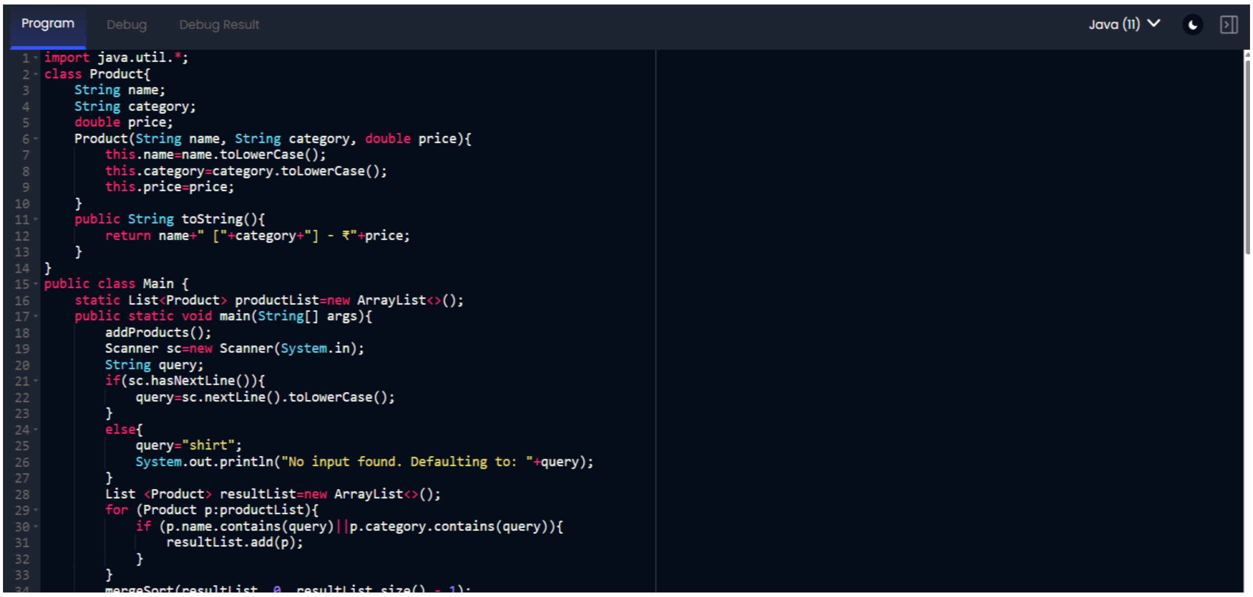
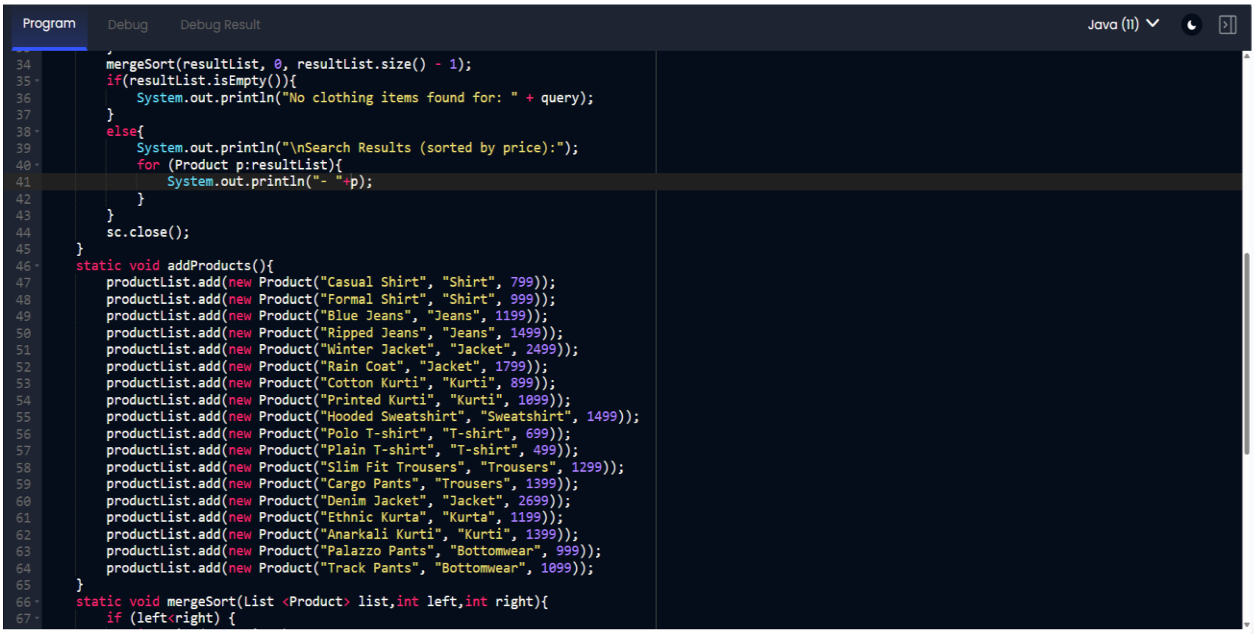
Exercise 2: E-commerce Platform Search Function

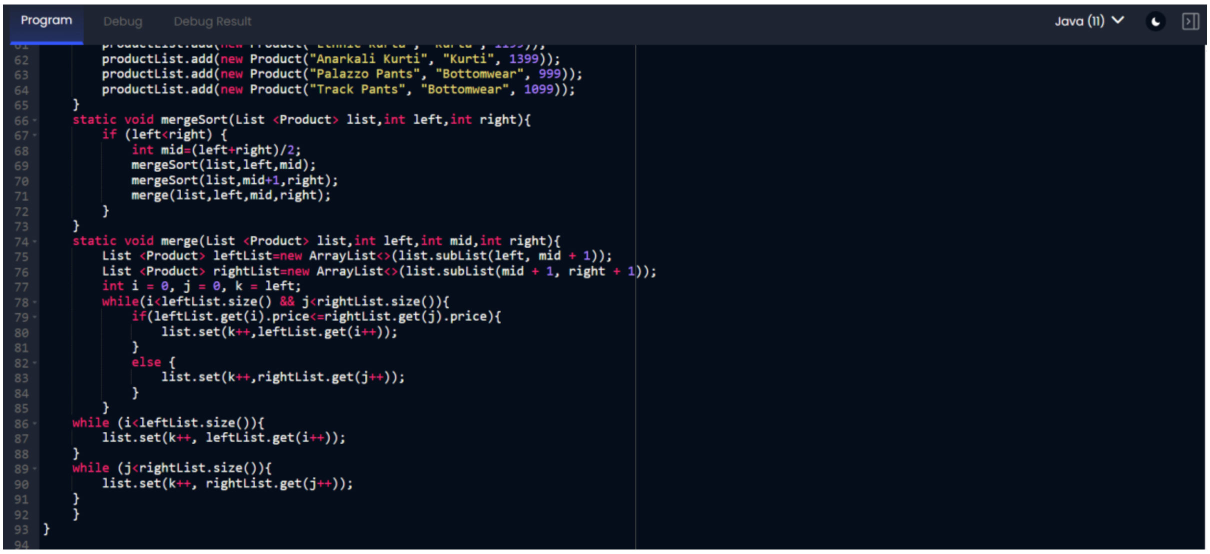
Clothing E-Commerce Platform



```java
import java.util.*;
class Product{
    String name;
    String category;
    double price;
    Product(String name, String category, double price){
        this.name=name.toLowerCase();
        this.category=category.toLowerCase();
        this.price=price;
    }
    public String toString(){
        return name+" ["+category+"] - ₹"+price;
    }
}
public class Main {
    static List<Product> productList=new ArrayList<>();
    public static void main(String[] args){
        addProducts();
        Scanner sc=new Scanner(System.in);
        String query;
        if(sc.hasNextLine()){
            query=sc.nextLine().toLowerCase();
        }
        else{
            query="shirt";
            System.out.println("No input found. Defaulting to: "+query);
        }
        List <Product> resultList=new ArrayList<>();
        for (Product p:productList){
            if (p.name.contains(query)||p.category.contains(query)){
                resultList.add(p);
            }
        }
        mergeSort(resultList, 0, resultList.size() - 1);
```

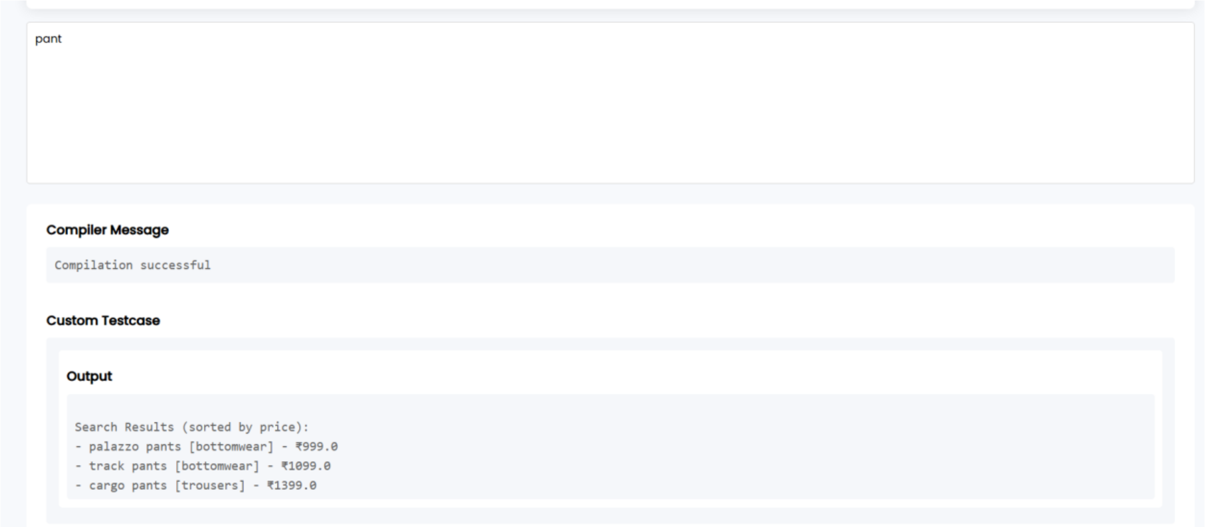


```java
        mergeSort(resultList, 0, resultList.size() - 1);
        if(resultList.isEmpty()){
            System.out.println("No clothing items found for: " + query);
        }
        else{
            System.out.println("\nSearch Results (sorted by price):");
            for (Product p:resultList){
                System.out.println("- "+p);
            }
        }
        sc.close();
    }
    static void addProducts(){
        productList.add(new Product("Casual Shirt", "Shirt", 799));
        productList.add(new Product("Formal Shirt", "Shirt", 999));
        productList.add(new Product("Blue Jeans", "Jeans", 1199));
        productList.add(new Product("Ripped Jeans", "Jeans", 1499));
        productList.add(new Product("Winter Jacket", "Jacket", 2499));
        productList.add(new Product("Rain Coat", "Jacket", 1799));
        productList.add(new Product("Cotton Kurti", "Kurti", 899));
        productList.add(new Product("Printed Kurti", "Kurti", 1099));
        productList.add(new Product("Hooded Sweatshirt", "Sweatshirt", 1499));
        productList.add(new Product("Polo T-shirt", "T-shirt", 699));
        productList.add(new Product("Plain T-shirt", "T-shirt", 499));
        productList.add(new Product("Slim Fit Trousers", "Trousers", 1299));
        productList.add(new Product("Cargo Pants", "Trousers", 1399));
        productList.add(new Product("Denim Jacket", "Jacket", 2699));
        productList.add(new Product("Ethnic Kurta", "Kurta", 1199));
        productList.add(new Product("Anarkali Kurti", "Kurti", 1399));
        productList.add(new Product("Palazzo Pants", "Bottomwear", 999));
        productList.add(new Product("Track Pants", "Bottomwear", 1099));
    }
    static void mergeSort(List <Product> list,int left,int right){
        if (left<right) {
```



```
        productList.add(new Product("Ethnic Kurta", "Kurta", 1199));
        productList.add(new Product("Anarkali Kurti", "Kurti", 1399));
        productList.add(new Product("Palazzo Pants", "Bottomwear", 999));
        productList.add(new Product("Track Pants", "Bottomwear", 1099));
    }
    static void mergeSort(List <Product> list,int left,int right){
        if (left<right) {
            int mid=(left+right)/2;
            mergeSort(list,left,mid);
            mergeSort(list,mid+1,right);
            merge(list,left,mid,right);
        }
    }
    static void merge(List <Product> list,int left,int mid,int right){
        List <Product> leftList=new ArrayList<>(list.subList(left, mid + 1));
        List <Product> rightList=new ArrayList<>(list.subList(mid + 1, right + 1));
        int i = 0, j = 0, k = left;
        while(i<leftList.size() && j<rightList.size()){
            if(leftList.get(i).price<=rightList.get(j).price){
                list.set(k++,leftList.get(i++));
            }
            else {
                list.set(k++,rightList.get(j++));
            }
        }
    while (i<leftList.size()){
        list.set(k++, leftList.get(i++));
    }
    while (j<rightList.size()){
        list.set(k++, rightList.get(j++));
    }
    }
}
```

Output:



```
pant
```

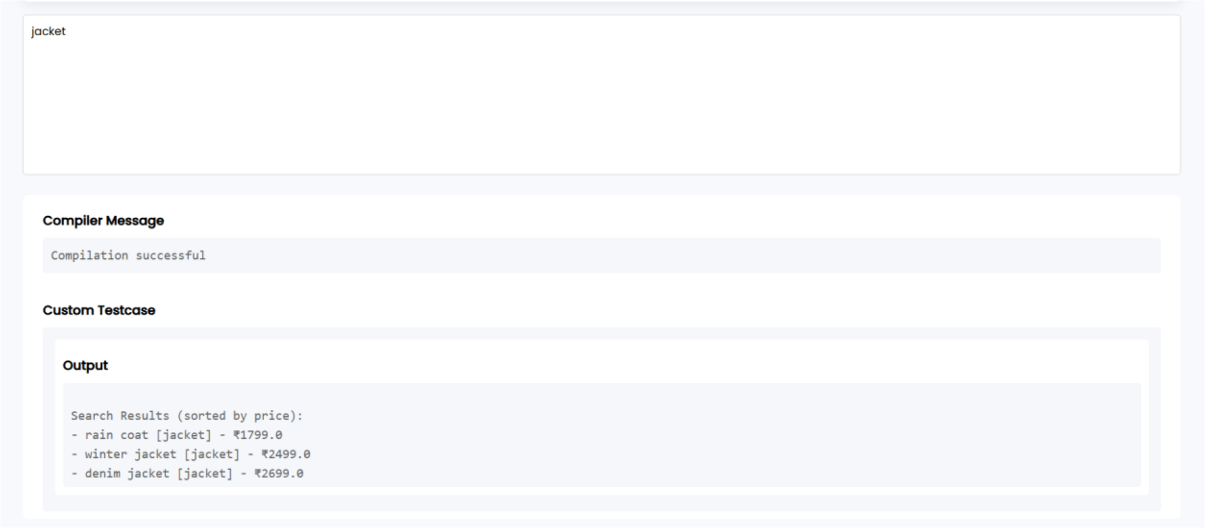
Compiler Message

```
Compilation successful
```

Custom Testcase

Output

```
Search Results (sorted by price):
- palazzo pants [bottomwear] - ₹999.0
- track pants [bottomwear] - ₹1099.0
- cargo pants [trousers] - ₹1399.0
```



jacket

Compiler Message

```
Compilation successful
```

Custom Testcase

Output

```
Search Results (sorted by price):
- rain coat [jacket] - ₹1799.0
- winter jacket [jacket] - ₹2499.0
- denim jacket [jacket] - ₹2699.0
```

Notes:

Arrays : Collection of datatype , all the type of the array must be same.

Syntax :

datatype [] variable_name = new datatype [size];

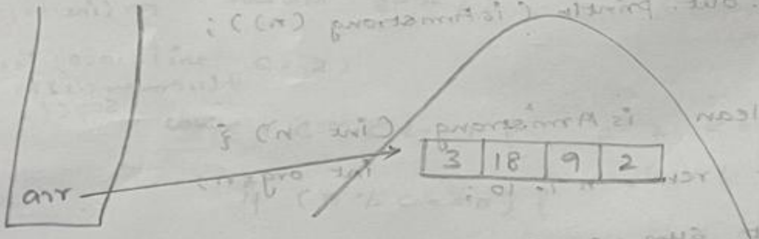
int [ ] ros ; ⇒ declaration of array. ros is getting defined in the stack.
↳ Ref var

ros = new int [5]; ⇒ obj is being created in the heap memory
⇒ Initialization

happens in compile time ⏜ int arr [ ] ; Runtime ⇒ Dynamic allocation.

int arr [ ] = new int [5] ;
↑ datatype ↑ ref var.
↓ memory used for creating an obj
↳ creating the object in heap memory.

Dynamic memory allocation - during the runtime the memory is allocated.



⇒ JVM handles this

3	8	19	12	7	28	33

⇒ In Other Languages it is continuous memory allocation

⟶ Dynamic memory allocation

Java ways if different , it does not Consist of pointer

⇒ It depends on JVM (Java Virtual Machine) whether it (Array) is continuous or not.

⇒ array objects are created in heap.

⇒ Heap objects are not continuous.

If array elements is not provided then the datatype accordingly it gives output.

Example :

```
int [] ros;
ros = new int [5];        O/P : 0  ⇒ Datatype = int
String []. arr = new String [4];
System.out.println( arr []);        O/P : null
```

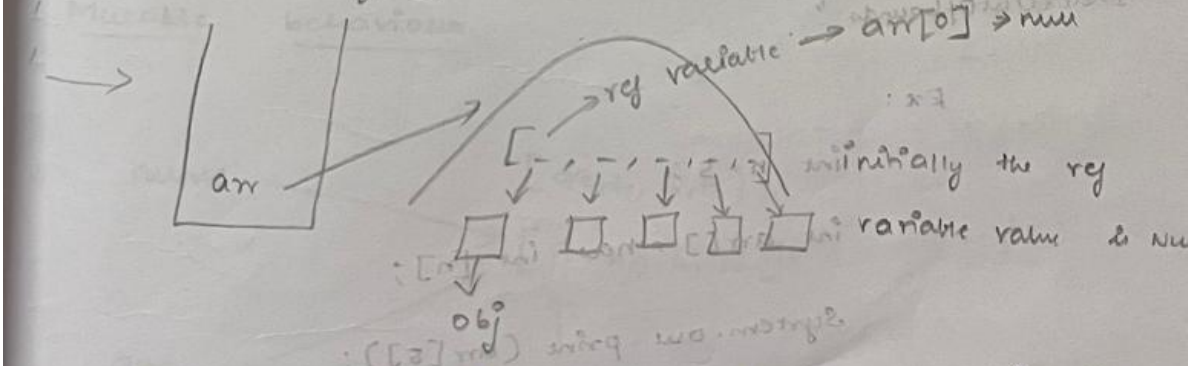
null is assigned to non-primitive datatype.

```
String []   arr = new String [5];
```

// internal working of object.

primitive datatype ⇒ it is stored in the
int, char, float   Stack memory. only.

Non-primitive datatype ⇒ it is stored in the
String, Array, Hashmap   heap memory.
[complex datatypes]



initially the ref variable value is Nu

⇒ All reference variable will point to null if the datatype is String

=> Java does not have continuous memory allocation it depends on JVM.

if the size of the array input is not known a

then:

Ex:
```
import java.util.*;
public class input { psvm() {
        Scanner s = new Scanner(System.in);
        for (int i=0 ; i< arr.length ; i++) {
                arr[i] = s.nextInt(); } }
    for ( int i=0 ; i<arr.length ; i++) {
            System.out.print ( arr[i] + " ");
        } }
```

```
for ( int num : arr) {
    System.out.print (num + " ");
}
```

Here:
Ex: arr = $\{\underbrace{23, 3, 4, 6, 51}_{num}\}$

num ⇒ every element of array

if size exceeds then it displays the error " IndexOutOfbounds ".

Ex:
```
int n = 5;
int arr[] = new int [n];
System.out.print (arr[5]);
```

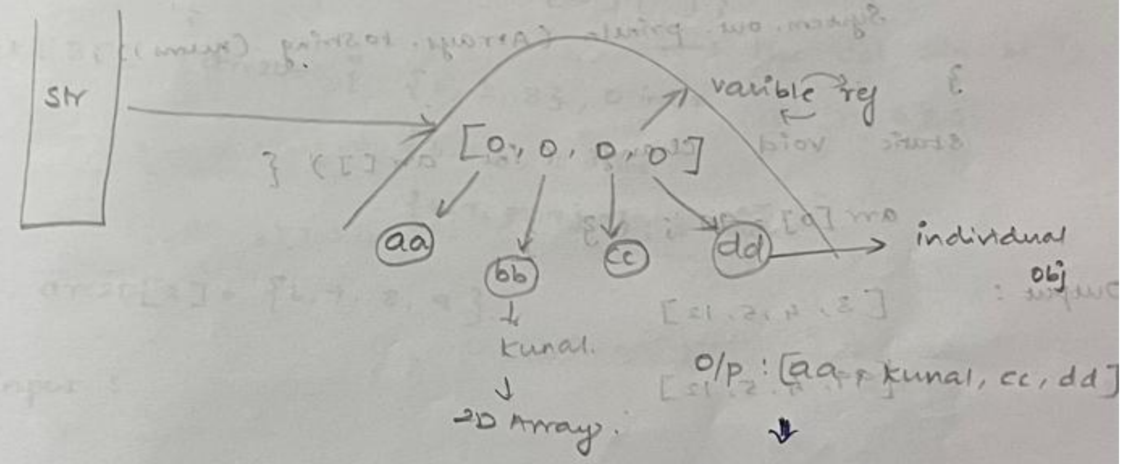
Then, error shows that IndexOutofBounds.

Arrays. toString ⇒ working.

```
String  str [] = new String [4];
for (int  i=0 ;  i< str.length ; i++) {
        Str [i]  =  s.next();
}
        Str [1] = "Kunal"; → 2D Array
System.out.println (Arrays.toString (str)); }  }
```

Input : aa bb cc dd

Output : [aa, bb, cc, dd]

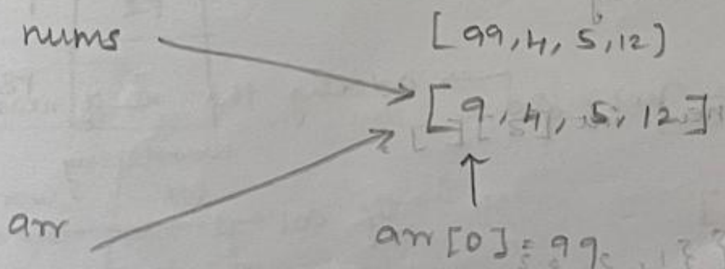


O/P : [aa, kunal, cc, dd]

⇓

2D Array. - directly accessing & changing eleme

Mutable behaviour.



Code.

Arrays are mutable in Java ⇒ you can change the obj yourself

String is immutable in Java.

Code:

```
import java.util.*;
public class PassingInFunctions { psvm() {
    int nums[] = {3, 4, 5, 12}
    System.out.println(Arrays.toString(nums));
    change(nums);
    System.out.println(Arrays.toString(nums));
}
static void change(int arr[]) {
    arr[0] = 99;    }  }
```

Output:

```
[3, 4, 5, 12]
[99, 4, 5, 12]
```

MultiDimensional Array - 2D Array : Matrix sort of a thing.

```
int[] ⇒ 1D Array.
int[][] ⇒ 2D Array.
int[][] arr = new int[3][];
```

→ adding the no. of rows is mandatory.

→ If col is not mentione then it is OK.

```
int[][] arr = { {1, 2, 3},
                {4, 5, 6},
                {7, 8, 9} };
```

Searching: LINEAR SEARCH:

Time complexity:

Constant ⇒ O(1) | O(N) ⇒ N → size of array

Best case | Worst case.

O(1) ⇒ Example.

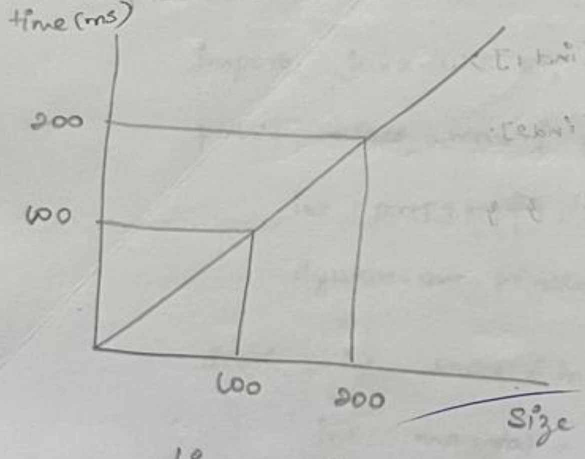
arr = [18, 9, 10 ...... 200 elements] ; target = 18

Ans: true ⇒ Only one comparison is made

Worst case: if there is no occurence of element in the array.

it completely iterates the array

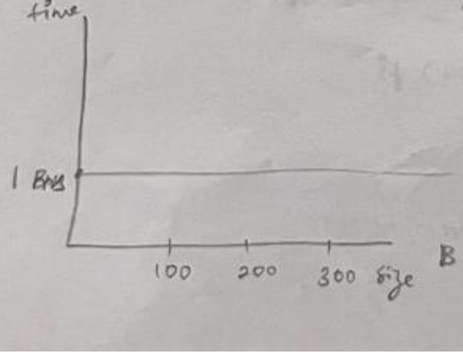
O(N) ⇒ iterates throughout the array.



100 elements = 100 ms

Worst case.

Linear time complexity



O(1) time complexity

Only one comparison.

Best case

Linear Search in 2D Array: min value java can hold is
[- 2147483648]

```
code :   import java.util.* ;  p class  search in 2D Array {
         public  static  void  main (String[] args) {
                 int [][]  arr =  { {23, 4, 1},        new int[][] is not used, so down
                                                         creating an obj
                                    {18, 2, 3, 9},
                                    {78, 99, 34, 99},
                                    {18, 22} };
                 int  target = 34 ;
                     int ans[] =
                 sop ( search (arr, length));
          }                  sop ( Arrays.toString ( ans));
       static  int[] search ( int [][] arr , int  target ) {
                                                     int max = arr[0][0];
            for ( int  row = 0 ; row < arr.length ; row++ ) {   ↳ Integer.
                                                                MIN_Va
                 for ( int  col = 0 ; col < arr [row]. length ; col++ ) {
                      if  (arr [row][col] == target ) {
                                        > max
                                                           max = arr[row][c
                          return   new  int [] {row, col} ; } }    ↓
                                                                 to
                          return   new  int [] {-1, -1} ;  } }    retu
```

Find the even number of digits :

Internal working :

nums = [18 , 124 , 9 , 1764 , 98 , 1]

Ans = 3

i) Count the no. of digits

176~~4~~   count : ~~0~~ ~~1~~ ~~2~~ ~~3~~ 4   ii) Convert 1764 ⇒ "1764"

17~~6~~   take the length

1~~7~~

~~1~~

0

* To find the no. of. digits.

```
    static int digits (int num) {
        return (int) (Math. log10 (num)) + 1 ;
    }
```

Shortcut to find the no. of. digits : Math. log10 (num) + 1

* iterating through the 2D Array.

```
arr = [[ 1, 2, 3 ]   r=0     for ( r=0 ; r < len (arr) ; r++ )
                                                     rowsum=0
       [ 4, 1, 6 ]   r=1    { for ( c = 0 ; c < len(row) ; c++ )
       [ 3, 3, 7 ]]  r=2       // every col of the each row
                                  rowsum += arr[r][c] }
```

<u>BINARY</u> <u>SEARCH</u>:

arr = [2, 4, 9, 10, 12, 14, 18, 19] ⇒ sorting

⟶ ascending Order

arr2 = [19, 12, 6, 5, 3, 2, -8, -14]

⟶ descending order

In Linear Search the max Comparisons is N times [iterates through the no. of. elements size]

Refer in book ⇒ some points.

points to be remembered ⇒ i) Sort the given array

ii) Find the middle element $mid = \frac{start + end}{2}$

iii) Check if the target > mid ⇒ search in right

else target < mid ⇒ search in left

iv) We found element if target == mid.

Example:

start
arr = [2, 4, 6, 9, $\boxed{11}$, 12, 14, 26, 36, 48]   end   Target = 12
0 1 2 3 $\boxed{4}$ 5 6 7 8 9
mid

$$mid = \frac{start + end}{2} = \frac{0+9}{2} = 4$$

mid < target
right side.

start   end
arr = [2, 4, 6, 9, 11, 12, 14, $\boxed{26}$, 36, 48]
0 1 2 3 4 5 6 $\boxed{7}$ 8 9
mid

$$mid = \frac{start + end}{2} = \frac{5+9}{2} = 7$$

mid ~~<~~ >target
So end comes to [mid-1]

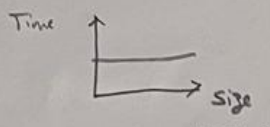
start end
arr = [2, 4, 6, 9, 11, $\boxed{12}$, 14, 26, 36, 48]
0 1 2 3 4 $\boxed{5}$ 6 7 8 9
mid

$$mid = \frac{5+6}{2} = \frac{11}{2} = 5$$ [target == mid]

★ point to be noted:

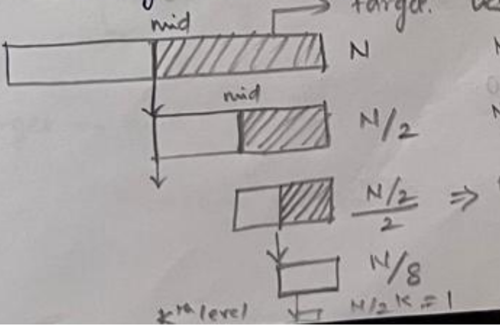
if start > end ⇒ then the element does not exists.

Best Case of Binary Search: if the first middle element is equal to the target then it is the best case. i.e mid == target [O(1)]



Best case.

★ Why Binary Search?



$N/2^0$

$N/2^1$

$N/2^2$ $\frac{\frac{N}{2}}{\frac{2}{1}} = \frac{N}{2} \times \frac{1}{2} = \frac{N}{4}$

$N/2^3$ $\frac{N/4}{2/1} = \frac{N}{4} \times \frac{1}{2} = \frac{N}{8}$

$$\frac{N}{2^k} = 1$$

$$N = 2^k$$

$$\log(N) = \log(2^k)$$

$$\log(N) = k \log 2$$

$$k = \frac{\log N}{\log 2}$$

$$k = \log_2 \textcircled{N} \rightarrow \text{size of the array.}$$

↓

total no. of comparison in worst case

Worst case of Binary search is $\Rightarrow O(\log n)$

* Comparison of Linear Search & Binary search

If a element should be find in the 1 million no

then in linear search [worst case]

It iterates 1 million time

in Binary Search [worst case]

It iterates only 20 times ↓

$\log_2(1000000)$

$$\log_2(1{,}000{,}000) = \frac{\log_{10}(1{,}000{,}000)}{\log_{10}(2)}$$

$$= \frac{6}{\log_{10}(2)} \approx \frac{6}{0.3010} \approx 19.93$$

$19.93 \approx 20.$

* Better way to find mid value.

int mid = (start + end)/2 ⇒ Start + end

⇓

It may exceeds the range of int in Java

⇓

It shows the error.

So, the better way to do the same thing is

* int mid = Start + (end - Start) / 2 ; ⊛

How it works :

$$m = s + \frac{(e - s)}{2}$$

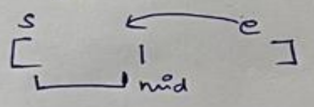
$$= \frac{2s + e - s}{2}$$

$$m = \frac{s + e}{2}$$

Three rules should be followed :

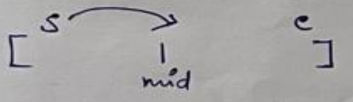
i) arr = $[\overset{start}{2}, 4, 6, 9, \overset{mid}{11}, 12, 14, 26, 36, \overset{end}{48}]$ target = 12

target < arr[mid]



end = mid-1

ii) target > arr[mid]



start = mid+1

iii) target == mid

return mid.

Binary Search : Basic code

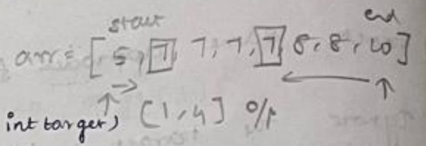
Search for the target and print in the index

```
public class BinarySearch {

    int[] arr = {2, 4, 6, 8, 10, 12, 14, 16};
    int target = 4
    int ans = binarysearch (arr, target);
    System.out.println(ans);  }

static int binarysearch (int[] arr, int target) {
    int start = 0;
    int end = arr.length - 1;
    while ( start <= end ) {
        int mid = (start + (end - start) /2;
        if (target < arr[mid]) {
            end = mid - 1 ; }
        else if ( target > arr[mid] ) {
            start = mid + 1 ; }
        else {
            return mid ;
        }
    } return -1;
  }
}
```

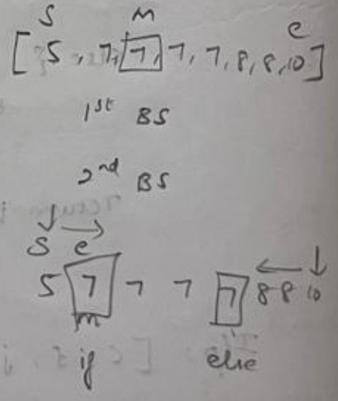
o/p : 1 ⇒ The index of the target

4) Find first and last position of Elements in sorted arr

I/p : nums = [5, 7, 7, 8, 8, 10] ; target = 8   o/p : [3, 4] else [-1,-1]



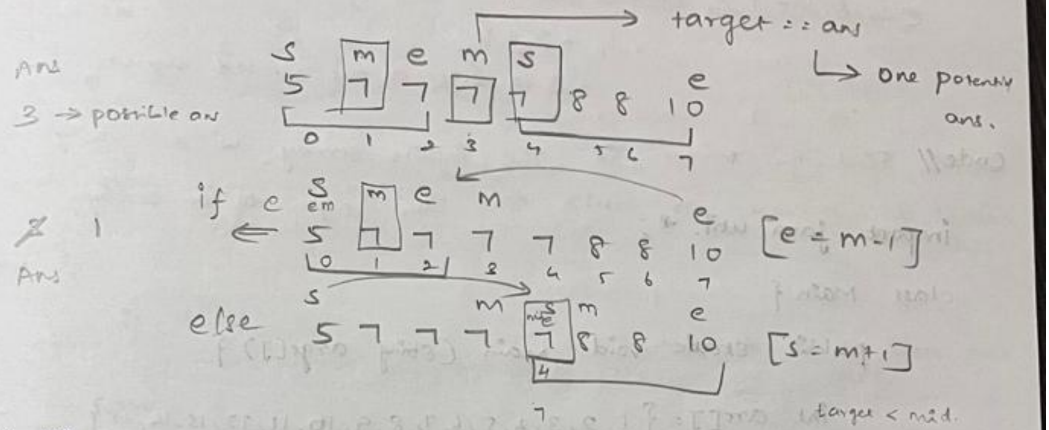
```
public int[] searchRange (int[] nums, int target)
        { int [] ans = {-1,-1} ;
                                     findstartIndex.:
          ans [0] = search (nums, target , true);
          if ( ans [0] != -1) {
               ans [1] = search (nums, target , false); }
          return ans; }

     int search (int[] nums, int target, boolean findstartIndex) {

          int ans = -1; int start = 0; int end = nums.length - 1 ;

     While ( start <= end) {
          int mid = start + (end - start) / 2 ;
          if (target < nums (mid) ) {
               end - mid-1 ; }
          else if (target > nums (mid]) {
               start = mid+1 ; }
          else { ans = mid ;
                                        // potential ans found
               if (findselection) {
                    end = mid-1; }
               else {
                    start = mid+1; }
               }
          }
               return ans ;
          }
```



Find a last occurrence of an element

arr = [5, 7, 7, 7, 7, 8, 8, 10] target = 7 o/p: [1, 4]

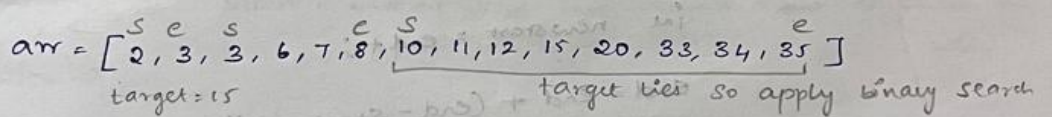
0 1 2 3 4 5 6 7



5) Find the position of an element in a sorted array of infinite numbers.

arr = [2, 3, 3, 6, 7, 8, 10, 11, 12, 15, 20, 33, 34, 35] target = 15

As we do no know the size of the array we should take start and end in the increasing way (i.e) First - 2 then 4 then 8. [double the size]


s e s e s e
arr = [2, 3, 3, 6, 7, 8, 10, 11, 12, 15, 20, 33, 34, 35]
target = 15
target lies so apply binary search

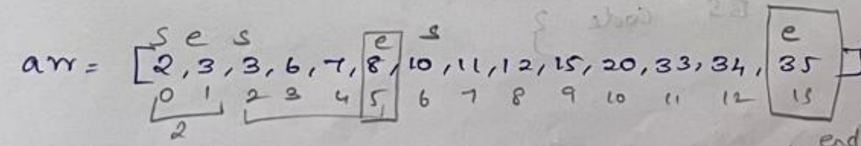
upto end greater than target keep doubling the size of start & end.

end = 1 + (1 - 0)* 2
= 1 + 2 * 2
= 5 ⇑

int newStart = end + 1

end = previous end + size * 2; end = end + (end - start + 1) * 2

⇓



end = 1 + 2 * 2
= 5

end = 5 + 4 * 2
= 13

end = 5 + (5 - 2 + 1)
= 5 + 4 * 2
end = 13