"A person who never made a mistake never tried any thing New"
— Albert Einstein

2·1

ARRAYS RECORDS & POINTER

Data. structure are classified as either linear or non-linear.

- A data structure said to be linear if its elements form a sequence.

- Operation on linear data structure.

(a) Traversal - Processing each element in list.
(b) Search - Finding the location of element
(c) Insertion - Add new element into the list
(d) Deletion - Removing an element from list
(e). Sorting - Arrange the element in some order
(f) Merging - combing two list into a single list.

* LINEAR ARRAYS

Length = (UB − LB) + 1

Eg: An automobile company uses an array AUTO to record the number of automobiles sold each year from 1990 to 2000. Rather than beginning the index set with 1, it is more useful to begin the index set with 1990

Then LB = 1990 & UB = 2000

Length = (UB − LB) + 1
= (2000 − 1990) + 1
= 10 + 1 = 11

2000 − 1990 = 00φ0

1984 − 1930 = 2

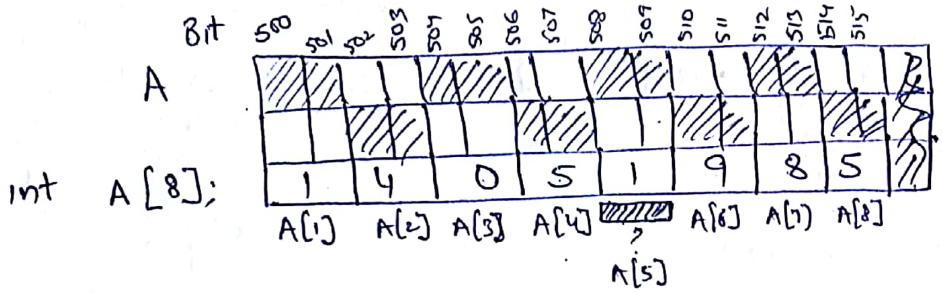
X	Y	Z	A	B
1	2	3	4	5

&= (5 − 1) + 1
= 4 + 1
= 5

Accordingly, the computer doesnt need to keep track of the address of every element of array A, but needs to keep Track only of the address of first element. denoted as Base (A) & called the base address of A

- Calculation of addresses of any element of A

$$LOC(A[K]) = Base(A) + w(K - LB)$$



$$Loc(A[5]) = Base(A) + 2(5-1)$$

$$Loc(A[5]) = 500 + 2(4) = 500 + 8 = \underline{508}$$

LOC → Memory location

Base → Starting memory location of Array

w → size of iteom in an array

K → Index of Kth element in an array

LB → Lower Bound of an array.

TRAVERSING LINEAR ARRAYS

Algorithm : TRAVERSING

Here LA is a linear array with lower bound LB & upper bound UB. This algorithm traverses LA applying an operation PROCESS to each element of LA

```
1. [Initialize counter]   Set K := LB
2. Repeat step 3 & 4 while K ≤ UB
3. [Visit element]   Apply PROCESS to LA[K]
4. [Increase counter]   Set K := K+1
[End of step 2 loop]
5. EXIT
```

OR

```
1. Repeat for K = LB to UB
        Apply PROCESS to LA[K]
   [End of loop]
2. EXIT
```

Algorithm: INSERT (A, N, K, ITEM)

Here A is linear array with N elements and K is positive integer s.t. K ≤ N. This algorithm insert ITEM into the Kth position in array A.

```
1. Set J:=N                        [# Initilize counter]
2. Repeat step 3 & 4 while J ≥ K
           step A[J+1] := A[J]     <Move Jth element downwards>
           Set J:= J-1             <Decrese counter>
   [END of step 2 loop]
3.   Set A[K] := ITEM              <insert element>
4.   Set N := N+1                  < Reset N>
5.   EXIT
```

:0:

Algorithm: DELETE (A, N, K, ITEM)

Here A is a linear array with N elements and K is a positive integer s.t. K ≤ N. This algo delete Kth iteam from A

```
1.  Set ITEM := A[K]
2.  Repeat for J=K to N-1
          Set LA[J] := LA[J+1]
      [END of Step 2 loop]
3.  Set N:=N-1          <Reset N in array A>
4.  EXIT
```

:0:

SORTING : BUBBLE SORT

Sorting array A refers to the operation of rearranging the elements of A so they are in increasing order.

PASS 1st	8	4	19	2	7	13	5	16
	4	8	19	2	7	13	5	16
	4	8	19	2	7	13	5	16
	4	8	2	19	7			
			2	7	19	13		
			2	7	13	19	5	16
	4	8	2	7	13	5	19	16
	4	8	2	7	13	5	16	19 ✓
PASS 2nd	4	8	2	7	13	5	16	19
	4	2	8	7				
		2	7	8	13	5	16	19
	4	2	7	8	13	5	16	19
	4	2	7	8	5	13	16	19
PASS 3rd	4	2	7	8				
	2	4	7	8	5	13	16	19
	2	4	7	5	8	13	16	19
PASS 4th	2	4	7	5	8	13	16	19
	2	4	5	7	8	13	16	19

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

Here DATA is an arrary with N elements

```
1. Repeat step 2 and 3 for K=1 to N-1
2.        Set PTR := 1          [Initialize pass pointer PTR]
3.    Repeat while PTR ≤ N-K          [Execute pass]
          (a) If DATA [PTR] > DATA [PTR +1] , then
      Interchange DATA [PTR] and DATA [PTR +1]
          [END of if structure]
          (b) Set PTR := PTR+1
          [END of inner loop (while)]
          [END of step 1 outer loop (for)]
4.  EXIT
```

Notes.

PassI — Step 1 involves n-1 comparison & A[N] will contain the largest element.

PassII — After step 2 2nd largest element will be A[N-1] & no of comparison are (N-2)

After n-1 passes (steps) the list will be sorted in increasing order.

Complexity of Bubble Sort Algo

$$f(n) = (n-1) + (n-2) + \ldots\ldots + 2 + 1$$

$$= \frac{n(n-1)}{2} = \frac{n^2-n}{2} = \frac{n^2}{2} \cancel{+} -\frac{n}{2}$$

$$= \frac{n^2}{2} + O(n) = \cancel{n}\, O(n^2) + O(n) = O(n^2)$$

o Sathi Ra

2-7

Searching: LINEAR SEARCH

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

Here DATA is a linear array with N elements, and ITEM is given item of information. This algorithm finds the LOC := 0 if search is uncessful.

```
1.  Set DATA[N+1] := ITEM
2.  Set LOC := 1
3.  Repeat while DATA[LOC] ≠ ITEM.
        Set LOC := LOC + 1
    [End of loop]
4.  If LOC = N+1, then
        set LOC := 0
5.  EXIT
```

—:o: ———————— :o: ——— :o: ——— :o: ———

→ Complexity of Linear search

suppose p_i is probability of each position to be the item & the q is probability of not occurance of iteam

~~f(n)=~~ $(p_1 + p_2 + p_3 \cdots p_n + q) = 1$

suppose element is present in our array then

$$f(n) = 1\,p_1 + 2\,p_2 + \ldots n\,p_n$$

$$= 1\frac{1}{n} + 2\frac{1}{n} + \ldots n\frac{1}{n}$$ (suppose element occur at equal probability)

$$= \frac{1}{n}(1+2+\ldots n) = \frac{1}{\cancel{n}}\left(\frac{\cancel{n}\cdot(n-1)}{2}\right) = \frac{n-1}{2} = O(n)$$

Hense the complexity of Linear search algrithm is $O(n)$

BINARY SEARCH

Suppose DATA is an array which is sorted in increasing numerical order or equivalent alphabetically. Then there is an extremely efficient searching algorithm, called Binary search

11.	22	30	$\boxed{33}$	40	$\boxed{44}$.	55	60	70
BEG			MID					END

$\frac{9}{2} = 4.5$

ITEM = 55 ; to be search.

is MID = 55   X   33 ≠ 55

— set BEG := MID+1   = 40

$\frac{5}{2}$ = ~~3~~2·5

MID = 44   is   44 ≠ 55

— set BEG = MID +1

$\frac{4}{2} = 2$

55 = 55 ✓

: o :   : o :

Algorithm: (Binary search) BINARY (DATA, LB, UB, ITEM, LOC)

Here DATA is a sorted array with LB & UB. & ITEM is to search. The variable BEG, END & MID denote begning, end, middle location of of segment of DATA. This algo find the LOC of ITEM in DATA or ~~elements of D~~ set LOC = NULL

```
1. Set BEG:= LB , END:= UB $ MID = INT ((BEG+END)/2)
2. Repeat step 3 & 4 while BEG ≤ END & DATA[MID] ≠ ITEM
3.     If ITEM < DATA[MID], then
               Set END := MID - 1
       Else
               set BEG := MID+1
       [End of If structure]
4.     Set MID := INT ((BEG+END)/2)
   [END OF step 2 loop]
5. If DATA[MID] = ITEM , then
           Set LOC:= MID
   Else
           set LOC := NULL
   [END OF If structure]
6. EXIT
```

Complexity of Binary Search

We require at most f(n) comparisons to locate ITEM where

$2^{f(n)} > n$ or equivalently $f(n) = \lfloor \log_2 n \rfloor + 1$

Thus the running time for the worst case is approximately equal to $\log_2 n$. complexity $\Rightarrow O(\log_2 n)$

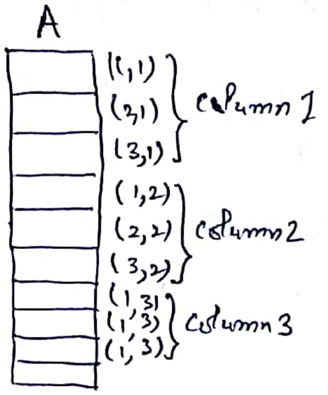
$2^{10} = 1024 > 1000$ &Range 2

+ very efficient

- The list must be sorted
- One must have direct access to middle element in sublist.

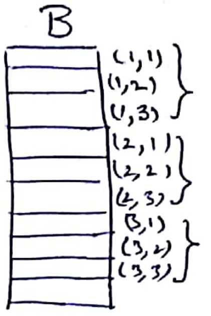
→ Keeping data in a sorted array is normally very expensive. when inserting & deletions (In such situations, one may use different data structure i.e. link list or binary search tree)

2 D ARRAY



column-major order

* storing array A column by-column

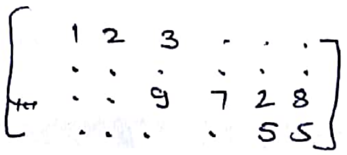


Row-major order

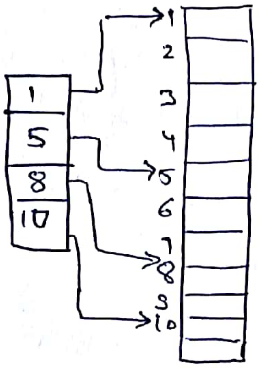
* storing array B row-by-row

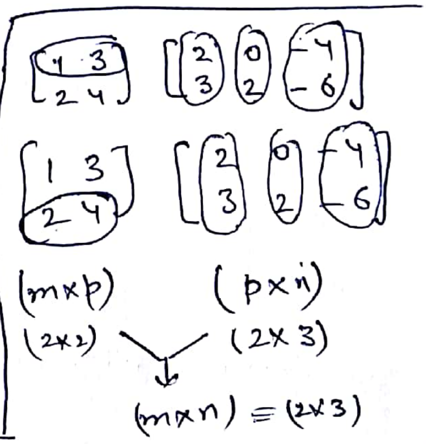
POINTER

Array whose rows - or column - being with different numbers of data elements & end with unused storage locations are said to be jagged.



A variable P is called a pointer if P "points" to an element in DATA. Analogously, an array PTR is called a pointer array if each element of PTR is a pointer.





Algorithm : MATMUL (A, B, C, M, P, N)

Let A be an M×P matrix array & B be P×N matrix array. This also stores the product of A & B in an M×N matrix array C.

```
1. Repeat step 2 to 4 for I = 1 to M
2.   Repeat step 3 & 4 for. J = 1 to N
3.       set C[I, J] := 0
4.   Repeat for K = 1 to P          // Repeat for Row of first matrix
                                    or  "  "  cloumn of 2nd  "
             C[I, J] := C[I, J] + A[I, K] * B[K, J]
     [End of inner loop of step 4]
     [End of step 2 (middle) loop]
     [End of step 1 (outer loop]
5. EXIT
```

A[1,1]	B[1,1]	X
A[1,2]	B[2,1]	
A[1,3]	B[3,1]	X

The complexity of above Algorithm is

$$C = m.n.p$$

SPARSE MATRICES

Matrices with a relative high proportion of zero entries are called <u>sparce matrices</u>.

The matrix, where all the entries above main diagonal are zero or equivalently, where non-zero entries can only occur on or below the main diagonal, is called a (lower) triangular matrix.

$$\begin{bmatrix}1&&&\\2&3&&\\3&4&6&\\4&5&7&8\end{bmatrix}$$

Tri-angular matrix

$$\left[\begin{matrix}\cancel{1}\\\cancel{2}\\\cancel{3}\\\cancel{4}\\\cancel{5}\end{matrix}\right.\begin{bmatrix}1&2&&&\\1&2&3&&\\&4&5&6&\\&&7&8&9\\&&&6&7\end{bmatrix}$$

Tri-diagonal matrix