) *base 2 assumed*

DSA Unit-1.3 Searching

26

## Comparing O(n) to O(log n)

27

Rates of growth and logarithmic functions

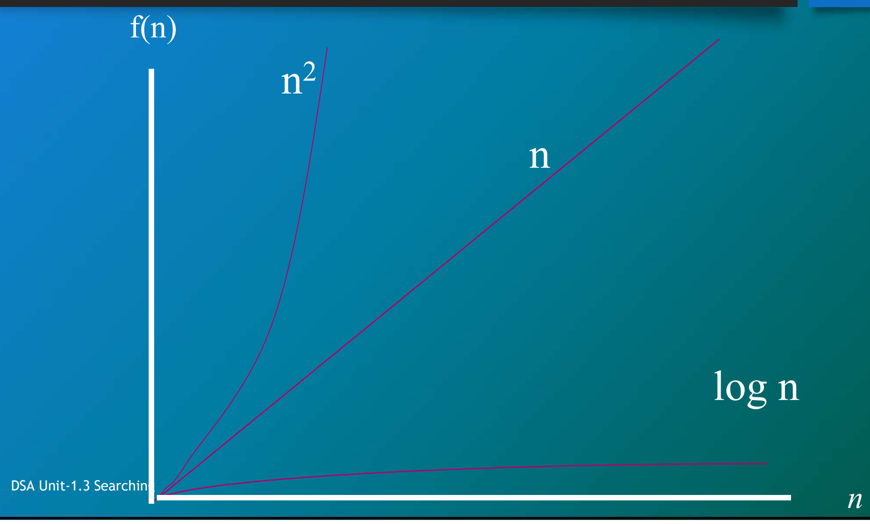
| Power of 2 | n | $\log_2 n$ |
|---|---|---|
| $2^4$ | 16 | 4 |
| $2^8$ | 128 | 8 |
| $2^{12}$ | 4,096 | 12 |
| $2^{24}$ | 16,777,216 | 24 |

DSA Unit-1.3 Searching

27

## Graph Illustrating Relative Growth n, log n, $n^2$

28




DSA Unit-1.3 Searchin

28

# Sequential Vs Binary Search

29

Table 2-1 Comparison of binary and sequential searches

| Size | Binary | Sequential (Average) | Sequential (Worst Case) |
|---|---|---|---|
| 16 | 4 | 8 | 16 |
| 50 | 6 | 25 | 50 |
| 256 | 8 | 128 | 256 |
| 1000 | 10 | 500 | 1000 |
| 10,000 | 14 | 5000 | 10,000 |
| 100,000 | 17 | 50,000 | 100,000 |
| 1,000,000 | 20 | 500,000 | 1,000,000 |

DSA Unit-1.3 Searching

29

# References

30


- **Books**
- D. E. Knuth, *The Art of Computer Programming: Vol. 3: Sorting and Searching, 2d ed., Addison-* Wesley, Reading, Mass., 1998.
- Samanta Debasis, **"CLASSIC DATA STRUCTURES", PHI, 2nd ed.**
- Ellis Horowitz and Sartaj Sahni , "Fundamentals of Data Structures", Computer Science Press, 1983
- R. Gilberg, B. Forouzan, "Data Structures: A pseudo Code Approach with C++", Cengage Learning, ISBN 9788131503140.
- E. Horowitz, S. Sahni, D. Mehta, "Fundamentals of Data Structures in C++", Galgotia Book Source, New Delhi, 1995, ISBN 16782928
- Dinesh P. Shah, Sartaj Sahani , "Handbook of DATA STRUCTURES and APPLICATIONS", CHAPMAN & HALL/CRC
- Bayer B. et al. (2015) Electro-Mechanical Brake Systems. In: Winner H., Hakuli S., Lotz F., Singer C. (eds) Handbook of Driver Assistance Systems. Springer, Cham
- **Web**
  - http://statmath.wu.ac.at/courses/data-analysis/itdtHTML/node55.html
  - https://en.wikipedia.org/wiki/Persistent_data_structure


DSA Unit-1.3 Searching



30


31