26/11/12 (16) Linear Search, Binary Search, Bubble Sort, Arrays Class in Java

① Searching algorithm:- Searching algorithms are designed to check for an Element (or) retrieve an Element from any data structure where it is store.

(i) Linear Search:- it is defined as a sequential search algorithm that starts at one End and goes through Each Element of a list until the desired Element is found, otherwise the search continues till the End of the data set.

Eg:- ar ⇒

10	20	50	70	80	90

⇒ Key = 70

Algorithm:-

① Traversing :- Travers the array in Each position check wheather key is Present (or) not.

② Match the Key Element with array Element

③ if Key Element is found, Print the index position of the array Element.

④ if Key Element is not found Print "Key not found".

Code :- linear Search

```
int[ ] ar = {10, 20, 40, 30, 60, 70, 80, 90};
boolean flag = false;
// taking key from user.
S.o.p ("Enter key to search");
Scanner scan = new Scanner (System.in);
int key = scan.nextInt( );
// traverse through array to find key.
for(int i=0; i<ar.length; i++){
    if( key == ar[i]){
        S.o.p("Key found at index"+i);
        flag = true;
        break;
    }
}
if (flag == false){
    S.o.p("Key not found");
}
```

(ii) Binary Search :- is a searching algorithm used in a sorted array by repeatedly dividing the search interval in half.

⇒ Array must ~~sorted~~ be sorted. it is Mandatory Condition in Binary Search.

→ this algorithm works if the array is sorted.

→ Fast searching with less number of comparisions.

→ Binary Search is faster than linear search.

Eg:- a = {20, 10, 50, 30, 40, 70, 60, 80, 90}

(1) Sort this array and perform Binary Search.

a = {10, 20, 30, 40, 50, 60, 70, 80, 90}

↳ sorted.

↓ Mid pt.

10	20	30	40	50	60	70	80	90
0	1	2	3	4	5	6	7	8

Key = 30

(1) go to middle index and check wheathe key is equal to element in index. if key is present then print found at index.

③ if Key not found in the Middle Chek Key < Middle Value and Key > Middle value.

Eg:

10	20	30	40	50	60	70	80	90
0	1	2	3	4	5	6	7	8

i =

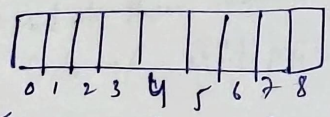
Mid Value

here :- Key < Middle value (50)
(30)

30 < 50.

④ if Key < Middle Value then Search operation done on left side of Mid Value.

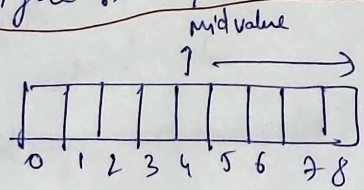
Eg:



Mid Value

Searching done on left side.

⑤ if Key > Middle Value then Search operation done on right side of Mid value.



Searching done on right.

Finding Mid Value :-

low = 0 (lowest index)
high = ar.length - 1 (highest index)

$$\boxed{mid = \frac{low + high}{2}}$$

Eg: ar →

10	20	30	40	50	60	70	80	90
0	1	2	3	4	5	6	7	8

↳ Mid.

Key = 30

(1) l = 0, h = 8 mid = $\frac{0+8}{2}$ ⇒ (4). ar[4] = 50

① key == ar[mid] ⇒ 30 == 50 ✗
② key < ar[mid] ⇒ 30 < 50 ✓ ⇒ h = Mid - 1
③ key > ar[mid] ⇒ 30 > 50 ✗

= 4 - 1

$\boxed{h = 3}$

(2) l = 0, h = 3 Mid = $\frac{l+h}{2}$ = $\frac{0+3}{2}$ ⇒ 1.5 ⇒ (1)

ar[1] = 20 30 == 20 ✗
30 < 20 ✗
30 > 20 ✓ ⇒ l = Mid + 1
= 1 + 1 = 2

(3) l = 2, h = 3 mid = $\frac{2+3}{2}$ ⇒ (2) ar[2] = 30

Key == ar(mid) ⇒ 30 == 30 ✓

$\boxed{\text{Key found}}$

Code :-

```
int[] ar = {10,20,30,40,50,60,70,80,90};
// taking key from user
S.o.p("Enter Key to Search");
Scanner Scan = new Scanner (System.in);
int Key = scan.nextInt();

int low = 0;
int high = ar.length -1;
while (low <= high){
  int mid = (low+high) /2;
  if(Key == ar[mid]){
      S.o.P("Key found at index" + mid);
      break;
  }
  Else if( Key < ar[mid]){
      high = mid -1;
  }
  Else if (Key > ar[mid]){
      low = mid +1;
  }
}
if (low > high){
      S.o.P("Key not found");
}
```

28/11/22

③ **Array class:-** is used to perform operation on arrays.

→ We can use Inbuit Methods.

→ all Methods are static.

→ Methods are invoke directly by class name.

Eg. Array.Sort[array name] → for Sorting array

Array.fill [array name, 5] → for fill array with Certain Value in all places.

→ we have to import
import.java.util.Arrays for to use This class

① **Array.fill( ) :-**

Code:-

```
int[ ] a = new int[4];
for (int Elem : a){
    s.o.p (Elem);
}
s.o.p( );
Arrays.fill(a,5); → filling array
for (int Elem : a){
    s.o.p(Elem);
}
```

I: o/p :- 0000

II: o/p : 5555

o/p :- 0000 → I
5555 → II

① Fill in Middle index :-

a → | 1 | 2 | 0 | 0 | 0 | 4 | 5 |
0 1 2 3 4 5 6

→ fill these ~~Elements~~ indexes.

```
int[] a = {1,2,0,0,0,4,5};
Arrays.fill(a,2,5,10);   ⇒ Arrays.fill(a, fromIndex, ToIndex, Value)
for(int Elem : a){
    S.O.P(Elem);
}
```

Where to ignore to fill

where to start filling

Inbuilt method

⇒ Arrays.fill(a, fromIndex, toIndex, Value);

- a → array name
- fromIndex → where to start filling
- toIndex → where to ignore filling
- Value → Value to fill.

② Array Sort Method :- Arrays.sort()

```
int[] ar = {20,30,50,10,25,60};
Arrays.sort(a);
for(int Elem : a){
    S.O.P(Elem);
}
```

o/p = 10 20 25 30 50 60

③ Binary Search :- Arrays Binary Search Method

Arrays.binarySearch(a, key);

↳ array name

Code :-

```
int a = {20,30,50,10,25,60};
Arrays.sort(a);   →
for(int Elem : a){
    S.O.P(Elem);
}
```

o/p :- 10 20 25 30 50 60

```
// using Arrays binary search Method.
S.O.Println();
int res = Arrays.binarySearch(a, 30);
S.O.P("Key found at index" + key);
```

→ array name

→ key to find in array.

→ gives index where key is presented.

o/p :- Key found at index 3

Arrays.binarySearch(arrayname, key);

↳ give index of key if it found.

↳ it give negative value if key not found.

⇒ if Key not found it gives negative Value.

Key = 15 a → 10 20 25 30 50 60

```
int res = Arrays.binarySearch(a, 15);
s.o.p(res);
```
↳ gives −2

(i) if Key 15 :- it would be in index 1

→ 15 fit between 10 & 20

10	20	25	30	50	60

↓ 15 (index 1)

⇒ −(index + 1)

⇒ −(1+1) ⇒ −2

(ii) if Key = 35

↳ fits b/w 30 and 50

10	20	25	30	50	60
0	1	2	3	4	5

⇒ it would in index 4

= −(4+1)

= −(5)

(4) Bubble Sort (or) Sinking Sort (or) Exchang Sort :-

In bubble sort algorithm, array is traversed from first element to last element. Here, current element is compared with next element if current element is greater than the next element, it is swapped.

Eg:- a → {7, 5, 2, 3, 1, 4, 6} → input

⇓

{1, 2, 3, 4, 5, 6, 7} → o/p after sort a

Steps to perform bubble sort :-

① step ① : going to 1st and comparing 1st and 0th index (here 7>5)

if it is true ↳ swap them.

7	5	2	3	1	4	6
0	1	2	3	4	5	6

⇒ swaping :- 7>5 → swap 7 and 5

5	7	2	3	1	4	6
0	1	2	3	4	5	6

→ Compare 7 and 2 and swap

7>2

5	2	7	3	1	4	6

→ Compare 7 and 3 and swap

7 > 3

5	2	3	7	1	4	6

→ Comparing 7 and 1 and swap

7 > 1 →

5	2	3	1	7	4	6

→ Comparing 7 & 4 and swap (7>4)

5	2	3	1	4	7	6

→ Comparing 7 and 6 and swap

7 > 6

5	2	3	1	4	6	7

→ for traversing through is done by i loop (outer loop). for sorting the array they use j loop