SEARCHING

- Searching method are governed by how data are stored in a computer, either in internal memory or in external memory.

- Searching methods are also decided by the Data Structure used to store information. In addition to this, searching methods are also decided by search over a small set of data to a large set of data.

- Searching is generally a "MOST-TIME CONSUMING TASK" in many applications and application of a good searching method eventually leads to a substaintial increase in performance.

- Searching
  - Internal Searching
    - Search with Key Comparison
      - Linear Search
        - Sequential Search
        - Binary Search
        - Interpolation Search
      - Non-linear Search
        - Tree Search
          - B.S.T
          - AVL
          - R-B Tree
          - Splay Tree
          - Digital Search
        - Multiway Tree Search
          - m-way Tree Search
          - B-tree Search
        - Graph Search
          - Depth first Search
          - Breadth first Search
    - Search w/o Key comparison
      - Address Calculation Search.
  - External Searching
    - B-tree Searching
    - B+tree Searching

LINEAR SEARCH TECHNIQUES

→ Searching methods involving data stored in the form of a linear data structure like array, linked list are called "**LINEAR SEARCH METHODS**".

→ 4 important linear search techniques:-

1. Sequential Search with Array.
2. Sequential Search with linked list.
3. Binary Search.
4. Interpolation.

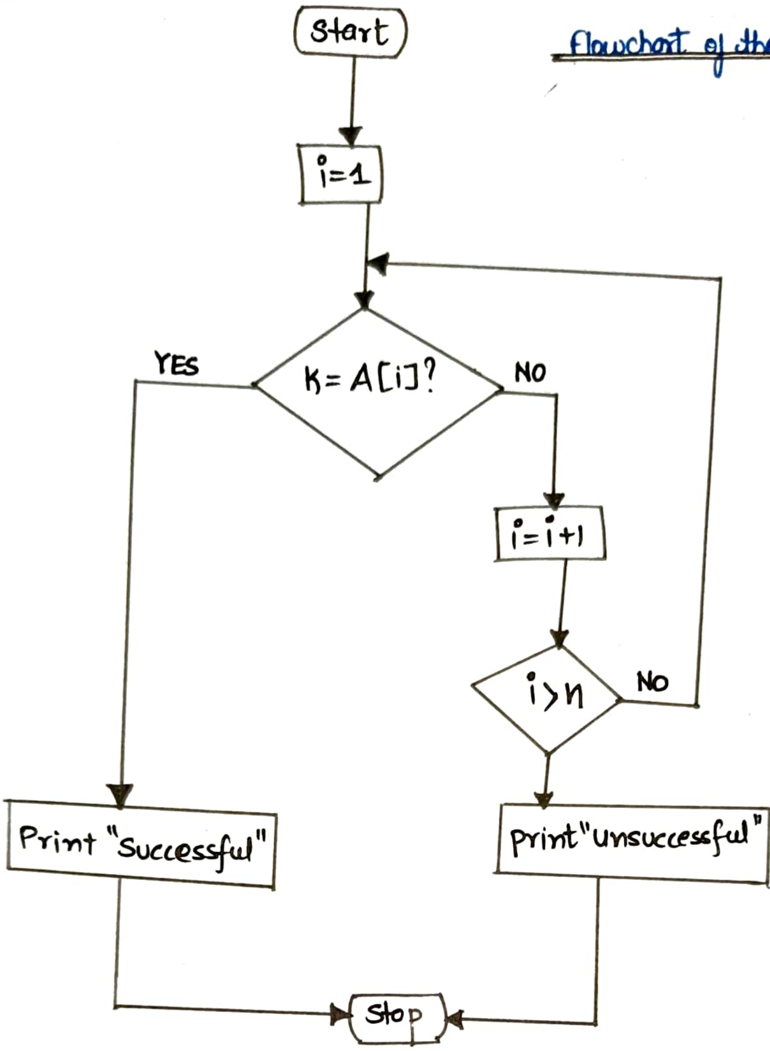
Linear Search with Array

→ This searching method is applicable when data are stored in an array. The basic principle of this searching method is to begin at the beginning, then sequentially continue the search until the right key is found or reached at the end of the array. Algorithm terminates whichever occurs first.

Steps :→

```
1.  i=1
2.  If (K=A[i]) then            // Begin search from the first location.
3.     Print 'Successful' at location i
4.     Goto Step 14.            // Termination of a successful search
5.  ELSE
6.     i=i+1
7.     if (i≤n) then
8.        Goto Step2            // Search from the next item
9.     Else
10.       print "Unsuccessful"
11.       Goto Step 14.  // Termination of an unsuccessful search.
12.    EndIf
13. EndIf
14. Stop
```

Flowchart of the algorithm Search Array



Complexity Analysis of the algorithm Search Array

Case 1 Key matches with the first element.

⇒ This is the case when the key is present in the array as the first element, that is, A[1]. Only one comparison in this case,

Thus T(n) the no. of comparisons is

$$\boxed{T(n) = 1}$$

Best case of the algorithm.

Case 2. Key does not exists

This the case of searching when the key is not present in the array. In this case, the else part of the if-statement in step 2 is executed for A[1], A[2], ... A[n], that is, n no. of times.

Hence, the total no. of comparisons $T(n)$ in this case is given by

$$T(n) = n$$

Worst case complexity.

Case 3. The key is present at any location in the array

Let p_i be the probability that the key may be present at the ith-location, for any $1 \le i \le n$.

To reach the ith location, the algorithm executes $(i-1)$ comparisons in step 2.

Hence, for a successful search at ith place, we need $(i-1)+1 = i$ number of comparisons.

So the expected no. of comparisons is given by

$$T(n) = \sum_{i=1}^{n} p_i \cdot i \quad -(2)$$

Therefore, we can write

$$p_1 = p_2 = \cdots\cdots p_i = \cdots p_n = \frac{1}{n}$$

Thus, Eq. (2) reduces to

$$T(n) = \frac{1}{n}\sum_{i=1}^{n} i$$

$$= \frac{1}{n} \times \frac{n(n+1)}{2} = \frac{n+1}{2}$$

Hence the no. of key comparisons, when the key is present at any location

$$T(n) = \frac{n+1}{2}$$ (Average Case Behaviour) of the algo. SearchArray.

Summary of no. of comparison in the algorithm SearchArray

Case	Number of Key comparisons	Asymptotic Complexity	Remark
Case 1	$T(n)=1$	$T(n)=O(1)$	Best Case
Case 2	$T(n)=n$	$T(n)=O(n)$	Worst Case
Case 3	$T(n)=\frac{n+1}{2}$	$T(n)=O(n)$	Average case.

Linear Search with Ordered List

An algorithm for linear search method with ordered elements stored in an array A[1... n] is defined below.

Steps

```
1.  i=1
2.  flag=TRUE
3.  While (flag≠FALSE) && (K≥A[i]) do
4.      If (K=A[i]) then
5.          flag= FALSE
6.          Print 'Successful at' i
7.      ELSE
8.          i=i+1
9.          If (i>n) then
10.             Break
11.         EndIf
12.     EndIf
13. EndWhile
14. If (flag=TRUE) then
15.     Print 'Unsuccessful'
16. Endif
17. Stop.
```

Smrithi

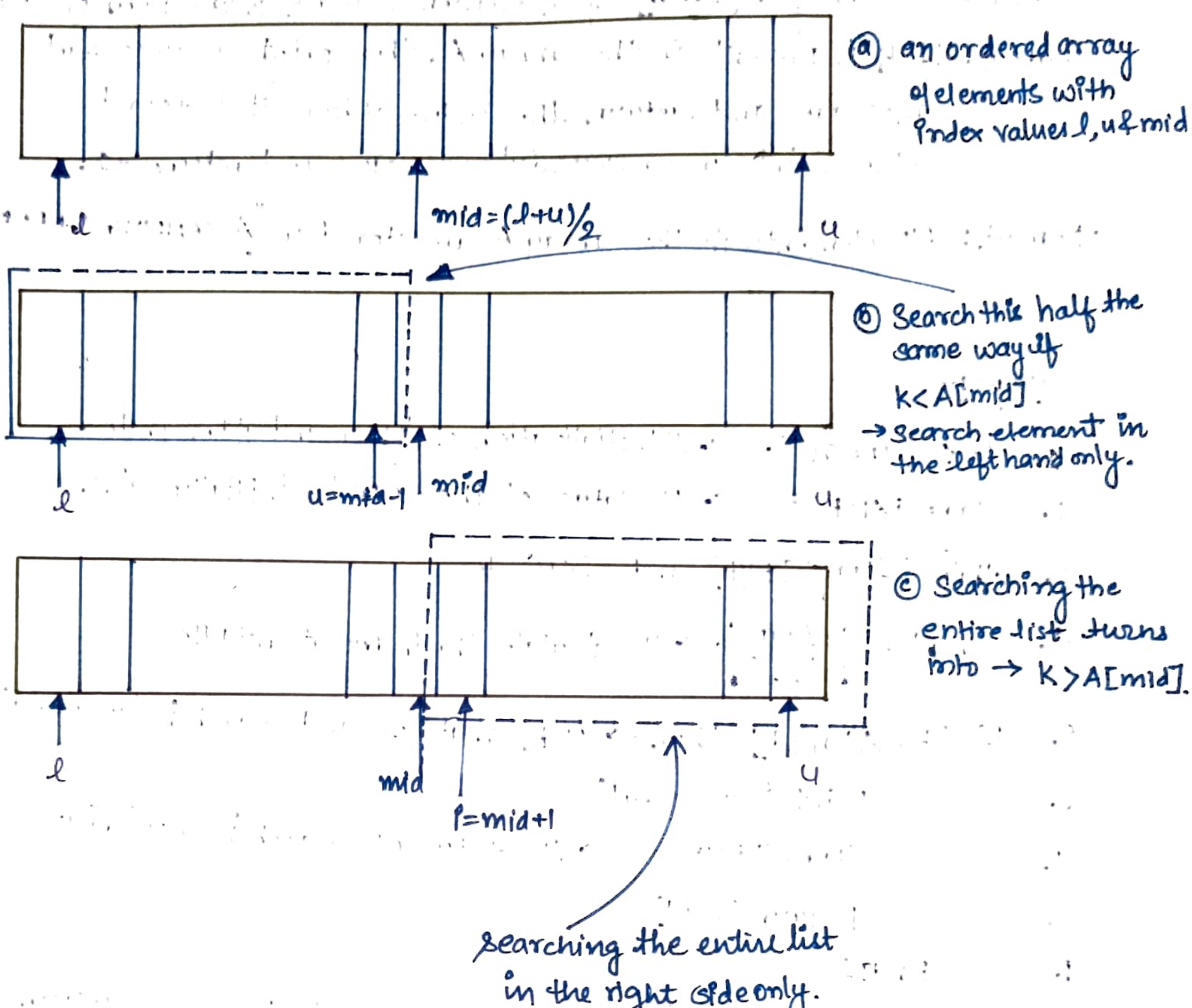
Analysis of the algorithm SearchArrayOrder

Same of Search Array, But for unsuccessful search, which gives the worst case behaviour of the algorithm Search Array, the Search Array Order has the same time complexity as the average case Behaviour.

Summary of number of comparisons in the algo. SearchArray order.			
Case	Number of key comparison	Asymptotic Complexity	Remark
Case 1.	$T(n)=1$	$T(n)=O(1)$	Best Case
Case 2.	$T(n)=\frac{n+1}{2}$	$T(n)=O(n)$	Average Case
Case 3.	$T(n)=\frac{n+1}{2}$	$T(n)=O(n)$	Worst Case

BINARY SEARCH TECHNIQUES

PRINCIPLE OF BINARY SEARCH METHOD



→ LINEAR SEARCH → not very efficient if the list is very large.

– It may be noted that BINARY SEARCH ALGORITHM is applicable only when the list is in "SORTED ORDER". Like a dictionary is in LEXICOGRAPHIC ORDER.

→ Suppose l and u are the lower and upper bound of the array. We want to search an item K in it. To do this we calculate a middle index with the values of l and u, let it be mid. Then we compare if K = A[mid] or not. If it is true → search complete. If not than if K < A[mid] ⟶ search in left half, otherwise in the right half ultimately you will find the value, if this process repeated again and again.

ALGORITHM BINARY SEARCH

- Input: An array $A[1...n]$ of n elements and K is the item of search.
- Output: If K is present in the array A, then print a successful message and return the index where it is found else print an unsuccessful message and return -1.
- Remark: All elements in the array A are stored in "ASCENDING ORDER".

Steps:-

1. $l = 1$, $u = n$ → // Initialization of lower and upper indexes.
2. flag = FALSE → // Status of the search, initially false.
3. While (flag ≠ TRUE) and ($l < u$) do
4. mid = $\lfloor \frac{l+u}{2} \rfloor$ → // Calculate the index at middle
5. If (K = A[mid]) then → // K matches and search is successful
6. Print 'Successful'
7. flag = TRUE → // Status of search is now True
8. Return (mid)
9. EndIf
10. If (K < A[mid]) then → // Let us check for possibility in left part.
11. u = mid − 1 → // This is the rightmost index of the left half.
12. Else → K > A[mid] and check the possibility at right part
13. l = mid + 1 → // This is the leftmost index of the right half.
14. EndIf
15. End While
16. If (flag = FALSE) then → // Search is failed.
17. Print 'unsuccessful'
18. Return (−1)
19. EndIf
20. Stop

TIME COMPLEXITY OF BINARY SEARCH ALGORITHMS

Binary Search is defined as a searching algorithm used in a sorted array by repeadedly dividing the search interval in half.

(1) Best Case Time Complexity

Best Case is when the element is at the middle index of the array. It takes only one comparison to find the target element. So the Best Case complexity is $O(1)$.

(2) Average case Time Complexity

Consider array arr[] of length N and element X to be found. There can be two cases:-

- ↳ Case 1: Element is present in the array.
- ↳ Case 2: Element is not present in the array.

There are N case1 and 1 case2. So total number of cases = N+1.

Now.

→ AN element at index $N/2$ can be found in 1 comparison.

→ AN elements at index $N/4$ and $3N/4$ can be found in 2 comparison.

→ AN elements at indices $N/8$, $3N/8$, $5N/8$ and $7N/8$ can be found in 3 comparisons and so on.

So, elements require

- 1 comparison = 1
- 2 comparison = 2
- 3 comparison = 4

⋮

- x comparison = 2^{x-1} where x belongs to the range $[1, \log N]$ because maximum comparisons = max. time N can be halved = maximum comparison to reach 1st element = $\log N$

So total comparisons

$= 1 * (\text{elements requiring 1 comparisons}) + 2 * (\text{elements requiring 2 comparisons}) + \cdots + \log N * (\text{elements requiring } \log N \text{ comparisons})$

$= 1*1 + 2*2 + 3*4 + \cdots + \log N * (2^{\log N - 1})$

$= 2^{\log N} * (\log N - 1) + 1$

$= N * (\log N - 1) + 1$

Total number of cases $= N+1$.

Therefore average complexity $= (N * (\log N - 1) + 1) / N+1$

$= N * \log N / (N+1) + \frac{1}{(N+1)}$

$\therefore$ here dominant term is $N * \log N / (N+1)$ which is approximately $\log N$. So the average case complexity is $O(\log N)$.

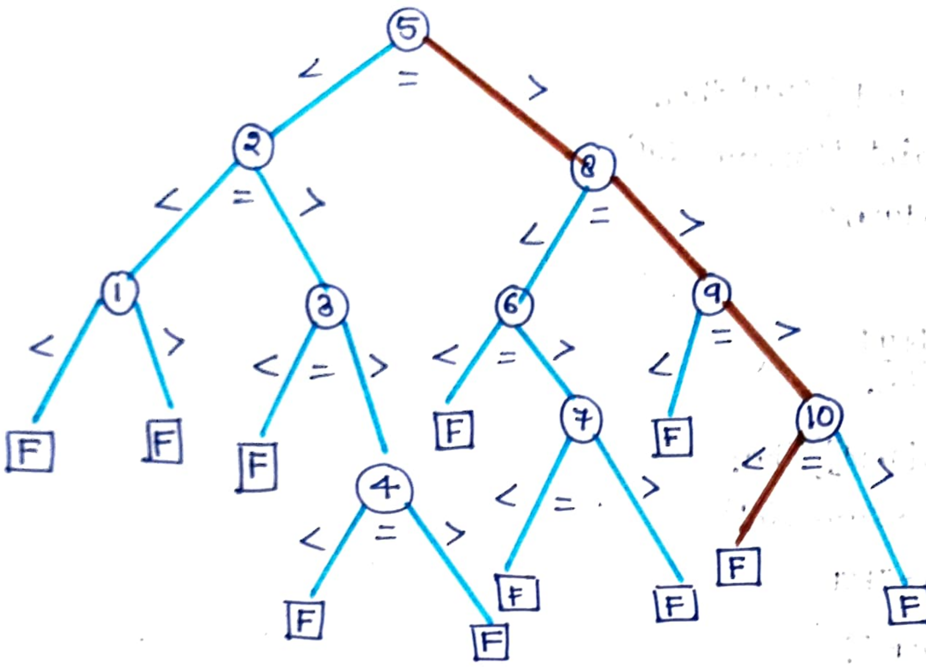
TIME COMPLEXITY FOR WORST CASE

$O(\log N)$.

AUXILLARY SPACE COMPLEXITY OF BINARY SEARCH ALGORITHM

Binary Search Algo. uses no extra space to search the element. Hence its auxiliary space complexity is $O(1)$.

Decision Tree with 10 element in Binary Search Method



Recursive Implementation of Binary Search

Steps

1. If (flag) and ($l > u$) then
2. print 'Unsuccessful'
3. Return 0
4. EndIf
5. mid = $\left\lfloor \frac{l+u}{2} \right\rfloor$
6. If (K = A[mid]) then
7. print 'Successful'
8. flag = TRUE
9. Return ()
10. Else
11. If (K < A[mid]) then
12. Binary Search (l, mid-1)
13. Else
14. Binary Search (mid+1, u)
15. EndIf
16. EndIf
17. Stop