Linear Search in Java

Searching → it is a Process of finding a given value position in a list of values.

Linear/Sequencial search :-

- It is basic & simple search algorithms.
- In sequential search, we compare the the target value with all the other elements given in the list.

ex→ arr = [19, 12, 19, 77, 29, 50] (unsorted array)

target = 77 (start)

In above example, the target value is compare with all the elements in array in sequential / linear way.

Time complexity

→ Best case :- O(1) → constant.

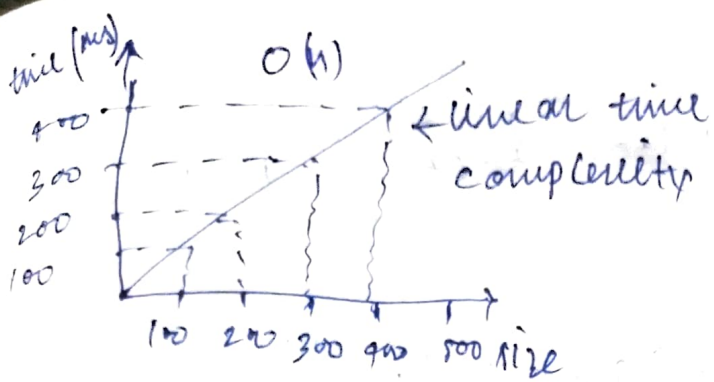
⇒ How many check will the loop make in best case i.e, the elements will be found at found at 0th index i.e, only one comparison will be made for best case.

→ Worst case :- O(n)

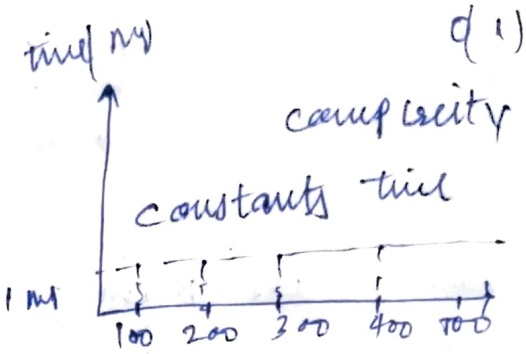
⇒ Worst case, here it will go through every element & then it says elements not found.

⇒

Size of array	No of comparison	time (ms)
100	100	100 ms
200	200	200 ms
n	n	



Worst case



Best case

arr = [18 , 12 , 9 , 14 , 77 , 50] size = 6

(0) (1) (2) (3) (4) (5)

unsorted.

Q1. find weather 14 exists in array or not.
if no value found, return -1.

Time complexity:

Best: O(1) // constant / worst case: O(N) N ⇒ size of array.

How many check will the loop make in
best case:- elements found at 0th index 2.
arr = [9, 9, 12, 18, ------ 200 elements]
target = 9.
1 comparison is best case. array is now
of size 1 lakh.
arr = [18, 12, 4, 7, ------ 1 lakh items]
(0) (1) (2) (3)
target = 18 answer ⇒ only one comparison make
is best cases.

Worst case:- You donot find the target item,
iterate / go through every item & then is the end
it say I did not find it.

size of array = 100 ⇒ 100 comparison = 100 ms.
200 ⇒ 200 comparison = 200 ms
1 lakh ⇒ 1 lakh comparison = 1 lakh ms

=

```
Public class Main {
Public static void main(String[] args) {
    int[] nums = {23, 45, 1, 2, 8, 19, -3, 16, -11, 2};
    int target = 19;
    boolean ans = linearSearch(nums, target);
    System.out.println(ans);
}
```

// 2nd way.

// search in the array: return index of item found.
otherwise of item not found return -1.

```
static int linearSearch(int[] arr, int target) {
        if(arr.length == 0) {
            return -1;
        }
    // run a for loop.
    for(int index=0; index<arr.length; index++) {
// check for element at every index if it is
    = target
    int element = arr[index];
```

```
        if (element == target) {
            return index;
        }
    }
    // this line will execute if none of the return
statements above have executed hence the target
    not found.
    return -1;
}

// 2nd way.
// search the target & return the index.
static int linearSearch2(int[] arr, int target) {
    if (arr.length == 0) {
        return -1;
    }
    // use a for loop
    for (int element : arr) {
        if (element == target) {
            return element;
        }
    }
// this line will execute if none of the
    return statements above have executed hence
    the target not found.
    return Integer.MAX_VALUE;
}
```

// 3rd way

```
// Search the target & return true & false.
static boolean binarySearch3(int[] arr, int target)
    if(arr.length==0){
        return false}
// use a for loop.
for(int element : arr){
  if(element==target){
    return true;
  }}
// this line will execute of none of the return
// statment above have executed. Hence the
// target not found.
    return false;
}
```

Question 2:- Reverse an String

```
= import java.util.Arrays.
Public class SearchStrings{
Public static void main(String[] args){
    String name = "Kumal";
    char target = 'u';
// Sout(Search(name, target));
// Sout(Arrays.toString(name.toCharArray()));
}
```

// 1st way :

```
static boolean search( String str , char target ) {
    if( str.length() == 0) {
     return false ; }
  for( int i =0; i < str.length(); i++) {
     if( target == str.charAt(i)) {
        return true ;
    } }
    return false ;
  }
```

// 2nd way :

```
static boolean search2( String str , char target) {
   if( str.length() == 0) {
    return false ; }
   for( char ch : str.toCharArray()) {
       if( ch == target) {
        return true ;
    }
   }
   return false ;
}
}
```

Question 3. Search an Array.

```
Public class SearchInRange {
Public static void main (String [] args) {
 int [] arr = { 10, 12, -7, 3, 14, 28};
  int target = 3456;
System.out.println( linearSearch ( arr, target, int start, int end) {
    if(arr.length == 0) {
        return -1;
    }
// run a for loop
for(int index = start; index <= end; index++){
// check for a element at every index if it is =
//   target.
  int element = arr[index];
    if(element == target) {
        return index;
    }
}
// this line will execute if none of the return statement
above executed hence the target not found.
    return -1;
}
}
```

Question 4. find minimum.

```
Public class FindMin {
Public static void main( String [] args) {
```

```
int [] arr = {19, 12, 7, 3, 14, 28};
System.out.println(min(arr));
}
// assume arr.length != 0
// return the minimum value in the array.
static int min(int arr[]) {
    int ans = arr[0];
    for(int i = 1; i < arr.length; i++) {
        if(arr[i] < ans) {
            ans = arr[i];
        }
    }
    return ans;
}
}
```

Question 5: Search in 2D Array.

```
import java.utils.Arrays.
Public class Search2DArray {
Public static void main(String args[]) {
int [][] arr = {
        {23, 4, 1},
        {19, 12, 3, 9},
        {79, 99, 34, 56},
        {19, 12}
};
```

```
    int target = 56;
int [] ans = search(arr, target); // to make of
    // return value {row, col}
System.out.println(Arrays.toString(ans));
System.out.println(max(arr));
System.out.println(output.MIN_VALUE);
}
static int [] search(int arr[][], int target) {
for(int row=0; row<arr.length; row++) {
for(int col=0; col<arr[row].length; col++) {
if(arr[row][col] == target) {
return new int [] {row, col};
}
}
}
return new int [] {-1, -1};
}
static int max(int [][] arr) {
int max = output.MAX_VALUE;
for(int [] ints : arr) {
for(int element : ints) {
if(element > max) {
max = element;
}
}
}
return max; } }
```

Leadcode Problem → max wealth.

```
Public class maxwealth {
Public static void main (String [] args) {   }
Public int maximumwealth( int [][] accounts ) {
    // Person = row
    // acount = col.
  int ans = integer.MIN_VALUE;
  for( int [] ints : accounts) {
// when you start a new row, take a new sum for that row.
   int sum =0;
  for( int anint: ints ) {
      sum + = anint;
  }
// now we have sum of accounts of Person. check with overall ans.
  if( sum > ans)
    ans = sum;
    } return ans;
    }
}
```

Question:-leetcode → even digits

```
Public class evendigits {
Public static void main ( String [] args) {
  int [] nums = {12, 345, 2, 6, 7896};
```

```
System.out.println(findNumbers(nums));
System.out.println(digits2(-349678));
}

static int findNumbers(int[] nums) {
    int count = 0;
    for(int num : nums) {
        if(even(num)) {
            count++;
        }
    }
    return count;
}

// function to check whether a number contains
   even digits or not.
static boolean even(int num) {
    int numberOfDigits = digits(num);
    return numberOfDigits % 2 == 0;
/*
    if(numberOfDigits % 2 == 0) {
        return true;
    }
    return false;
*/

}
```

```
static int digits2(int num) {
    if(num<0) {
        num = num * -1;
    }
return (int) (Math.log10(num)) + 1;
}

// count number of digits in a number.
static int digits(int num) {
 if(num<0) {
   num = num * -1;
 }
 if(num==0) {
    return 1; }

 int count = 0;
 while(num>0) {
    count++;
   num = num/10;
 }
 return count;
}
}
```

Binary Search

Algorithms :- ← Sorted Array

arr = [2, 4, 6, 9, 11, 12, 14, 20, 36, 48]

↑ ascending order.

target = 36

1. Find the middle element
2. check :-
   if target > middle ⇒ search in right.
   else ⇒ search in left.
3. if target == middle ⇒ we found element.

Example :- on the above array. target = 36

arr = [2(0), 4(1), 6(2), 9(3), (11)(4), 12(5), 14(6), 20(7), 36(8), 48(9)]

1st ⇒ find middle element

$$mid = \frac{start + end}{2} = \frac{0+9}{2} = 4$$ [the element at index 4 is the middle element].

2nd → let's check :-

if target > middle ⇒ 36 > 11 ⇒ Yes! ⇒ check in right side.

now, arr = [2, 4, 6, 9, 11, 12(5), 14(6), (20)(7), 36(8), 48(9)]

$$mid = \frac{5+9}{2} = 7$$ [the element at index 7 is the middle elemt]

wects check,

if target > middle ⇒ 36 > 20 ⇒ Yes! ⇒ check in right side.

i.e, arr = [2, 4, 6, 9, 11, 12, 14, 20, 36, 49]

mid = $\frac{8+9}{2}$ = 8 [the element at index 8 is the middle elemt.].

Here,

target == middle ⇒ 36 == 36 ⇒ we found element at index 8.

Here, the space in which we are searchy is getting divided into two spaces.

⇒ time complexity :-

Best case :- O(1)

worst case :- O(log n)

= Order agnostic Binary Search

with say if we don't know that the array is sorted in ~~asc~~ ascendig, or descendig order.

arr = [90, 75, 18, 12, 6, 4, 3, 1] target = 7

Here, target > middle ⇒ search in left.

Here, start > end → Descendig order

90 > 1

when

start < end → Ascending order

$$\text{middle mid} = \text{start} + \frac{(\text{end} - \text{start})}{2}$$

Binary search

```
Public class Binarysearch {
    Public static void main(String [] args) {
int [] arr = {-18, -12, -4, 0, 2, 4, 15, 16, 18, 22, 99};
 int target = 22;
 int ans = binarysearch(arr, target);
  System.out.println(ans);
 }
// return the index
// return -1 if it does not exist.
 static int binarysearch(int [] arr, int target) {
      int start = 0;
      int end = arr.length - 1;
while(start <= end) {
// find the middle element
// int mid = (start + end) / 2; might be possible that
(start + end) exceeds the range of int in Java.
 int mid = start + (end - start) / 2;
   if(target < arr[mid]) {
      end = mid - 1;
   } else if(target > arr[mid]) {
      start = mid + 1; }
```

```
else {   // as found
    return mid;
}
}
return -1;  }}
```

= ① find the middle elements
② target > mid = search in right side.
③ else search in left sides.
④ if middle element == target elements.

arr = [2, 4, 6, 9, 11, 12, 14, 20, 36, 40] target = 36

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

arr = [2, 4, 6, 9, 11, 12, 14, 20, 36, 40]

M = $\frac{5+9}{2} = \frac{14}{2} = 7$

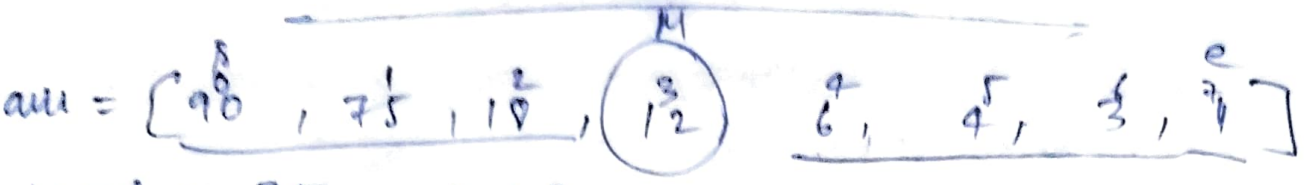
mid = [2, 4, 6, 9, 11, 12, 14, 20, 36, 40]

mid = $\frac{8+9}{2} = \frac{17}{2} = 8$

target == mid (36)

worst case = log N

Order agnostic Binary search



target = 75, $\frac{0+7}{2} = 3$

target > middle ⇒ left

$s = m + 1$

target < middle ⇒ right

$e = m - 1$

```
public class OrderAgnostic {
int arr[] = {-18, -12, -4, 0, 2, 3, 4, 15, 16, 18, 22, 48};
int arr[] = {99, 88, 75, 22, 11, 10, 5, 2, -3};
    int target = 22;
    int ans = orderAgnostic(arr, target);
    System.out.println(ans);
}
static int orderAgnostic(int arr[], int target){
    int start = 0;
    int end = arr.length-1;
// find whether the array is sorted in
   ascending or descending.
boolean isAsc = arr[start] < arr[end];
    while(start <= end){
// find the middle element
//int mid = (start + end)/2; might be possible
//that exceeds the range of int in java
```

```
int mid = start + start (end - start) / 2 ;
if( arr[mid] == target) {
    return mid ;
  }
if( asc) {
  if(target < arr[mid]) {
    end = mid - 1 ;
  } else {
    start = mid + 1 ;
  }
} else {
    if( target > arr[mid]) {
      end = mid - 1 ;
    } else {
      start = mid + 1 ;
    }}}
  return -1 ;
 }}
```

Q1. Row columns matrix .

```
import java.utils.Arrays ;
Public class Rowcolumns matrix {
Public static void main( String [] args) {
  int [] [] arr = {
        {10, 20, 30, 40},
        {15, 25, 35, 45},
        {28, 29, 37, 49},
        {33, 34, 38, 50}  } ;
```

```
System.out.println(Arrays.toString(search(arr, 99)));
}

static int[] search(int[][] matrix, int target) {
    int r = 0;
    int c = matrix[0].length - 1;
    while(r < matrix.length && c >= 0) {
        if(matrix[r][c] == target) {
            return new int[] {r, c};
        }
        if(matrix[r][c] < target) {
            r++;
        } else {
            c--;
        } } return new int[] {-1, -1};
    }
}
```

Q2. Sorted Matrix Sort.

```
import java.util.Arrays;
Public class Sorted Matrix {
Public static void main(String[] args) {
  int[][] arr = { {1, 2, 3},
                  {4, 5, 6},
                  {7, 8, 9} };
System.out.println(Arrays.toString(search(arr, 9)));
}
```

// search in the row Provided & the col Provided.

```
static int [] binarySearch(int [][] matrix, int row,
int cStart, int cEnd, int target) {
    while(cStart <= cEnd) {
    int mid = cStart + (cEnd - cStart)/2;
    if(matrix[row][mid] == target) {
     return new int [] {row, mid};
    }
     if(matrix[row][mid] < target) {
        cStart = mid+1;
      } else {
        cEnd = mid-1;
     } }
     return new int[] {-1,-1};
   }

static int [] search(int [][] matrix, int target) {
    int row = matrix.length;
    int cols = matrix[0].length; // be cautious,
                                    matrix may be empty
    if(cols == 0) {
      return new int[] {-1,-1};
    }
    if(row == 1) {
    return binarySearch(matrix, 0, 0, cols-1, target);
    }
     int rStart = 0;
     int rEnd = row - 1;
     int cMid = cols/2;
```

```
// run the loop till 2 rows are remaining.
while (rstart < (rend - 1)) { // while this is true it
will have more than 2 rows.
int mid = rstart + (rend - rstart) / 2;
if (matrix[mid][cmid] == target) {
  return new int[] {mid, cmid};
}
if (matrix[mid][cmid] < target) {
  rstart = mid;
} else {
  rend = mid; }}
// Now we have two rows
// check whether the target is in the col of 2 rows
if (matrix[rstart][cmid] == target) {
return new int[] {rstart, cmid}; }
if (matrix[rstart+1][cmid] == target) {
return new int[] {rstart+1, cmid}; }
// search in 1st half
if (target <= matrix[rstart][cmid-1]) {
return binarySearch(matrix, rstart, 0, cmid-1, target);
}
// search in 2nd half.
if (target >= matrix[rstart][cmid+1] && target <=
matrix[rstart][cols-1]) {
return binarySearch(matrix, rstart, cmid+1,
cols-1, target); }
```

// search in 3rd half.

if(target <= matrix[rstart+1][cmid-1]) {

return binarySearch(matrix, rstart+1, 0, cmid-1, target);

} else {

return binarySearch(matrix, rstart+1, cmid+1,

cols-1, target);

}}.

Sorting

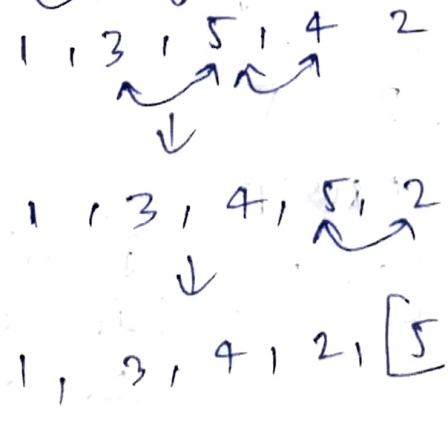
① Bubble Sort

Sorting → It is a process of arranging items systematically.

Bubble Sort :→ It is the simplest algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

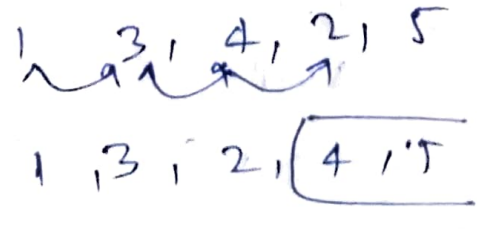
Example :- 3, 1, 5, 4, 2

First Pass :-



With first pass through the entire array, the largest element (here, 5) came to the end.

Second Pass :-



With second pass second largest element came at second last index.

1, 3, 2, 4, 5 — don't need to compare again & again since it is already sorted.

third Pass :- 1, 3, 2, 4, 5

1, 2, 3, 4, 5 sorted

* Bubble sort is also known as sinking sort or exchange sort.

let understand more deeply how actually bubble sort works.

counter
i = 0
1st Pass

3, 1, 5, 4, 2
0 1 2 3 4

swap

1 3 5, 4, 2

swap

1, 3, 4, 5, 2 → outermost loop it runs (n-1) times

i = 1
2nd Pass

1, 3, 4, 2, 5

swap

1, 3, 2, 4, 5 — j will only check this because element after already sorted

i = 2
3rd Pass

1, 3, 2, 4, 5

swap

1, 2, 3, 4, 5 sorted.

Complexity

* Space complexity :- O(1) // constant ⇒ since here no extra space is required i.e, like copying the array etc. is not required.
also known as inplace sorting algorithms.

* Time complexity :
① Best case → Array is sorted.

i=0 first pass

1 , 2 , 3 , 4 , 5 → only once it runs we don't need to check it again.

NOTE → when i never swaps for value of i, it means array is sorted. Hence, you can end the program.

* **Best case comparisons = N-1 ⇒ N**

since in time complexity constants are ignored, we don't want exact time, we just want relationship i.e, mathematical function.

② Worst case → sorting descending order array to ascending order.

5, 4, 3, 2, 1
j

4, 5, 3, 2, 1
j

4, 3, 5, 2, 1
j

4, 3, 2, 5, 1

4, 3, 2, 1, 5 ⇒ (N-1) swap.

i=1 second pass

4, 3, 2, 1, 5 → already sorted, run loop only for this

3, 4, 2, 1, 5

3, 2, 4, 1, 5

3, 2, 1, 4, 5 (N-2) swaps

i=2 third swaps

3, 2, 1, 4, 5

2, 3, 1, 4, 5

2, 1, 3, 4, 5 ⇒ (N-3) swaps

i=3 ~~fourth~~ swaps

2, 1, 3, 4, 5

1, 2, 3, 4, 5 ⇒ (N-4) swaps

total comparison ⇒ $N-1+N-2+N-3+N-4$

$\Rightarrow 4N - (1+2+3+4)$

$\Rightarrow 4N - \left[\frac{N \times (N+1)}{2}\right]$

$\Rightarrow 4N - \frac{N^2+N}{2}$

$\Rightarrow O\left(\frac{7N-N2}{2}\right)$

total comparison $= O(N2)$ ⇒ time complexity

constant i.e. denominating terms are ignored.

* Stable Sorting Algo → (10) (20) (20) (3) (10)

After Sorting ↓

(10) (10) (20) (20) (30)

⇒ Here after Sorting the order is maintained
⇒ order is same when value is same.

* unstable Sorting Algo :- (10) (20) (20) (30) (10)

After Sorting →

(10) (10) (20) (20) (30)

Code :-

```
import java.util.Arrays;

Public class Main {
Public static void main(String [] args) {
   int [] arr = {5, 3, 4, 1, 2};
      bubble (arr);
System.out.Println(n| Arrays.toString(arr));
   }

   Static Void bubble(int [] arr) {
        boolean swapped;
    // run the steps n-1 times
    for| int i=0; i<arr.length; i++) {
        swapped = false;
// for each step max item will come at the last
    respective index.
for| int j=1, j<arr.length-i ; j++) {
// swap if the item is smaller than the previous
    item.
```

```
if( arr[j] < arr[j-1] ){
    //swap
    int temp = arr[j];
    arr[j] = arr[j-1];
    arr[j-1] = temp;
    swapped = true;
}
```

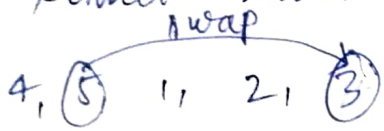
// if you did not swap for a particular value of i, it means the array is sorted hence stop the program

```
if(!swapped){ // false = true
    break;
}}}
```

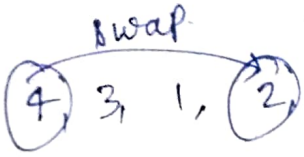
(2) Selection sort

Selection sort :- Select an element & put it on its correct index

Ex



[get the greatest element of the array & put it on its correct index]



1, 2, 3, 4, 5 → sorted

→ Here we select maximum elements & put it on right side, we can also do vice versa. select minimum element & put it on right side.

complexity :-

total comparison ⇒ 4, 5, 1, 2, 3 (n-1)

4, 3, 1, 2, 5 (n-2)

already at correct position ignore in future steps. → 2, 3, 1, 4, 5 (n-3)

2, 1, 3, 4, 5 (n-4)

1, 2, 3, 4, 5 ⋮ 0

⇒ $0 + 1 + 2 + 3 + 4 \cdots (n-1) = \frac{n(n-1)}{2}$

⇒ $\boxed{\frac{n^2 - n}{2}}$

Worst case → $O(N^2)$

Best case → $O(N^2)$

⇒ it is not stable sorting algorithms.

⇒ It performs well on small lists.

Code

```
import java.util.Arrays;
Public class Main {
Public static void main (String [] args) {
```

```
int arr[] = {5, 3, 4, 2};
selection(arr);
System.out.println(Arrays.toString(arr));
}

static void selection(int[] arr){
for(int i=0; i<arr.length; i++){
// find the max item in the remaing array &
// swapp with correct index.
int last = arr.length-i-1;
int maxIndex = getMaxIndex(arr, 0, last);
swapp(arr, maxIndex, last);
}
}

static void swap(int[] arr, int first, int
        second){
int temp = arr[first];
arr[first] = arr[second];
arr[second] = temp;
}

static int getMaxIndex(int[] arr, int start,
        int end){
int max = start;
for(int i=start; i<=end; i++){
```

```
    if (arr[max] < arr[i]) {
        max = i;
    }
    return max;
}
}
```

(2) Insertion sort

array ⇒ 5, 3, 4, 1, 2 [Partially sorting the array]

for every index, put that index element at the correct index of LHS.

1st pass → i = 0 → this will be sorted.

[5, 3], 4, 1, 2

3, 5, 4, 1, 2

2nd pass → i = 1 → this will be sorted

[3, 5, 4], 1, 2

3, 4, 5, 1, 2

3rd pass → i = 2 → this will be sorted.

[3, 4, 5, 1], 2

1, 3, 4, 5, 2

4th pass → i = 3

[1, 3, 4, 5, 2] → this will be sorted

[1, 2, 3, 4, 5] → array is sorted.

* To understand what every 'i' is doing.

outer loop

i	j

5, 3, 4, 1, 2
0 1 2 3 4

Sort array till index 1 pass ① → 0

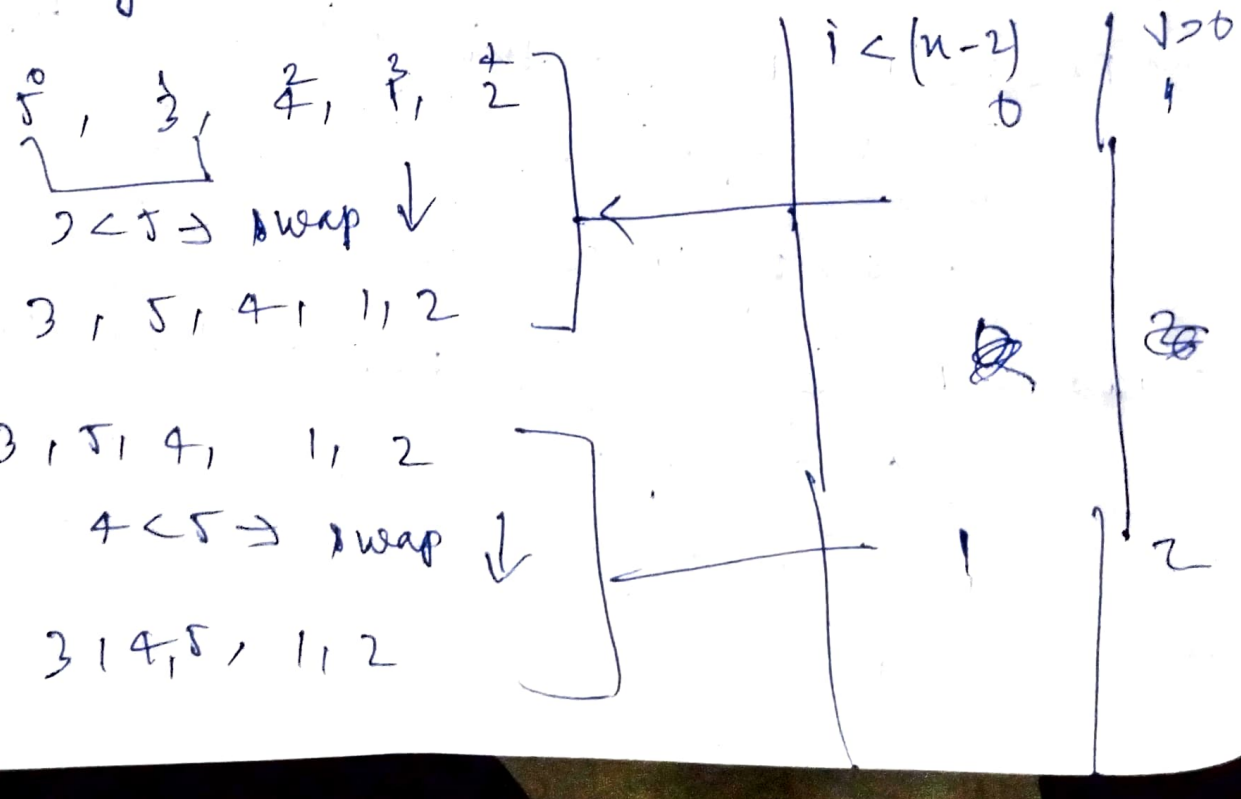
Sort array till index 2 pass ② → 1

Sort array till index 3 pass ③ → 2

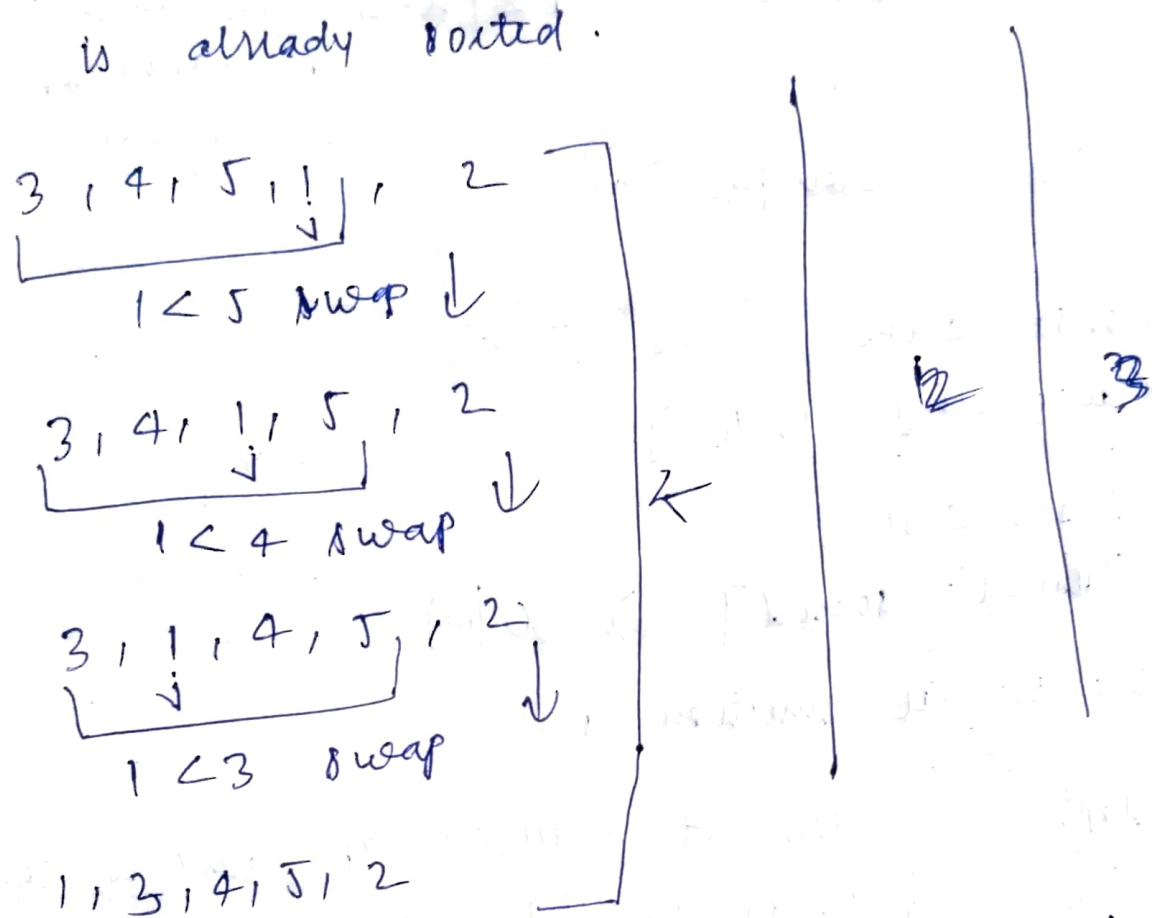
Sort array till index 4 pass ④ → 3

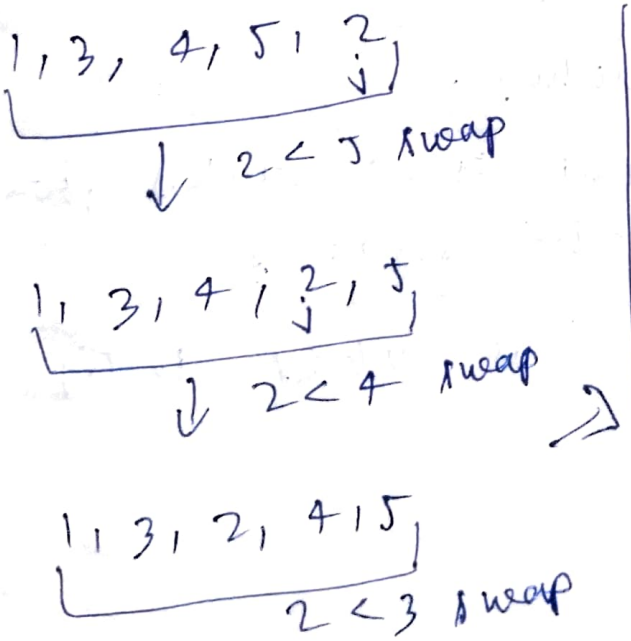
i.e 'i' will run from 0 to (n-2)

* Working



Now, since 3<4 already noted when element j is not smaller than element (j-1) break the loop because the previous (LHS) side array is already sorted.





1, 3, 4, 5 | 2

Sorted already so break.

1, 2, 3, 4, 5

Now if we take $i = 4$, then $j = 5$ which is going out of bound.

therefore, we take $\boxed{i < (n-1)}$ where n is length of array

Time complexity

① Worst case ⇒ $O(n^2)$
[descending sorted]

② Best case
[already sorted] ⇒ $O(n)$

Why to use insertion sort

* Adaptive :- steps get reduced if array is sorted [i.e, no of swap are reduced as compare to unsorted to sort.]
* Stable sorting algorithms.
* use for smaller value of n: works good when array is partially sorted
  it takes part in hybrid sorting algorithms.

code

```
import java.util.Arrays;
Public class Main{
Public static void main(String [] args) {
int arr[] = {5,3,4,1,2};
insertion(arr);
System.out.Println(Arrays.toString(arr));
}
static void insertion(int [] arr) {
for(int i=0; i<arr.length-1; i++){
for(int j=i+1; j>0; j--) {
if(arr[j] < arr[j-1]) {
swap(arr, j, j-1);
} else {
break;
}}}}
Static void swap(int [] arr, int first, int second){
int temp = arr[first];
arr[first] = arr[second];
arr[second] = temp;
}
}
```

4. cyclic sort

cyclic sort :→ when given numbers from range 1 to N → use cyclic sort.

eg → 3, 5, 2, 1, 4

Q1. what is cyclic sort & how its work?

Let say N = 5 → { Here the no's are jumbled sort are from 1 to 5 }

{ when the array is sorted in that case all the numbers are going to be at their correct indices. }

so after → 1, 2, 3, 4, 5 → { Here after sorting index will be value - 1 }.

index = value - 1 { using the we gonna sort the array }.

why?

→ Because index starts from 0.

check → swap → move

Q1. Might be → You have given the array find the missing numbers.

Or you have given from 1 to n - find the duplicate number.

* worst case → (3)[0], 5[1], 2[2], 1[3], 4[4]

check is 3 at the correct index if not then do 3-1 = 2 (index = value-1) the swap with correct index.

∴ 2, 5, 3, 1, 4 → { after swapping we know that 3 is at correct Position now but we do not know whether the other number that come at the Position of 3 is correct or not so, check again.

① { if 2 is correct Position no it's not the again swap it, because it should be at index no 1. but it's at no, 0 so, swap it with index

(5), 2, 3, 1, 4 → } same theory as above }

(4), 2, 3, 1, 5 → { same }

{ 1, 2, 3, 4, 5 } → { Now we check if 1 is at the correct Position, if it is then move forward, 4 so on Hence

* we know that every unique item is only getting swapped once.

* Here we are not incrementing i when we are swapping so that might result is more than n iteration of the loop.

* Worst case → N-1 (swap)
→ (N-1) + N
→ (2n-1)
∴ O(N) linear.

= code

Public class CyclicSort {
Public static void main (String [] args) {
int [] arr = {3, 3, 2, 1, 4};
Sort(arr);
System.out.println(Arrays.toString(arr));
}

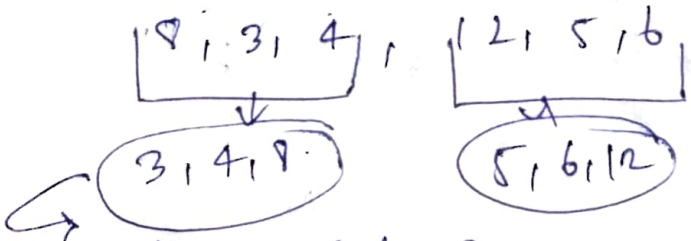
static void Sort (int [] arr) {
int i = 0;
while (i < arr.length) {
int correct = arr[i] - 1;
if (arr[i] != arr[correct]) {
swap(arr, i, correct);
} else {
i++; } } }

```
static void swap(int[] arr, int first,
        int second) {
int temp = arr[first];
arr[first] = arr[second];
arr[second] = temp;
}
```

Using Recursion

* merge sort using Recursion

* merge sort → In Recursion divide the array into two parts & sort the first part & second parts give the the array & then we will merge that.



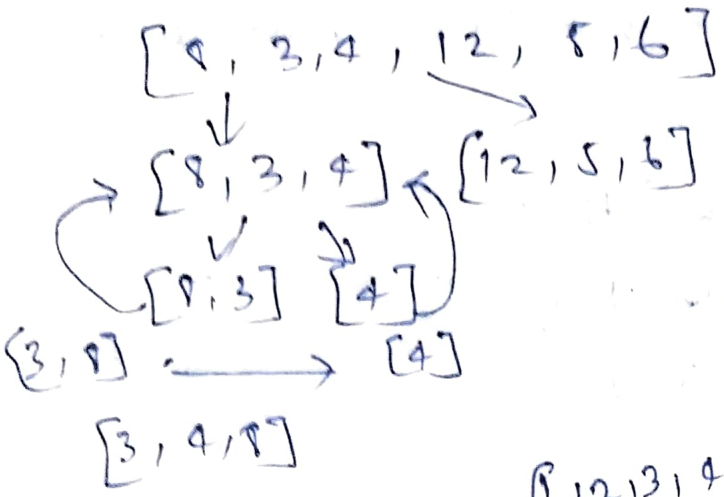
I will check who is smaller so I will take 2 pointer i, j then we will comparision :

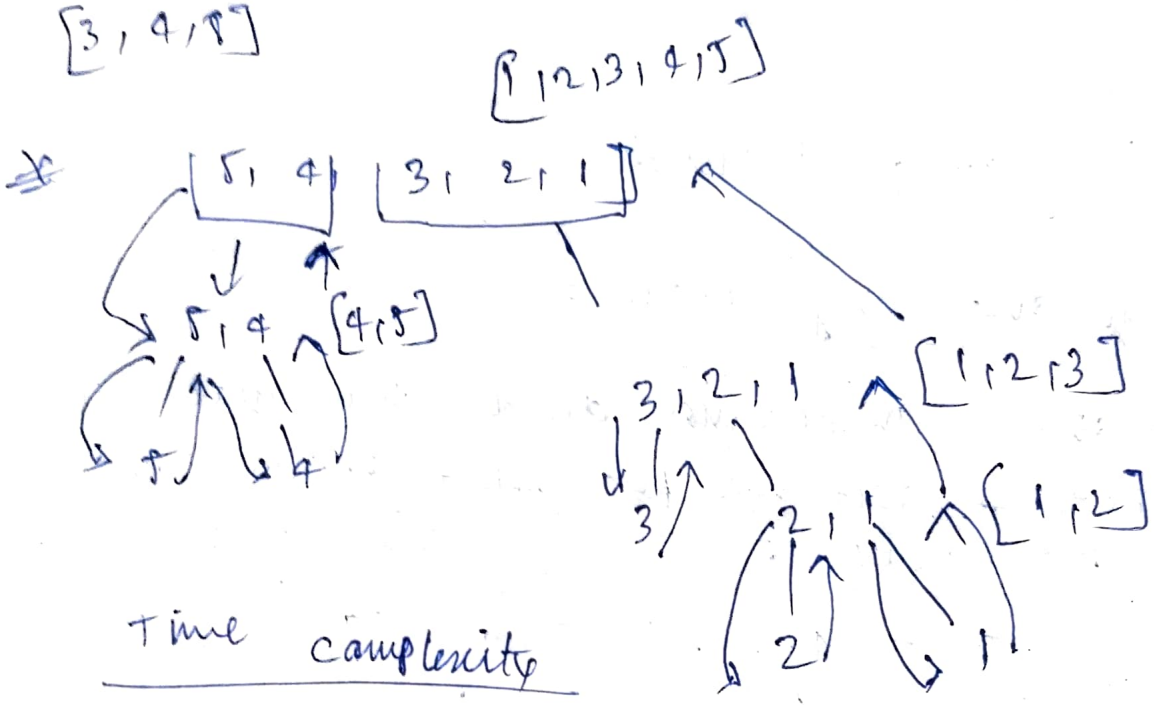
Steps

① Divide array into two parts
② Get both parts sorted via recursion.
③ Merge the sorted parts.

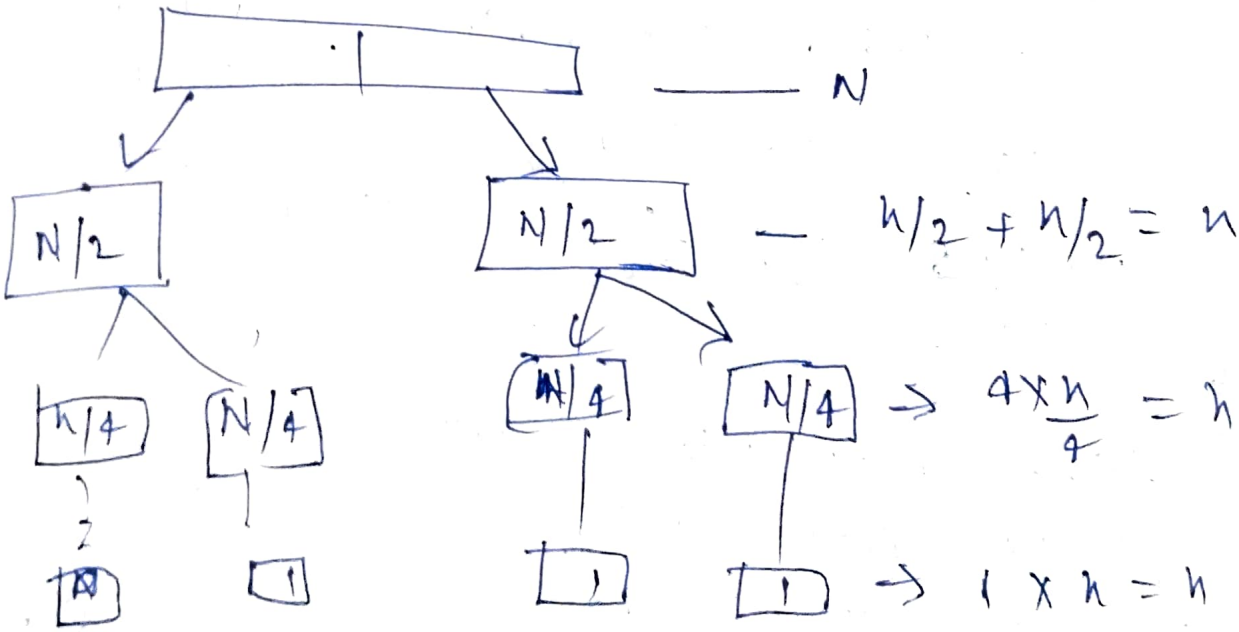
Explain point ③ arr1 = [3, 5, 9] arr2 = [4, 6, 8]

size(arr1 + arr2) = [3, 4, 5, 6, 8]

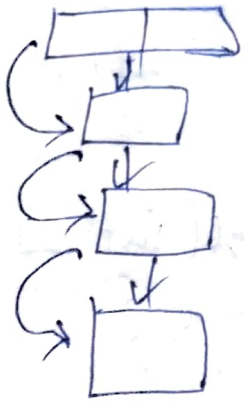




Time complexity



* At every level, N elements are merged.



$N = \frac{N}{2^0}$

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

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

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

$\Rightarrow 1 = \frac{N}{2^K} \Rightarrow 2^K = N$

$\Rightarrow K \log_2 = \log n$

$\boxed{K = \log_2 N}$ $\Rightarrow O(N + \log N)$

code

```
import java.util.Arrays;
Public class Mergesort {
Public static void main(String [] args) {
int [] arr = {4, 5, 3, 2, 1};
arr = mergesort(arr);
System.out.println(Arrays.toString(arr));
}
Static int [] mergesort(int [] arr) {
if(arr.length == 1) {
return arr;
}
int mid = arr.length/2;
int [] left = mergesort(Arrays.copyOfRange(arr, 0, mid));
int [] right = mergesort(Arrays.copyOfRange(arr, mid,
arr.length));
```

```
    return merge( left, right);
  }

Private static [] int merge( int [] first, int
                    [] second) {
int i ~~mid~~
int [] mix = new int [ first.length + second.length];
    int i = 0;
    int j = 0;
    int k = 0;
  while( i < first.length && j < second.length) {
    if( first[i] < second[j]) {
        mix[k] = first[i];
          i++;
        } else {
        mix[k] = second[j];
          j++;
        }
         k++;
        }
// It may possible that one of the arrays is
   not complete copy of the remaining elements
      while( i < first.length) {
          mix[k] = first[i]
          i++;
          k++;
      }
```

```
while (i < record.length) {
    mix[k] = record[i];
    i++;
    k++;
    } return mix;
}
```

Space complexity

Auxiliary space $\Rightarrow O(N)$

Solving complexity using Akra-Bazzi formula

$$T(N) = T(N/2) + T(N/2) + (N-1)$$

$$\Rightarrow 2T(N/2) + (N-1)$$

$$\Rightarrow 2 \times \frac{1}{2^p} = 1 \quad \Rightarrow \boxed{p = 1}$$

Time complexity $(N) = n + n\int_1^n \frac{u-1}{u^2}$

$$\Rightarrow \int_1^n \frac{1}{u} - \frac{1}{u^2}$$

$$\Rightarrow \int_1^n \frac{du}{u} - \frac{du}{u^2}$$

$$\Rightarrow \log u - \int u^{-2} du$$

$$\Rightarrow \log u + u^{-1}$$

$$\Rightarrow \left[\log u + \frac{1}{u}\right]_1^n$$

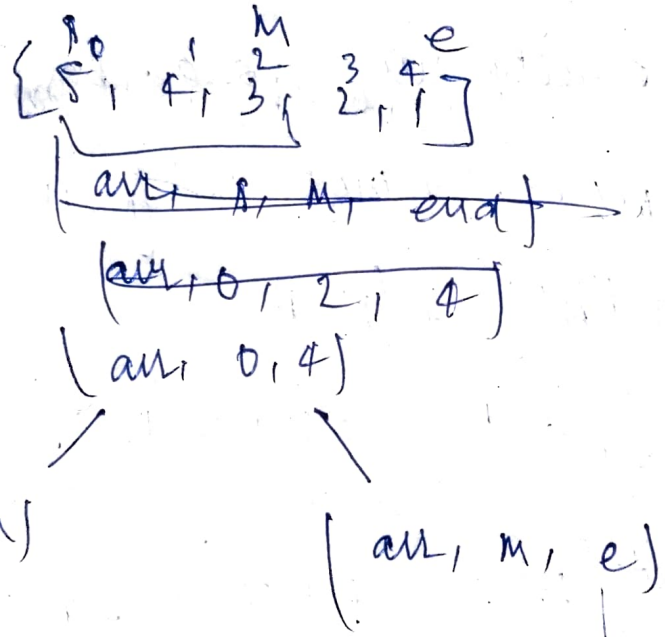
$\Rightarrow n \log n + \frac{1}{2n} - 1$

$\Rightarrow n + n\left[\log n + \frac{1}{2n} - 1\right]$

$\Rightarrow n + n\log n + 1 - 1$

$\Rightarrow O(n \log n) \Rightarrow O(n \log n)$

In-place sort



code

```
import java.util.Arrays;
Public class MergeSort {
Public static void main(String args[]) {
    int [] arr = {5,4,3,2,1};
 mergesortInplace(arr, 0, arr.length);
System.out.println(Arrays.toString(arr));
}
```

```
static void mergesortinplace(int[] arr, int s,
                             int e) {
    if(s-s==1) {
        return; }
    int mid= (s+e)/2;
    mergesortinplace(arr, s, mid);
    mergesortinplace(arr, mid, e);
    mergeinplace(arr, s, mid, e);
}
private static void mergeinplace(int[] arr,
                    int s, int m, int e) {
    int[] mix = new int[e-s];
    int i = s;
    int j = m;
    int k=0;
    while(i<m && j<e) {
        if(arr[i] < arr[j]) {
            mix[k] = arr[i];
            i++;
        } else {
            mix[k] = arr[j];
            j++;
        }
        k++;
    }
```

" it may be possible that one of the array is not complete copy the remaining elements.

```
while (i < m) {
    mix[k] = arr[i];
    i++;
    k++;
}
while( j < e) {
    mix[k] = arr[j];
    j++;
    k++;
}
for( int i=0; i < mix.length ; i++) {
    arr[s+i] = mix[i];
}}}
```

quick sort using Recursion

* Pivot ⇒ it is a reference point, it is any thing in your arrays.

Pivot :⇒ choose any elements ⇒ after first pass all the elements < Pivot will be on LHS of Pivot & element > Pivot will be at RHS. of Pivot.

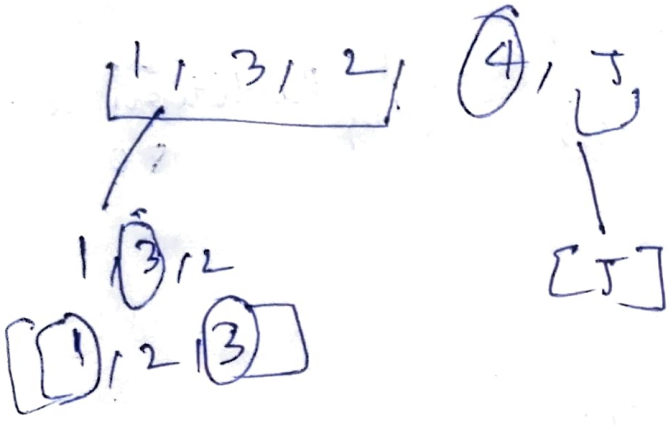
ex

5, (4), 3, 2, 1

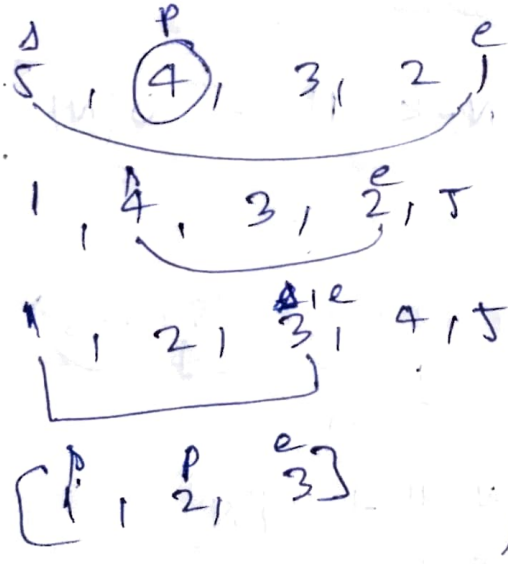
1, 3, 2, (4), 5

I have take random element after every pass all the element

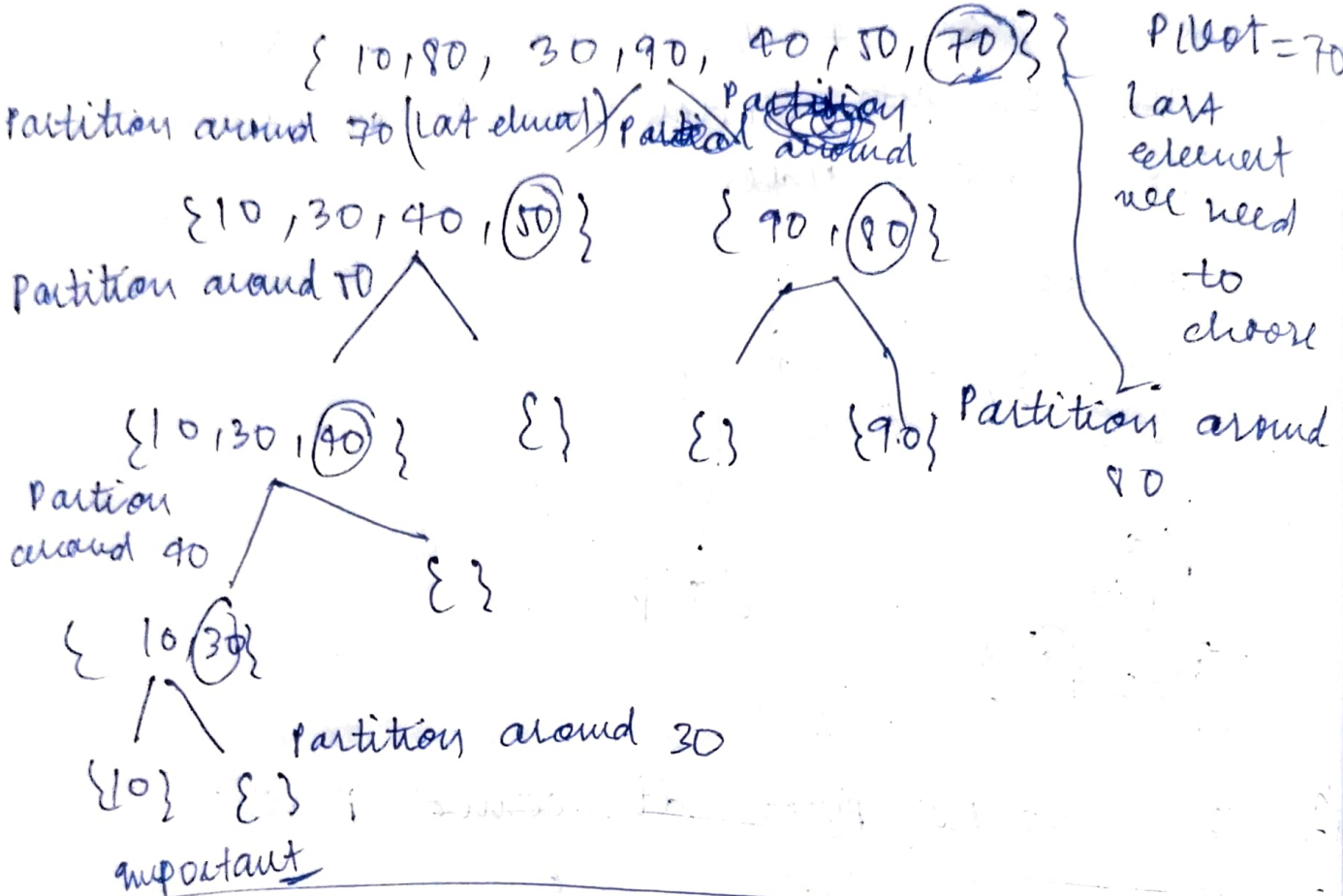
* smaller than 4 should be left side
* all the elements greater than 4 should be right hand side.



* How to put pivot at correct position.



* How to pick pivot :- * You can choose random elements
* you can pick corner elements
* pick the middle elements



important

$$T(N) = T(K) + T(N-K-1) + O(N)$$

Worst case :-

3, 5, 8, 9, 20, 34, 19, 66 Pivot

(N-1)

$$T(N) = T(0) + T(N-1) + O(N)$$

$$T(N) = T(N-1) + O(N)$$

$\Rightarrow$

Best case :-

$$T(N) = T(N/2) + T(N/2) + O(N)$$

$$T(N) = 2T(N/2) + O(N)$$

(Hybrid sorting algorithm) tim sort

merge sort + insertion sort ⇒ works well with partially sorted data.

code

```
import java.util.Arrays;
Public class QuickSort {
 Public static void main (String [] args) {
 int [] arr = { 5, 4, 3, 2, 1};
 // sort(arr, 0, arr.length-1);
 sout( Arrays.toString(arr));
    Arrays.sort(arr);
 }
 Static void sort( int [] nums, int low, int hi) {
      if( low >= hi) {
         return;
       }
      int s = low;
      int e = hi;
      int m = s + (e-s)/2;
      int pivot = nums[m];
      while ( s <= e) {
// also a reason why if its already sorted it will
not swap  while ( nums[s] < pivot) {
                s++;
      }
      while ( nums[e] > pivot) {
            e--; }
```

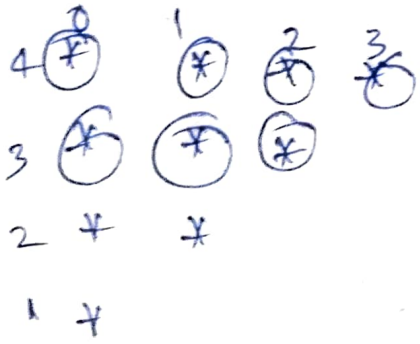
```
if(s<=e){
int temp = nums[s];
 nums[s] = nums[e];
 nums[e] = temp;
  s++;
  e--;
});
```

// now mid pivot is at correct index, please sort two halves now.

```
sort(nums, low, e);
 sort(nums, s, hi);
}
}
```

Recursion - Pattern questions + Bubble sort + Selection sort.

---

① 

```
fun(row, col)
  (4,0)
    ↓
  (4,1)
    ↓
  (4,2)
    ↓
  (4,3)
    ↓
 new line
  (3,0)
  (3,1)
  (3,2)
```

code

```
import java.util.Arrays;
Public class Triangle{
 Public static void main(String[] args){
     triangle2(4,0);
  }
static void triangle2(int r, int c){
      if(r==0){ return }
       if(c<r){
         triangle2(r,c+1);
        System.out("*");
      }else{
       triangle2(r-1,0);
       System.out();
    }}
static void triangle(int r, int c){
     if(r==0){ return; }
    if(c<r){
    System.out("*");
    triangle(r,c+1);
   }else{
     System.out("");
      triangle(r-1,0);
  }}
```

```
import java.util.Package;
Public class Sorting {
Public Static void main( String [] args) {
int [] all = [1, 4, 3, 7 ] ;
Selection(all, all.length ,0,0) ;
Sout(Arrays.toString(all)) ;
}
Static void bubble (int [] all, int n, int c) {
    if(n==0) { return ; }
    if(c<n) {
     if(all[c] > all[c+1]){
       // swapp
       int temp = all[c];
       all[c] = all[c+1];
       all[c+1] = temp;
     }
     bubble(all, n, c+1) ;
     } else {
       bubble (all, n-1,0) ;
    }}
Static void Selection (int [] all, int n, int c,
    int max) {
    if(n==0) { return; }
    if( c<n) {
     if(all[c] > all[max]) {
```

```
selection(arr, n, c+1, c);
} else {
selection(arr, n, c+1, max);
} } else {
int temp = arr[max];
arr[max] = arr[n-1];
arr[n-1] = temp;
selection(arr, n-1, 0, 0);
}}
}
```