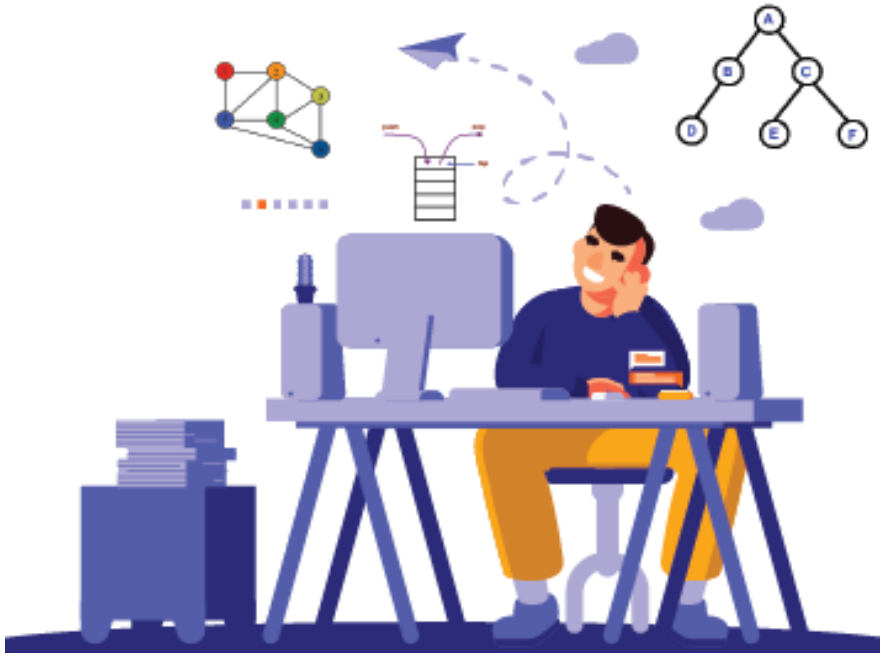
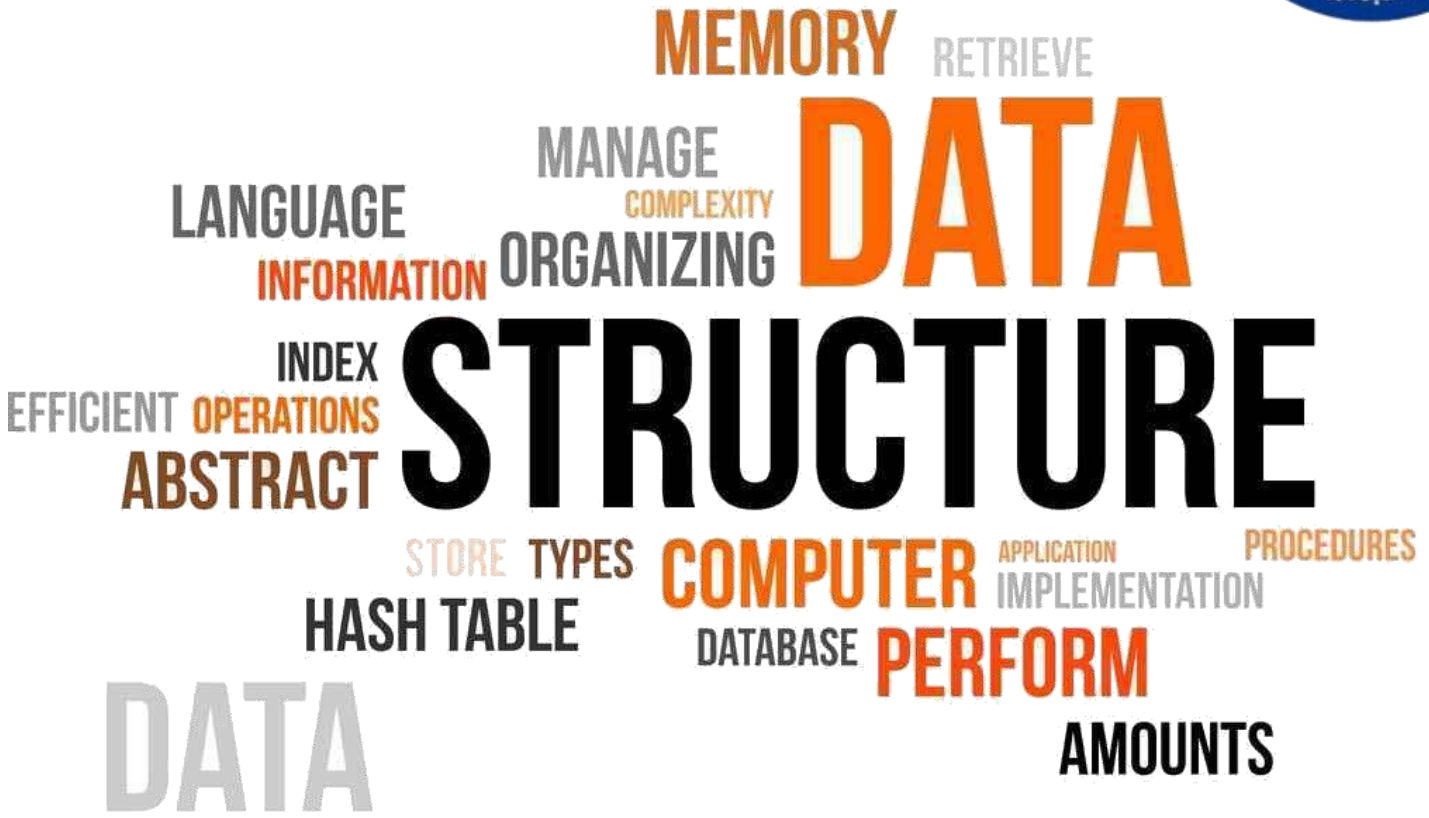


Data Structures

Course code: IT623



Dr. Rahul Mishra
Assistant Professor
DA-IICT, Gandhinagar

Array, Linear and Binary Search, and Matrices

Array

* We have studied linear and non-linear data structures.
* A data structure is said to be linear if its elements form a sequence, or, in other words, a linear list.
* Two basic ways to represent linear structures in memory

  a) One way is to have the linear relationship between the elements represented by means of sequential memory locations ⟶ arrays (This chapter)

  b) Another is to have the linear relationship between the elements represented by means of pointers or links. ⟶ Linked list (Next chapter)

The operations one normally performs on any linear structure {array or linked list}

(a) Traversal : Processing each element

(b) Search : Finding the location

(c) Insertion : Adding new

(d) Deletion : Removing one

(e) Sorting : Arranging the elements in some type of order

(f) Merging : Combining two lists into a single list.

⇨ The particular linear structure that one chooses for a given situation depends on the relative frequency with which one performs these different operations on the structure.

Sorting: Bubble Sort

- Let A be a list of n numbers
- Sorting A refers to the operation of rearranging the elements of (A) so they are in increasing order, i.e.;

$$A[1] < A[2] < A[3] \ldots A[N]$$

ex-

8, 4, 19, 2, 7, 13, 5, 16

↳

2, 4, 5, 7, 8, 13, 16, 19.

* Sorting may also mean arranging numerical data in decreasing order or arranging non-numerical data in alphabetical order.

Bubble Sort

→ Suppose the list of numbers A[1], A[2], ... A[N] is in memory.

→ The bubble sort algorithm works as follows:

step 1. : Compare A[1] and A[2] and arrange them in the desired order, so that A[1] < A[2]. Then compare A[2] and A[3] and arrange them so that A[2] < A[3]. Then compare A[3] and A[4] and arrange them so that A[3] < A[4]. Continue until we compare A[N-1] with A[N] and arrange them so that A[N-1] < A[N].

* Observe that step 1 involves n-1 comparisons.

➔ During step 1, the largest element is "bubbled up" to the nth position or "sinks" to the nth position.

Step:2 Repeat step 1 with one less comparison; that is, now we stop after we compare and possibly rearrange A[N-2] and A[N-1].

Step 2 involves N-2 comparisons and when step 2 is completed, the second largest element will occupy A[N-1].

Step:3 Repeat step 1 with two fewer comparisons

!
!

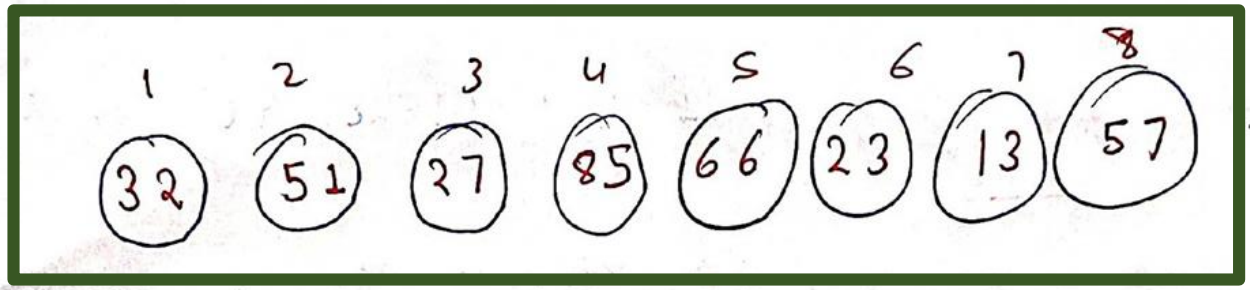
Step:4

!

Step: N-1 Compare A[1] with A[2] and arrange them so that A[1] < A[2].

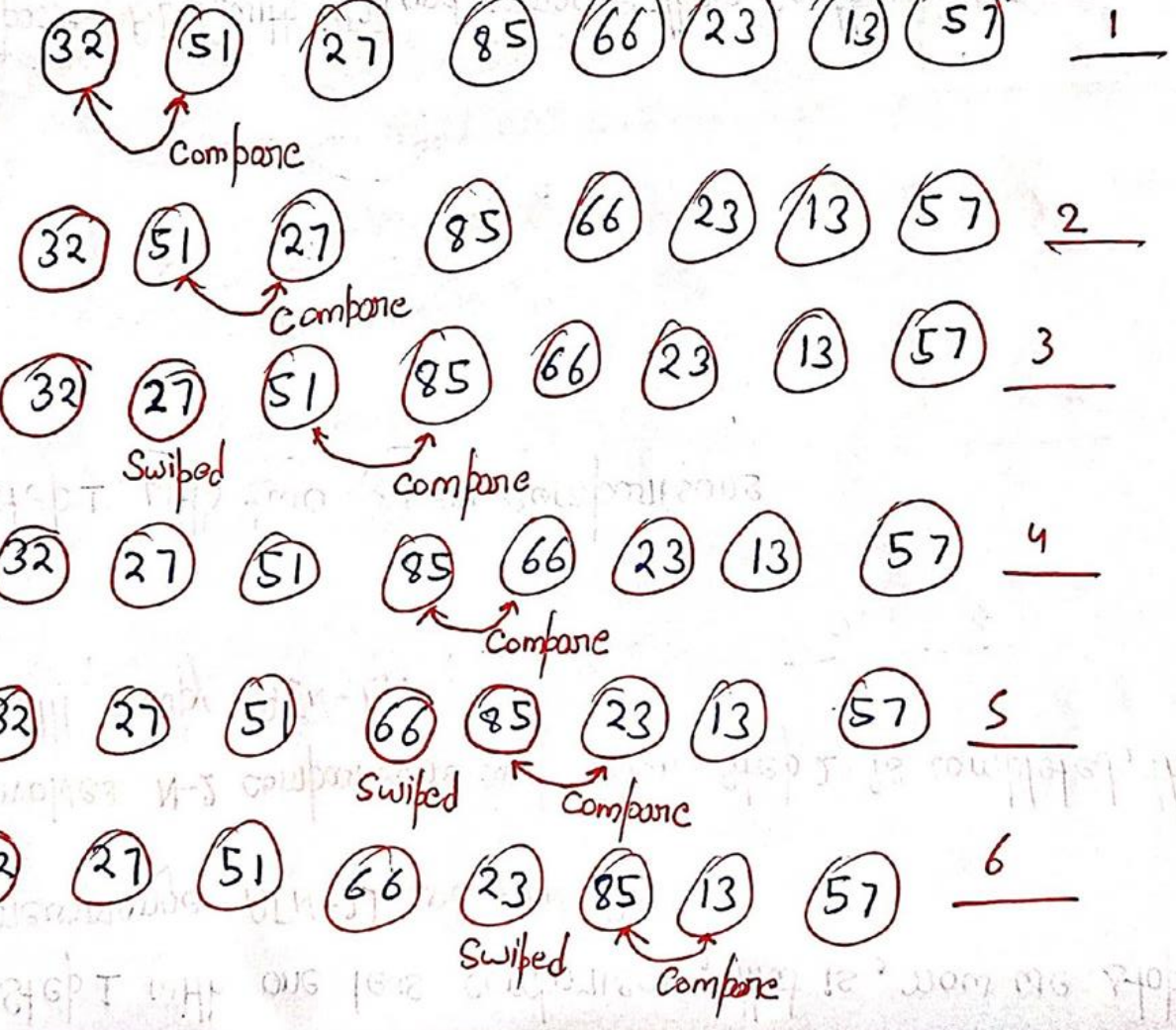
16

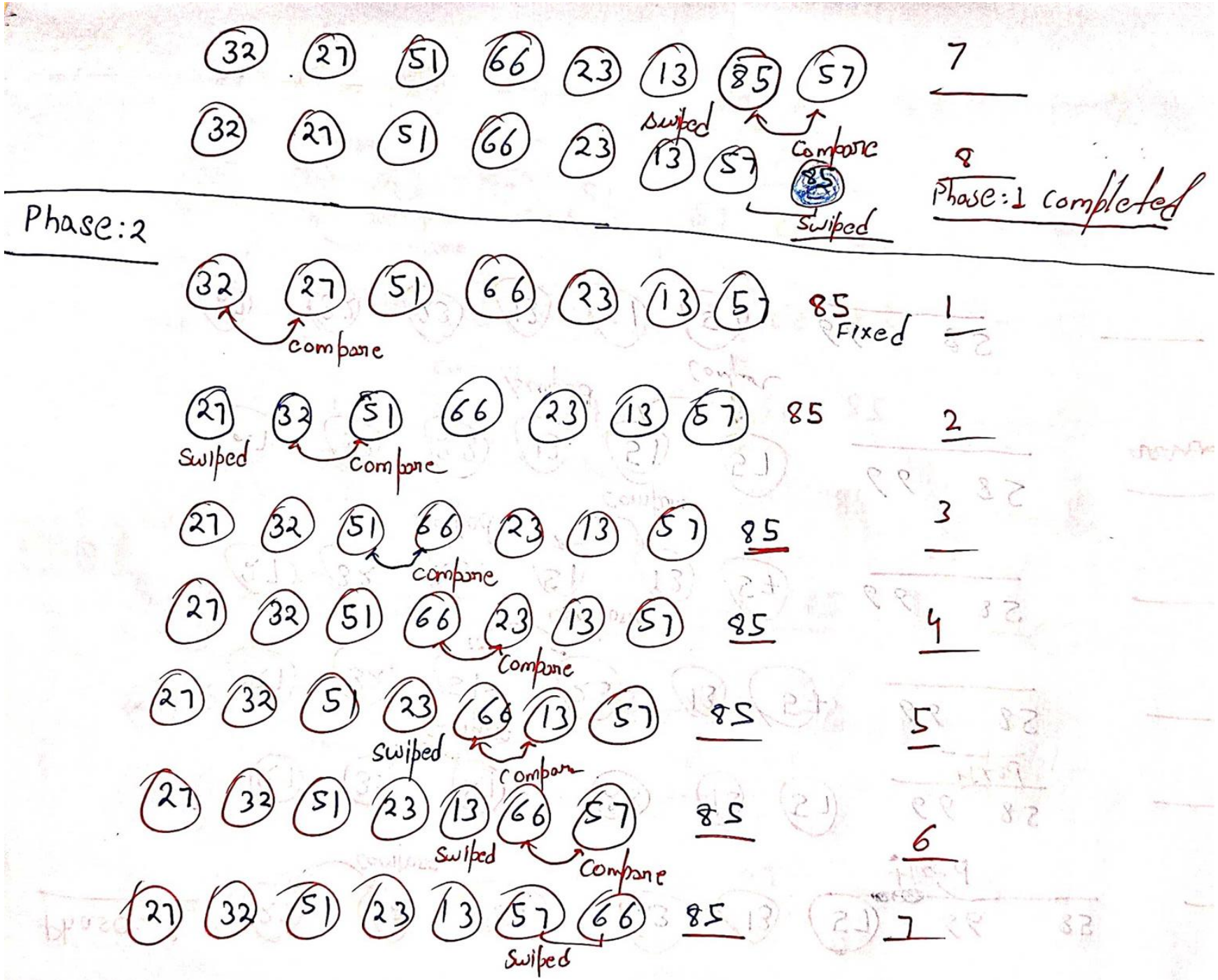
Example :

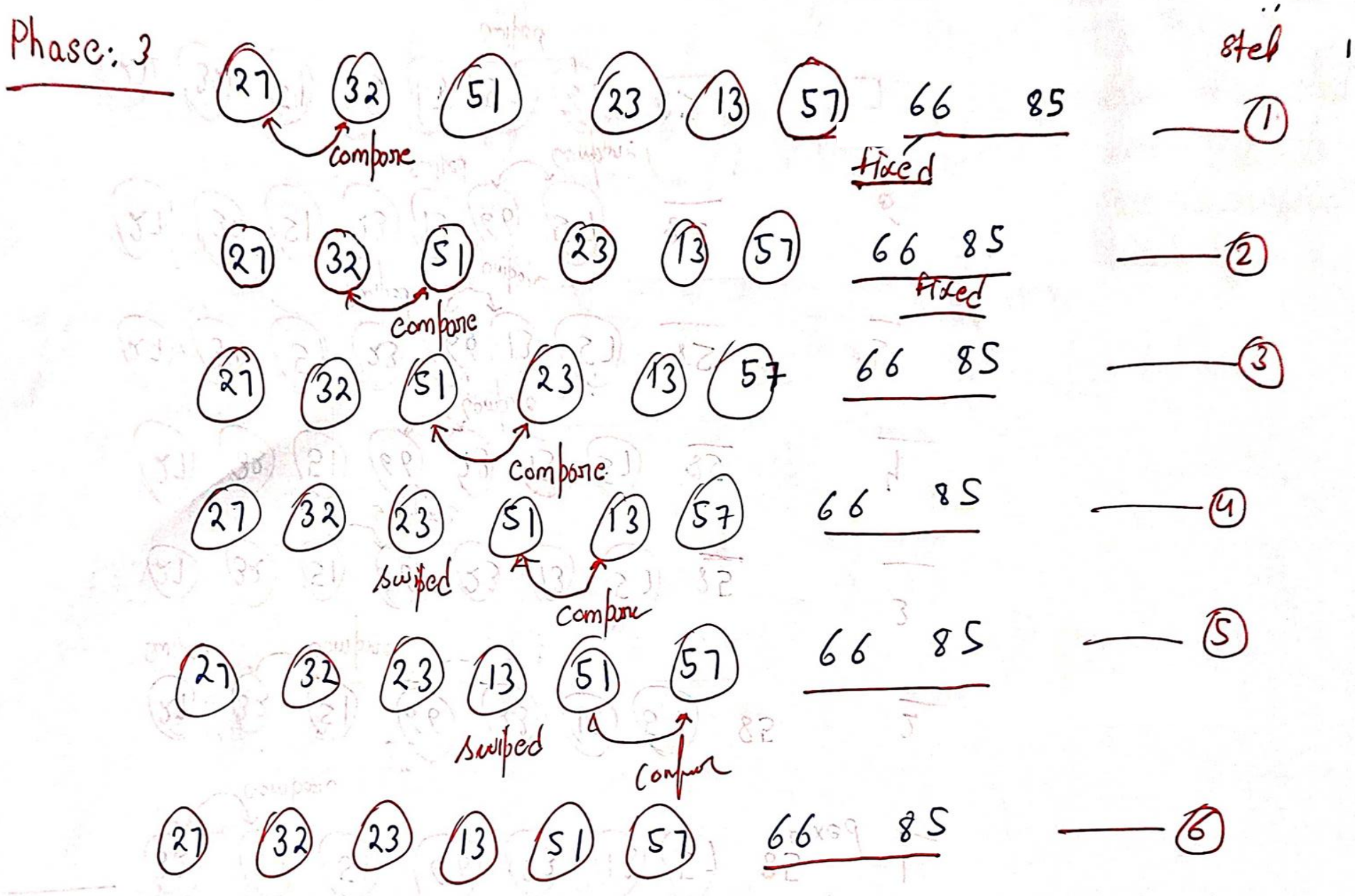


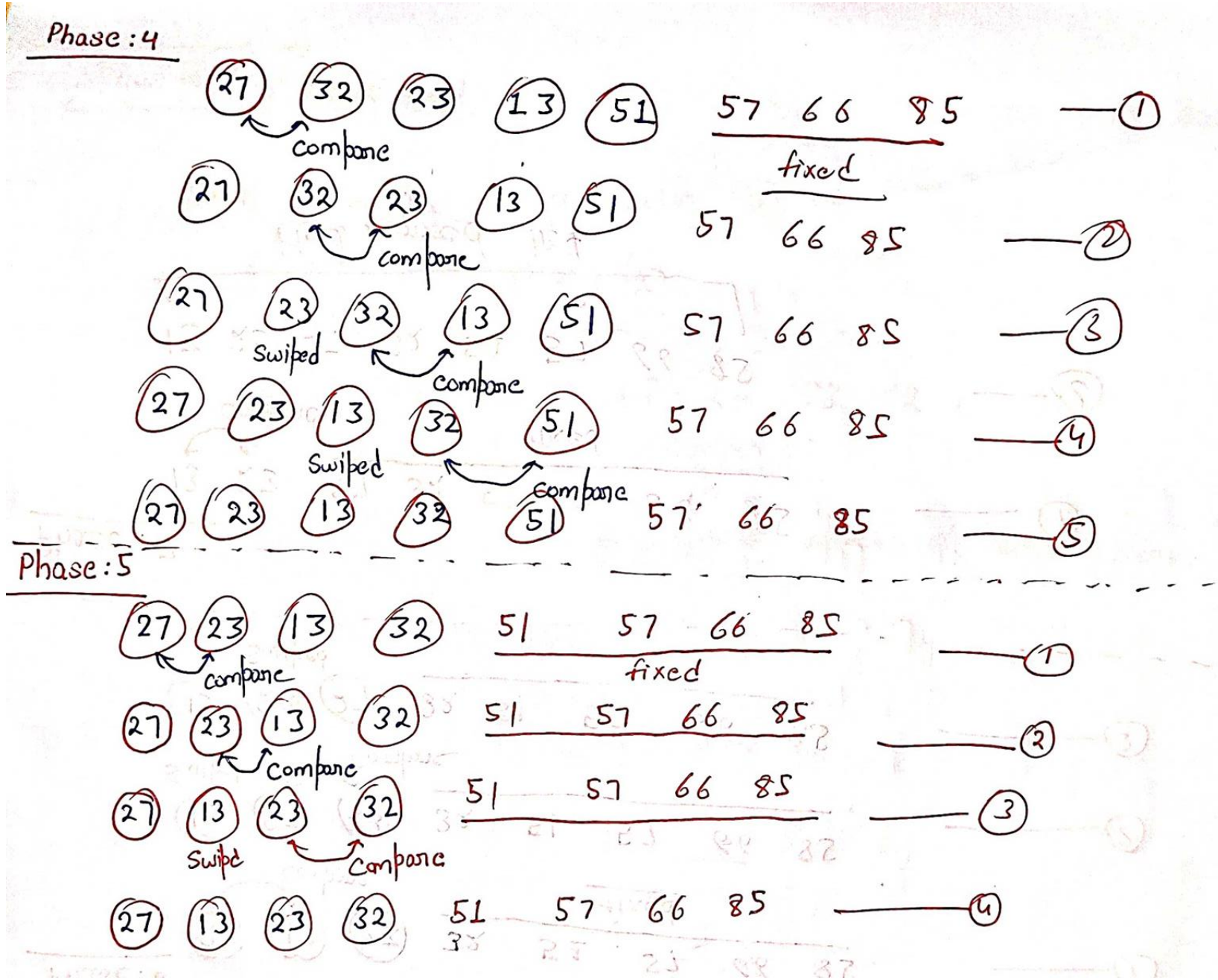
Given an unsorted array. Task is to sort the array using Bubble sort.

Phase:1

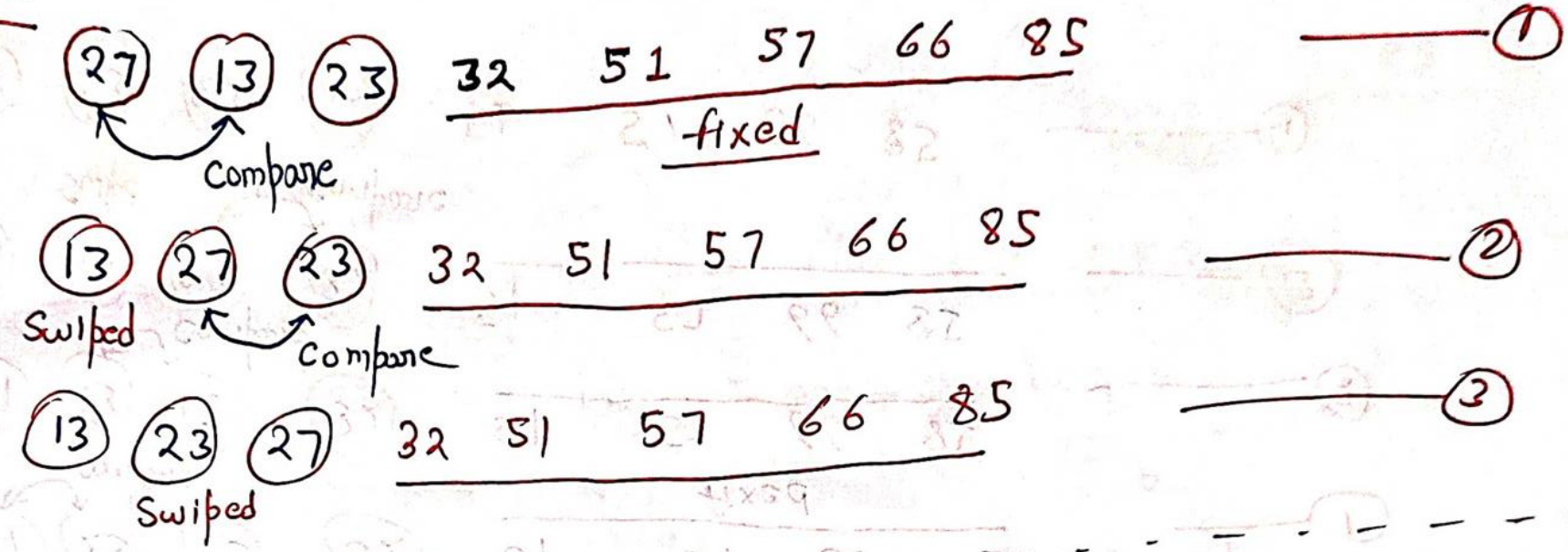



32 27 51 66 23 13 85 57
7
swiped
compare
32 27 51 66 23 13 57 85
8
swiped
Phase:1 completed
Phase:2
32 27 51 66 23 13 57 85
compare
Fixed
1
27 32 51 66 23 13 57 85
swiped
compare
2
27 32 51 66 23 13 57 85
compare
3
27 32 51 66 23 13 57 85
compare
4
27 32 51 23 66 13 57 85
swiped
compare
5
27 32 51 23 13 66 57 85
swiped
compare
6
27 32 51 23 13 57 66 85
swiped
7

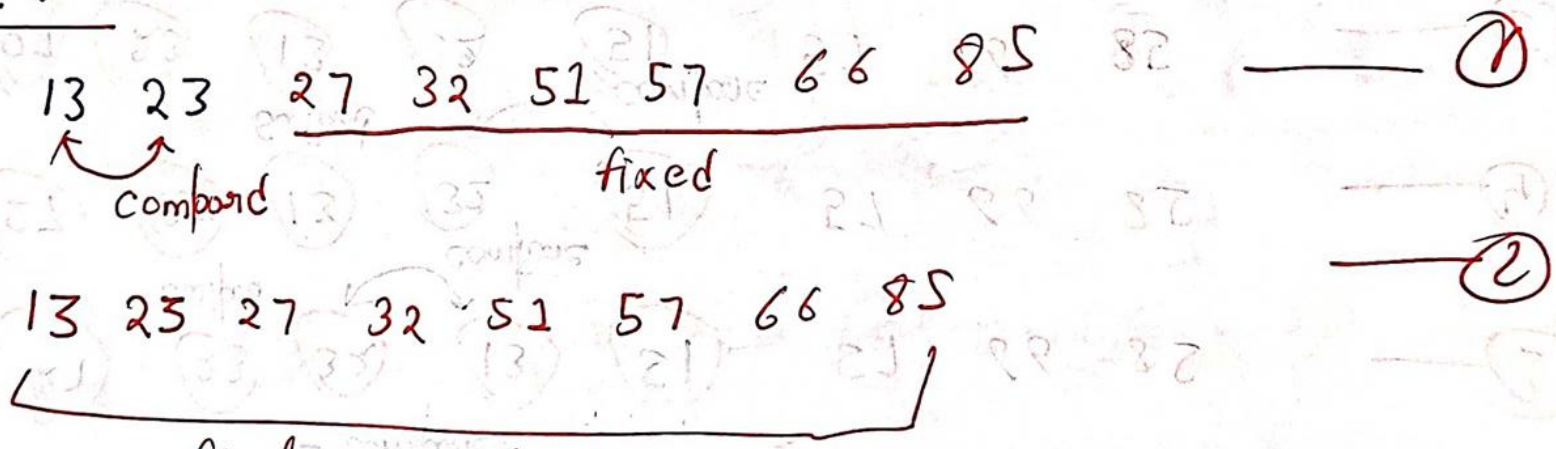

Phase: 3
Step
27 32 51 23 13 57 66 85
compare
Fixed
1
27 32 51 23 13 57 66 85
Fixed
compare
2
27 32 51 23 13 57 66 85
compare
3
27 32 23 51 13 57 66 85
swiped
compare
4
27 32 23 13 51 57 66 85
swiped
compare
5
27 32 23 13 51 57 66 85
6


Phase : 4
27 32 23 13 51 57 66 85 — 1
compare
fixed
27 32 23 13 51 57 66 85 — 2
compare
27 23 32 13 51 57 66 85 — 3
swiped
compare
27 23 13 32 51 57 66 85 — 4
swiped
compare
27 23 13 32 51 57 66 85 — 5
Phase : 5
27 23 13 32 51 57 66 85 — 1
compare
fixed
27 23 13 32 51 57 66 85 — 2
compare
27 13 23 32 51 57 66 85 — 3
swipd
compare
27 13 23 32 51 57 66 85 — 4

Phase : 6



Phase : 7



Algorithm : Bubble Sort

Solve following example using all the steps ?

12 minutes

Ques: 17 83 11 07 26 73 62 15 ?

* Determine number of swaps required?

Algorithm: (Bubble Sort) BUBBLE (DATA, N)

/* DATA is an array with N elements */
/* This algorithm sorts the elements in DATA. */

1. Repeat steps 2 and 3 for K = 1 to N-1
2. Set PTR = 1 [Initializes pass pointer PTR.]
3. Repeat while PTR ≤ N-K [Executes pass]

   (a) IF DATA[PTR] > DATA[PTR+1], then:
   Interchange DATA[PTR] and DATA[PTR+1] (swapping operation)
   [End of IF structure]

   (b) Set PTR = PTR + 1.

   [End of inner loop.]

[End of step 1 outer loop]

4. Exit.

Complexity of the Bubble Sort Algorithm:

* The time for a sorting algorithm is measured in terms of the number of comparisons.
* The number $f(n)$ of comparisons in the bubble sort is easily computed.
*
  - $n-1$ comparisons during the first pass, which places the largest element in the last position.
  - $n-2$ comparison in the second step. and so on ....

$$f(n) = (n-1) + (n-2) + \cdots + 2 + 1 = \frac{n(n-1)}{2} = \frac{n^2}{2} + O(n)$$

$$= O(n^2)$$

From formula — $1+2+3+\cdots+n = \frac{n(n+1)}{2}$

Searching: Linear Search

DATA ← a collection of data element in memory

ITEM ← specific item of information

* Searching refers to the operation of finding the location LOC of ITEM in DATA or even printing that some message that ITEM does not appear here.
* We can search particular item and can insert if not available.

(1) (2)

* (Searching and insertion algorithm)

1> Linear Search — Complexity $O(n)$

2> Binary Search — Complexity $\log_2(n)$

Linear Search

→ Given no information about DATA, the most intuitive way to search for a give ITEM in DATA is to compare ITEM with each element of DATA one by one.

→ Traversing DATA sequentially to locate ITEM, is called linear search or sequential search.

→ To simplify the matter, we first assign ITEM to DATA[N+1], the position following the last element of DATA. Then the outcome

$$LOC = N+1$$

where LOC denotes the location where ITEM first occurs in DATA, signifies the search is unsuccessful.

→ The purpose of this initial assignment is to avoid repeatedly testing whether or not we have reached the end of the array DATA.

Algorithm: (Linear Search) LINEAR (DATA, N, ITEM, LOC)

* DATA is a linear array with N elements, and ITEM is a given item of information.
* This algorithm finds the location LOC of ITEM in DATA, or sets LOC = 0 if the search is unsuccessful.

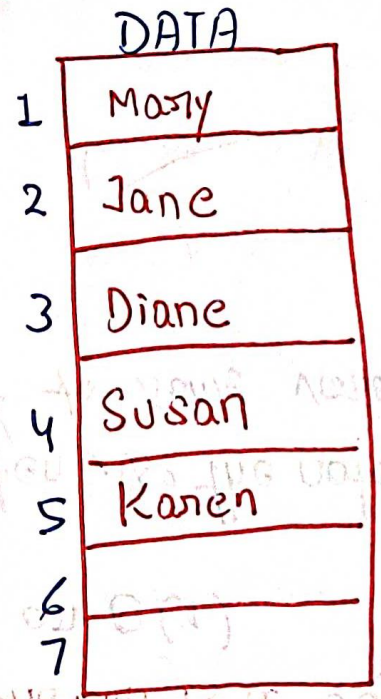
1. [Insert ITEM at the end of DATA] Set DATA[N+1] = ITEM
2. [Initialize counter] Set LOC = 1
3. [Search for ITEM]

   Repeat while DATA[LOC] ≠ ITEM

   Set LOC = LOC + 1

   [End of loop]
4. [Successful?] If LOC = N+1, then set LOC = 0
5. Exit.

Example :



Task : 1 Search Rahul ?

N+1 added.

Not found

Task : 2 Search susan ?

N+1

Found

* Complexity of the Linear Search Algorithm

⇒ The complexity of the search algorithm is measured by the number $f(n)$ of comparisons required to find ITEM in DATA, where DATA contain n elements.

⇒ Two important cases to consider are the average case and worst case.

* The worst case occurs when one must search through the entire array DATA, i.e; ITEM does not appear in DATA. Required comparisons are:

$$f(n) = n+1$$

* The complexity of average case is given by no of comparisons as:
(This was discussed earlier)

$$f(n) = 1 \cdot 1/n + 2 \cdot 1/n + \dots + n \cdot 1/n + (n+1) \cdot 0 = (1+2+\dots+n) \cdot 1/n$$

$$= n+1/2 \approx n/2$$ Average case is nearly half of (n).