June 26, 2025

Unit 2: Arrays

* Array is a finite ordered collection of homogeneous data elements which provide random access to the element. Array element share a common name.

for e.g.,

```
     datatype
        ⊥
       int  a[5] ← size of array
            T
         name of array
```

a	10	20	30	40	50
	0	1	2	3	4

Application of array

1. Array is used to store list of values.
2. Array is used to perform matrix operation.
3. Array is used to implement search algorithm.
4. Array is used to implement sort algo.
5. Array is used to implement DS.
6. Array is used to implement CPU scheduling algo.

* Types of array

Array are classified into two type

- → One dimentional array
- → Multidimentional array → Two dimentional array
  - → n-dimentional array

27/June/2025

* Single dimentional or one dimentional array

→ The simpler form of an array is one dimentional array, the array is given a name an its element are reffered by thier subscript or indices.

Imp 2M Q) Calculate the address of any element of an array by using formula:

address (ith) element = Base address + (offset of ith element from base address)

address of ith element = Base address + offset of ith element from base add.

offset of ith element = (no. of element before ith element) * (size of that element)

Example : If int a[5] is an array of five elements each integer is of 2 bytes and base address is 100 then find the address of third and fifth element.

for 3rd = 100 + (2×2) = 104 } add. (3rd element) = 100 + (2×2)

for 5th = 100 + (2×4) = 108. } add. (5th element) = 100 + (2×4)

Two dimentional array

↳ Two dimentional array is a collection of elements placed in m Rows and n Columns. In two dimentional array elements are stored either column by column i.e., column major representation, or row by row i.e., row major representation.

① Row major representation

* To calculate address in 2D array using formula:

arr[i][j] = base address + [(i * no. of columns + j) * size of element (data type)]

Q To calculate address of ~~no. 2×3~~ num[2][3] of integer array of size 3×4 with base address 1000

$\Rightarrow$ arr[2][3] = 1000 + (2×4+3)×2
= 1000 + 22
$\boxed{=1022}$

0th				1st				2nd			
1	2	3	4	5	6	7	8	9	10	11	12
1000	02	04	06	08	10	12	14	16	18	20	22

Q To calculate address of A(1,2) of integer array of size 2×3 with base address 100.

$\Rightarrow$ arr[1][2] = 100 + (1×3+2)×2
= 100+10
= 110

② Column major representation

* To calculate address in 2D array using formula
arr[i][j] = base adress + [(j × no. of rows + i)] × [(size of element (datatype)].

Q ~~Integer~~ int A[2][3] = {1,2,3,4,5,6}, assuming A is stored in column major order with 1st element of A is at address 1000 and each integer occupying 2 bytes. What would be the address of element A[1][2]

$\Rightarrow$ arr[1][2] = 1000 + (2×2+1)×2
= 1000+10
= 1010

2m

Sparse Matrix.

↳ It is a 2-D data object with 'm' rows & 'n' columns, therefore having total m×n elements in which most of the elements have value 'zero'. Sparse matrix contains lesser non-zero element than 'zero', so, instead of stooring zeros with non-zero element, we only store non-zero element. This means stooring non-zero element with triples (row, column, v where,

row: row index - non-zero elements
column: column index of non-zero elements
value: value of the non-zero element present of at index (row & column).

for e.g.,

$$\begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 0 & 7 & 0 & 0 \\ 1 & 0 & 0 & 0 & 4 \\ 2 & 3 & 0 & 0 & 6 \\ 3 & 0 & 0 & 0 & 0 \end{array}$$

Row	0	1	2	2
Column	1	3	0	3
values	7	4	3	6

Use of sparse matrix:

① Stoarage: There are lesser non-zero elements than zeros and becoz of this lesser memory can used to store only those element

② Computing time: Computing time can be saved by logically designed DS traversing only. A DS traversing only non-zero element.

Imp

* Sorting Concept

↳ Sorting is a technique to re-arrange the element in ascending or descending order which can be numerical, graphical or user defined order.

Sorting algos can be divided:

① Internal sort: This method used only the primary memory during sorting process. All data items are held in main memory and no secondary memory is used for sorting process.
for e.g. Bubble sort, insertion sort, quick sort.

② External sort: Sorting large amt. of data requires external or secondary memory. This process uses external memory such as HDD, floppy disk, magnetic tape.
for e.g., Merge sort.

* Bubble Sort

↳ In this sorting method, the list is divided into two sublist, sorted & unsorted. The smallest element is bubbled (shift) from unsorted sublist after moving the smallest element, the imaginary wall, moves one element ahead.

Algorithm: Arranging element in ascending order

```
Step 1: Read N (no. of element)
Step 2: Read array A[0], A[1] .. A[n-1].
```

July 02, 20

~~Step3: for (i=0; i<n;~~

```
Step3: for i=0 to n-1 do
           for j=0 to n-i-1 do
             if (A[j] > A[j+1]) then
                 temp = A[j]
                 A[j] = A[j+1]
                 A[j+1] = temp

Step 4: stop
```

Q Consider the array A containing following elements to be sorted using bubble sort.
A = {13, 11, 14, 15, 19, 9}

Pass1: $\underleftrightarrow{13 \quad 11} \quad 14 \quad 15 \quad 19 \quad 9$

$11 \quad \underleftrightarrow{13 \quad 14} \quad 15 \quad 19 \quad 9$

$11 \quad 13 \quad \underleftrightarrow{14 \quad 15} \quad 19 \quad 9$

$11 \quad 13 \quad 14 \quad \underleftrightarrow{15 \quad 19} \quad 9$

$11 \quad 13 \quad 14 \quad 15 \quad \underleftrightarrow{19 \quad 9}$

$11 \quad 13 \quad 14 \quad 15 \quad 9 \quad \boxed{19}$

Pass2: $\underleftrightarrow{11 \quad 13} \quad 14 \quad 15 \quad 9 \quad \boxed{19}$

$11 \quad \underleftrightarrow{13 \quad 14} \quad 15 \quad 9 \quad \boxed{19}$

$11 \quad 13 \quad \underleftrightarrow{14 \quad 15} \quad 9 \quad \boxed{19}$

$11 \quad 13 \quad 14 \quad \underleftrightarrow{15 \quad 9} \quad \boxed{19}$

$11 \quad 13 \quad 14 \quad 9 \quad \boxed{15} \quad \boxed{19}$

Pass 3: $\underleftrightarrow{11 \quad 13} \quad 14 \quad 9 \quad \boxed{15} \quad \boxed{19}$

$11 \quad \underleftrightarrow{13 \quad 14} \quad 9 \quad \boxed{15} \quad \boxed{19}$

$11 \quad 13 \quad \underleftrightarrow{14 \quad 9} \quad \boxed{15} \quad \boxed{19}$

$11 \quad 13 \quad 9 \quad \boxed{14} \quad \boxed{15} \quad \boxed{19}$

Pass 4: $\underleftrightarrow{11 \quad 13} \quad 9 \quad \boxed{14} \quad \boxed{15} \quad \boxed{19}$

$11 \quad \underleftrightarrow{13 \quad 9} \quad \boxed{14} \quad \boxed{15} \quad \boxed{19}$

$11 \quad 9 \quad \boxed{13} \quad \boxed{14} \quad \boxed{15} \quad \boxed{19}$

Pass 5: $\underleftrightarrow{11 \quad 9} \quad \boxed{13} \quad \boxed{14} \quad \boxed{15} \quad \boxed{19}$

$\boxed{9 \quad 11 \quad 13 \quad 14 \quad 15 \quad 19}$

Q Sort the following nubs using bubble sort method

{108, 3, 97, 65, 71, 23, 57, 93, 100}

Pass 1 : 108 3 97 65 71 23 57 93 100

3 108 97 65 71 23 57 93 100

3 97 108 65 71 23 57 93 100

3 97 65 108 71 23 57 93 100

3 97 65 71 108 23 57 93 100

3 97 65 71 23 108 57 93 100

3 97 65 71 23 57 108 93 100

3 97 65 71 23 57 93 108 100

3 97 65 71 23 57 93 100 $\boxed{108}$

Pass 2 : 3 97 65 71 23 57 93 100 $\boxed{108}$

3 97 65 71 23 57 93 100 $\boxed{108}$

3 65 97 71 23 57 93 $\boxed{100}$ $\boxed{108}$

3 65 71 97 23 57 93 $\boxed{100}$ $\boxed{108}$

3 65 71 23 97 57 93 $\boxed{100}$ $\boxed{108}$

3 65 71 23 57 97 93 $\boxed{100}$ $\boxed{108}$

3 65 71 23 57 93 97 $\boxed{100}$ $\boxed{108}$

3 65 71 23 57 93 $\boxed{97}$ $\boxed{100}$ $\boxed{108}$

Pass 3 : 3 65 71 23 57 93 $\boxed{97}$ $\boxed{100}$ $\boxed{108}$

3 65 71 23 57 93 $\boxed{97}$ $\boxed{100}$ $\boxed{108}$

3 65 71 23 57 $\boxed{93}$ $\boxed{97}$ $\boxed{100}$ $\boxed{108}$

3 65 23 71 57 $\boxed{93}$ $\boxed{97}$ $\boxed{100}$ $\boxed{108}$

3 65 23 57 71 $\boxed{93}$ $\boxed{97}$ $\boxed{100}$ $\boxed{108}$

3 65 23 57 71 $\boxed{93}$ $\boxed{97}$ $\boxed{100}$ $\boxed{108}$

Pass 4 : 3 65 23 57 71 $\boxed{93}$ $\boxed{97}$ $\boxed{100}$ $\boxed{108}$

3 65 23 57 71 $\boxed{93}$ $\boxed{97}$ $\boxed{100}$ $\boxed{108}$

3 23 65 57 71 $\boxed{93}$ $\boxed{97}$ $\boxed{100}$ $\boxed{108}$

3 23 57 65 71 $\boxed{93}$ $\boxed{97}$ $\boxed{100}$ $\boxed{108}$

3 23 57 65 71 $\boxed{93}$ $\boxed{97}$ $\boxed{100}$ $\boxed{108}$

Pass 5 : 3 23 57 65 71 $\boxed{93}$

$3 \quad \underleftrightarrow{23 \quad 57} \quad 65 \quad 71 \quad \boxed{93} \quad \boxed{97} \quad \boxed{100} \quad \boxed{108}$

$3 \quad 23 \quad \underleftrightarrow{57 \quad 65} \quad 71 \quad \boxed{93} \quad \boxed{97} \quad \boxed{100} \quad \boxed{108}$

$3 \quad 23 \quad 57 \quad \underleftrightarrow{65 \quad 71} \quad \boxed{93} \quad \boxed{97} \quad \boxed{100} \quad \boxed{108}$

$3 \quad 23 \quad 57 \quad 65 \quad \boxed{71} \quad \boxed{93} \quad \boxed{97} \quad \boxed{100} \quad \boxed{108}$

Pass 6: $\underleftrightarrow{3 \quad 23} \quad 57 \quad 65 \quad 71 \quad 93 \quad 97 \quad 100 \quad 108$

$3 \quad \underleftrightarrow{23 \quad 57} \quad 65 \quad 71 \quad 93 \quad 97 \quad 100 \quad 108$

$3 \quad 23 \quad \underleftrightarrow{57 \quad 65} \quad 71 \quad 93 \quad 97 \quad 100 \quad 108$

$3 \quad 23 \quad 57 \quad \boxed{65} \quad \boxed{71} \quad \boxed{93} \quad \boxed{97} \quad \boxed{100} \quad \boxed{108}$

Pass 7: $\underleftrightarrow{3 \quad 23} \quad 57 \quad 65 \quad 71 \quad 93 \quad 97 \quad 100 \quad 108$

$3 \quad \underleftrightarrow{23 \quad 57} \quad 65 \quad 71 \quad 93 \quad 97 \quad 100 \quad 108$

$3 \quad 23 \quad \boxed{57} \quad \boxed{65} \quad \boxed{71} \quad \boxed{93} \quad \boxed{97} \quad \boxed{100} \quad \boxed{108}$

Pass 8: $\underleftrightarrow{3 \quad \boxed{23}} \quad 57 \quad 71 \quad 93 \quad 97 \quad 100 \quad 108$

$3 \quad 23 \quad 57 \quad 71 \quad 93 \quad 97 \quad 100 \quad 108$

Sorted Array using Bubble Sort.

* Complexity of bubble sort

↳ The no. of passes required may be betn 1 & n-1. There are n-1 comparison in 1st iteration, n-2 comparison in 2nd iteration and one comparison in last iteration. The total no. of comparison $(n-1)+(n-2)+(n-3) \ldots +1 = \frac{n(n-1)}{2}$, So, the time complexity is $\boxed{O(n^2)}$.

Advantages & disadvantages.

Advantage: Simple & easy to implement.

disadvantage: It is slowest method, becoz complexity is $O(n^2)$

disadvantage: Inefficient for large array size

Function for bubble sort:

```
Void Bubbled_Sort(int A[max], int n)
{
  int i, j, temp; for (i=0; i<n; i++) {
  for (j=0; j<n-i-1; j++)
  {
    if (A[j] > A[j+1])
    {
      temp = A[j];
      A[j] = A[j+1];
      A[j+1]= temp;
    }
  }
}
```

July 3, 202

* Insertion sort

↳ In insertion sort, the element is inserted at appropriate place. The array is divided into two parts, sorted and unsorted sub-array, in each pass 1st element of unsorted sub-array is picked up and moved into sorted sub-array, By inserting it in suitable position.

Algorithm:

```
Step 1: set i=1;
step 2: key = A[i];
Step 3: set j = i-1;
step 4: if key < A[j] then
            A[j+1] = A[j];
            j = j+1;
        repeat step 2 until j >= 0
Step 5: A[j+1] = key;
        i = i+1
step 6: repeat step 2 to 5
        till i<N
step 7: stop
```

Elements 22, 14, 25, 10, 5

22 | 14 25 10 5
sorted | unsorted
key = 14   14<22 = T

14 22 | 25 10 5
sorted | unsorted
key = 25   25<22 = F

14 22 25 | 10 5
sorted | unsorted
key = 10   10<25 = T
10<22 = T
10<14 = T

10 14 22 25 | 5
sorted | unsorted
key = 5;   5<25 = T
5<22 = T
5<14 = T
5<10 = T

5 10 14 22 25 5
→ Sorted Array

complexity of insertion sort:

Total no. of comparisons are $(n-1)+(n-2)+(n-3)+\dots+1$
which is $\frac{n(n-1)}{2}$

$$\Rightarrow \frac{n^2-n}{2}$$

$$\Rightarrow O(n^2)$$

Worst case complexity => $O(n^2)$
Average case complexity => $O(n^2)$
Best case complexity => $O(n)$

Advantages:

① Relatively simple & easy to implement

disAdvantage ② Inefficient for large array as time complexity is $O(n^2)$.

Advantage ③ Insertion sort is highly efficient if the array is already in sorted order, (Best case)

Imp ★

Quick Sort

Quick Sort is fastest sorting method, it follows divide & conquere method i.e., no. are divided & again-sub-divided, this division goes on until it is not possible to divide further, the procedure is applied recursively to the two part of the array on either side of pivot elemnt, it is also called partition exchange sort. Pivot element can be any element from the array, it can be 1st element, last element or any other element

Example

35, 50, 15, 25, 80, 20, 90, 45

P $\boxed{35}$ $\underset{L}{50}$ 15 25 80 20 90 $\underset{H}{45}$

check L ≥ P ✓
H ≤ P × dec

P $\boxed{35}$ $\underset{L}{50}$ 15 25 80 20 $\underset{H}{90}$ 45

check L ≥ P ✓
H ≤ P × dec

P $\boxed{35}$ $\underset{L}{50}$ 15 25 80 $\underset{H}{20}$ 90 45

check L ≥ P ✓ exchange
H ≤ P ✓

P $\boxed{35}$ $\underset{L}{20}$ 15 25 80 $\underset{H}{50}$ 90 45

check L ≥ P × inc
H ≤ P × dec

P $\boxed{35}$ 20 $\underset{L}{15}$ 25 $\underset{H}{80}$ 50 90 45

check L ≥ P × inc
H ≤ P × dec

P $\boxed{35}$ 20 15 $\underset{H}{\underset{L}{25}}$ 80 50 90 45

check L ≥ P × inc
H ≤ P ✓

$P\boxed{35}\ 20\ \ 15\ \ \underset{H}{25}\ \ \underset{L}{80}\ \ 50\ \ 90\ \ 45$

check ~~[illegible]~~ $L \geq P$ ✓ but H&L crossed, swap pivot
$H \leq P$ ✓

$P_1\boxed{25}\ \ \underset{L_1}{20}\ \ \underset{H_2}{15}\ +\infty \mid \boxed{35} \mid P_2\boxed{80}\ \ \underset{L_2}{50}\ \ 90\ \ \underset{H_2}{45}\ \ +\infty$

check $L_1 \geq P_1$ × inc $L_2 \geq P_2$ × inc
$H_1 \leq P_1$ × dec $H_2 \leq P_2$ × dec

$P\boxed{25}\ \ \underset{H_1}{20}\ \ \overset{L_1}{\underset{H_1}{15}}\ +\infty\ \ 35\ P\boxed{80}\ \ 50\ \ \underset{L_2}{50}\ \ \underset{H_2}{45}\ \ +\infty$

check $L_1 \geq P_1$ × inc | check $L_2 \geq P_2$ ✓ swap
$H_1 \leq P_1$ ✓ | $H_2 \leq P_2$ ✓

$P_1\boxed{25}\ 20\ \ \underset{H_1}{15}\ \overset{L_1}{+\infty}\ \ 35\ P_2\boxed{80}\ \ \underset{L_2}{50}\ \ \underset{\cancel{H_2}}{45}\ \ \underset{H_2}{90}\ \ +\infty$

Check $P_1 \geq P_1$ ✓ H&L | check $L_2 \geq P_2$ × inc
$H_1 \leq P_1$ ✓ crossed | $H_2 \leq P_2$ × dec

$15\ \ 20\ \ 25\ \ 35\ P\boxed{80}\ \ 50\ \ \underset{\underset{H_2}{L_2}}{45}\ \ 90\ \ +\infty$

check $L_2 \geq P_2$ × inc
$H_2 \nleq P_2$ ✓

$15\ 20\ 25\ 35\ P\boxed{80}\ 50\ \underset{H_2}{45}\ \ \underset{L_2}{90}\ +\infty$

~~each~~ crossed, swap pivot

15 20 25 35 45 50 80 90 sorted //

24 30 27 32 11 21 19

$P\boxed{24}\ \ \underset{L}{30}\ \ 27\ \ 32\ \ 11\ \ 21\ \ \underset{H}{19}$

check $L \geq P$ ✓ swap
$H \leq P$ ✓

$P\boxed{24}\ \ \underset{L}{19}\ \ 27\ \ 32\ \ 11\ \ 21\ \ \underset{H}{30}$

check $L \geq P$ × inc
$H \leq P$ ✓ dec

$P\boxed{24}\ \ 19\ \ \underset{L}{27}\ \ 32\ \ 11\ \ \underset{H}{21}\ \ 30$

check $L \geq P$ ✓ swap
$H \leq P$ ✓

$P\boxed{24}\ \ 19\ \ \underset{L}{21}\ \ 32\ \ 11\ \ \underset{H}{27}\ \ 30$

check $L \geq P$ × inc
$H \leq P$ × dec

$P\boxed{24}\ \ 19\ \ 21\ \ \underset{L}{32}\ \ \underset{H}{11}\ \ 27\ \ 30$

check $L \geq P$ ✓ swap
$H \leq P$ ✓

$P\boxed{24}\ \ 19\ \ 21\ \ 11\ \ 32\ \ 27\ \ 30$

check $L \geq P$ × inc ;
$H \leq P$ × dec

$P\ \boxed{24}\ \ 19\ \ 21\ \ \underset{H}{11}\ \ \underset{L}{32}\ \ 27\ \ 30$

crossed H&L swap pivot

$\underbrace{11\ \ 19\ \ 21}_{\text{sorted}}\ \ \boxed{24}\ \ 32\ \ 27\ \ 30$

$11\ \ 19\ \ 21\ \ 24\ \ P\ \boxed{32}\ \ \underset{L}{27}\ \ \underset{H}{30}\ \ +\infty$

check $L \geq P$ × inc

$H \leq P$ × dec

$11\ \ 19\ \ 21\ \ 24\ \ P\ \boxed{32}\ \ \underset{H}{27}\ \ \underset{L}{30}\ \ +\infty$

cross H&L swap pivot

$11\ \ 19\ \ 21\ \ 24\ \ \boxed{30}\ \ \underset{L}{27}\ \ \underset{H}{32}\ \ +\infty$

check $L \geq P$ × inc

$H \leq P$ × dec

$11\ \ 19\ \ 21\ \ 24\ \ P\ \boxed{30}\ \ \underset{H}{27}\ \ \underset{L}{32}\ \ +\infty$

crossed H&L, swap pivot

11 19 21 24 27 30 32

Sorted

July 8, 2025

Time complexity for best and average case is ~~nlog(n)~~ O(nlogn) and for worst case $O(n^2)$

Advantages: ① It is faster method among all sorting methods

② Its efficency is relatively good

③ Requires small amount of memory

Disadvantage: ① It is complex method of sorting.

② It is little hard to implement than other sorting algos

Merge Sort

↳ The basic concept of merge sort is divide the list into two smaller sub-list of approximately equal size, reccursively repeat this procedure till only one element is left.

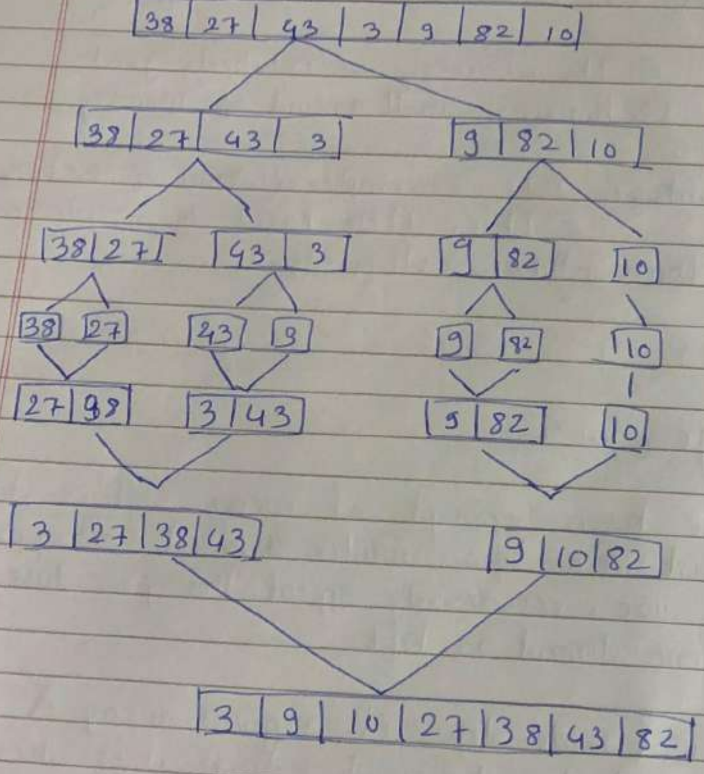
Step ① Divide step :- if a given array A has 0 or 1 element, simply return it is already sorted. other wise split into two sub-arrays, each combining half of the element.

Step ② Conquer step:- Conquer by recursively sorting two sub-arrays

Step ③ Combine step:- Combine the element by merging sorted sub-array and into a sorted sequence

Example

38, 27, 43, 3, 9, 82, 10



Complexity

↳ Time complexity of merge sort for best, worst and average case is O(nlogn).

* Searching Techniques Q

Searching algos are used to find elements in the list. Two searching algos are

① Linear Search Q
② Binary search Q

① Linear Search / Sequential search.

↳ Linear search also called as orderly search or sequential search, becoz, every key element is search ~~froo~~ from 1st element in an array i.e., a[0] to the last element i.e., a[n-1]

Algorithm :

```
step ①: i=0
step ②: if the element in the list is equal to
        the desired element then return 'i'
        if (A[i]==key) then
        return i
step ③: if i<n ~~i=i+1~~ then
        i=i+1; goto step ②
step ④: if i=n then return -1
step ⑤: STOP
```

Example

目 3 .

Complexity: O(n)

Advantages: ① Linear search is simple, easy to understand and implement

② It does not require thed data to be sorted in any particular order

③ If key element match the first element in array then linear search algo is best case in terms of executing time

Disadvantage: ① If the array size is very large then linear search isn't efficient.

② When key element match the last element in the array or key element does'nt match any element in array then this search algo is in worst case.

* Binary Search

Binary search is quicker than linear search, it cannot be applied on unsorted DS. ~~Bs~~ Binary Search is based on divide and conquer approach

Algorithm

```
S1: Initialize lb = 0, ub = n-1
S2:   while lb <= ub
S3:   mid = [ (lb + ub) / 2 ]
S4: if a[mid] == key
S5:   set pos = mid
S6:   break & jump to step 10
S7:   Else if key < a[mid]
S8:     ub = mid-1
S5:   else lb = mid+1
S10: if pos < 0
S11:   print "element not found"
S12:   else print pos.
```

Q consider array 5, 7, 11, 19, 30, 35, 42, key = 11

5	7	11	19	30	35	42

$mid = \frac{lb+ub}{2}$

$= \frac{0+6}{2}$

$= 3$

$a[3] > key$

$ub = mid-1$

$= 3-1$

$= 2$

$mid = \frac{ub+lb}{2} = \frac{0+2}{2} = 1$

$a[1] < key$

$ab = mid+1$

$= 1+1$

$= 2$

$mid = \frac{ub+lb}{2}$

$= \frac{2+2}{2}$

$= \frac{4}{2}$

$= 2$

$a[2] = key.$

* Advantages

↳ ① It is very time efficient as time complexit is O(log n)

* Disadvantage

↳ ① Array should be sorted before applying th algorithm

—— End ——

—— Khatam ——

—— समाप्त ——