* Linear Search

- Tip: If you want to change name of all 'x' in code file to 'y' → R.Click on x → refractor → rename → 'y'.
- Unlike binary search, this can be applied for unsorted array also.
- Worst case Time complexity → O(N) N: size of array
  Best " " " → O(1)
- Pseudo Code =>

```
psvm{
int[] nums = {...};
~~int target =~~
} sout(search(nums, 23));
static int search(int[] arr, int target){
    if(arr.length==0){
        return -1;
    }
    for(int i=0; i<arr.length; i++){
        if(target == arr[i]){
            return i;
        }}
    return -1; → if no. do not exist.
}
```

  - ↳ use it for range of index also → you know how
  - ↳ use it for checking even in an array.
  - ↳ " " " " " even no. of digit number in an array.
  - ↳ also use for finding smallest value in arrary.
    - ↳ first put first index value as min. & run the loop if any other element found smaller than index 0 → assign min to it.

- For a 2D Array =>
  Pseudo code =>

```
psvm{
int[][] nums = {{32...3},{2...3},...{3};
sout(search(nums, 44));

static int[] search(int[][] arr, int target){
```

```
if(arr.length == 0){
    return new int[]{-1,-1};
}

for(int row=0; row<arr.length; row++){
    for(int col=0; col<arr[0].length; col++){
        if(target == arr[row][col]){
            return new int[]{row, col};
        }
    }
}
return new int[]{-1,-1};
```

* Binary Search

- for sorted array only.
- Concept ⇒
  1. Find the middle element.
  2. target > mid ⇒ search in right
     else search in left.
  3. if middle element == target //ans.

- complexity ⇒ O(log N).
- better way to find mid :-

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

$$m = s + \frac{(e-s)}{2}$$; → both are same just the diff. of not exceeding the range of int.

- rest same as B.S. ⇒

```
static int search(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;
        }
```

~~star~~ else if (~~start~~ target > arr[mid]) {

start = mid + 1;

} else

return mid;

return -1;

}

- Order Agnostic → for ascending or descending sorted.
  ∴ use if-else condition & change start or end for descending respectively by observing.

Questions:-

Q.1 Ceiling of a number.

Ceiling = smallest element in array ≥ target.

arr = [2, 3, 5, 9, 14, 16, 18], target = 15

∴ ceiling = 16.

→ Sol:- when loop breaks (start <= end) → return start → Ans (Sochna) (instead of return -1)

Q.2 Floor of a no.

Floor = greatest no. in array ≤ target.

∴ floor = 14.

→ Sol:- when loop breaks (") → return end → Ans (Sochna) (instead of return -1)

* Binary Search in Matrix

⇒ Matrix must be sorted in row wise and column wise manner.

10	20	30	40
15	25	35	45
28	29	37	49
33	34	38	50

[target = 37]

→ "not strictly sorted"

"40"

⇒ check first ~~co~~ row → lower bound = 10, upper bound = 40 (4th column)

Note:- if target < upper bound then the element is also not in that particular column ∴ upper bound → [col--;]

⇒ Now if no. is greater than 30 (after col-- upperbound=3rd col) then no. must be in that column only ! + [(row++;)]

since 10, 20 removed

⇒ again 35 → < target ∴ row++

⇒ 37 = target ∴ ans.

⇒ Hence, ① element == target → ans. ② element < target → row++ ③ element > target → [col--]

⇒ In ~~best~~ linear search of matrix → complexity was O(N²).

(worst case) Here :- row is moving from 0 to last (n time), and column is decreasing by 1 (n times) from n-1.

∴ n + n = 2n

∴ O(N) Ans. B.S. in 2D matrix.

⇒ Pseudo code ⇒ in search method ↓ Do not forget to use ~~Array.to~~ String in sout in main func.

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

Q. Search in a sorted matrix.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

- (1) If element == target //ans
- (2) " " > " //ignore rows after it
- (3) " " < target // ignore above rows

- start → lowerbound → row start → 1st row
  end → upper " → row end → last row

- In the end 2 rows are remaining

1	2	3	4
5	6	7	8

target = 3

(1) check whether the mid col. you are at centering ans. → (2,6) → mid col.
(2) Consider the 4 parts
(3) check B.S. in all 4 parts

Complexity :- 1. To find these 2 rows ~~the~~ size of col. will be → log(N)
2. searching across column → log(M)

$$\therefore O(\log(N) + \log(M))$$

→ understand code from video then note.

Date 16/07/24
Page

* **Bubble Sort or Sinking sort or Exchange sort**
  ↳ "comparison sort method"

In every step you have to compare adjacent elements.

1st Pass

eg. ⇒ 3 1 5 4 2

1 3 5 4 2 (No)

1 3 4 5 2

1 3 4 2 5 ↳ reached end

With the first pass through the array, the largest element came to the end.

2nd Pass

1 3 4 2 5 (No, No, Yes)

1 3 2 4 5 ∴ no need to compare these

With pass no. 2, 2nd largest element is at 2nd from last index.

3rd Pass

1 3 2 4 5 (No, Yes)

1 2 3 4 5

i = counter (outer loop)

Concept ⇒

[For i=0 / 1st pass]

i j
3 1 5 4 2
1 3 5 4 2
1 3 4 5 2
1 3 4 2 5

internal loop → checking is $j < i-1$ ∴ swap else move forward. (N−1) times

∵ 3 is not gonna check itself.

For i=1 second pass

1 3 4 2 5
1 3 2 4 5

∴ j will only move till that part where array is not sorted.

(no need to compare them)

i.e. in 3rd pass j will only check 1, 3, 2

For i=2 third pass

1 3 2 4 5
1 2 3 4 5

Hence, j is moving till $< (length - i)$ or $<= len-i-1$

Space complexity = O(1) // constant // No extra space req.

aka inplace sorting algo. { ie. copying the array etc. not required.

Time Complexity: (definition (time))
Best case : O(N) ⇒ sorted
Worst case : O(N²) ⇒ sorted in opp.

As the size of array growing → the no. of comparison also grows.

• Best case:

1	2	3	4	5

i=0 → ith pass

Note:- When j never swaps for a value of i, it means array is sorted.

∴ Best case comparison = O((N-1) × 1)
= O(N).

• Worst case

reverse ⇒	5	4	3	2	1	j
i=0	4	3	2	1	5	N-1
i=1 2nd pass	3	2	1	4	5	N-2
i=2 3rd pass	2	1	3	4	5	N-3
~~i=3 4th pass~~	1	2	3	4	5	N-4

N-i-1

Total comparisons = (N-1) + (N-2) + (N-3) + (N-4)

∴ (Generalised) = 4N - (1 + 2 + 3 + 4 + ... n)

= 4N - (N*(N+1)/2)

= O((7N - N²)/2) = O(N²) Ans

Stable and Unstable Algorithms

Stable = order should be same when value is same.

Unstable = vice-versa.

e.g ⇒ 10 20 20 30 10

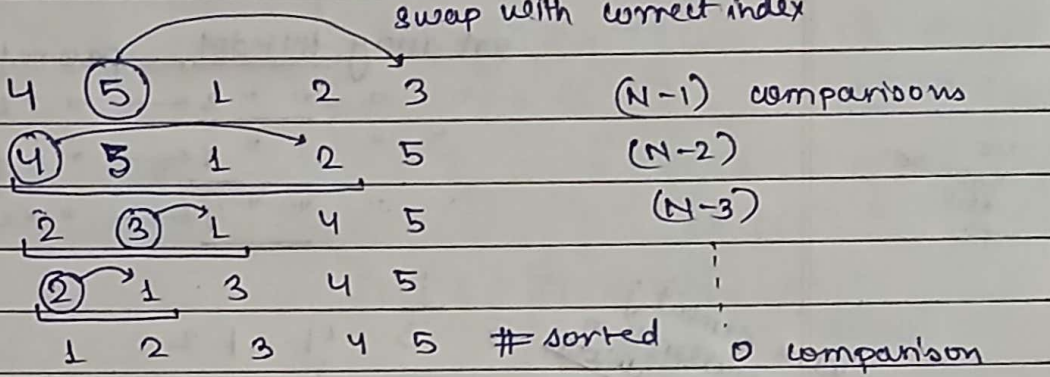
↓ sort

10 10 20 20 30

In original array black 10 was after blue 10 and
in sorted " " " " also " " " " . ∴ stable.

but if, 10 10 20 20 30 → ∴ unstable

Selection Sort



Ily, we can do for smallest no.

Ily, loop running → length-i

Total comparisons = 0 + 1 + 2 + 3 + ... + (n-1)

$= \frac{(n-1)(n)}{2} = O(N^2)$.

- Worst case = $O(N^2)$
- Best case = $O(N^2)$
- unstable algorithm
- use case: performs well on small lists/arrays.

16/06/24

best, easy

Insertion Sort

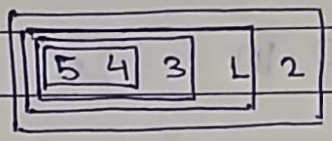
→ Sort while increasing index

5 3 4 1 2

sort till index 1
sort " " 2
" " " 3
" " " 4

⇒ For every index you're at : Put that index element at the correct index of LHS.

⇒ Sort box wise ⇒ (insertion sort)



i → (<N-2) j → (>0)

sort array till index ← para no. ← 0 — 1 (i+1)
" " " " 2 ← " " 2 ← 1 — 2 ∵ 0th wala i.e. 1st no. ko kya hi sort krna
" " " " 3 ← " " 3 ← 2 — 3
" " " " 4 ← " " 4 ← 3 — 4

~~4/5~~ ~~4 3 5 → 3 4 5~~ ~~4 3 5~~ ~~4 3~~

~~4 → not needed~~

i will run from 0 to (N-2)

also reason if i=4, j=5 no 5th index

when element j is not smaller than element j-1, break loop

5 3 4 1 2
3 5 4 1 2
3 4 5 1 2

Reason

Bcoz LHS array is already sorted

3 4 5 1 2
3 4 1 5 2
3 1 4 5 2
1 3 4 5 2

condition (j>0) ∴ break

for i=2, till index 3 its sorted

1 3 4 5 2
1 3 4 2 5
1 3 2 4 5
1 2 3 4 5 # sorted

⇒ Worst case complexity : $O(N^2)$

⇒ Best " " : $O(N)$

⇒ use case : Adaptive i.e. steps get reduced if array is sorted.
no. of swaps reduce as comparison to Bubble sort

⇒ Stable Algo

⇒ Used for smaller values of N ⇒ works good when array is partially sorted + it takes part in hybrid sorting algorithms (e.g. quick sort, merge sort, heapsort then insertion)

* Recursion LvL-1 Ques.

Sum Digit → e.g. n = 1342 → 1+3+4+2 = 10 (Ans.)

2 + f(134) → N/10
N%10 → 4 + f(13)
3 + f(1)
1 + f(0)

stack pushing

2 + 8 = 10
4 + 4 = 8
3 + 1 = 4

popping stack

f(1) = 1 ⇐ [f(0) = 0]
Base cond^n

Similarly Prod Digit. n = 1342 → 1×3×4×2 = 24

2 × f(134)
4 × f(13)
3 × f(1)
1 × f(0)

Base condition [f(0) = 1]

2342
2 × f(234)
4 × f(23)
3 × f(2)
2 × f(0)

Alternate base condition → (n%10 == n
return n;)

e.g. n = 505
5 × f(50)
0 × f(5)

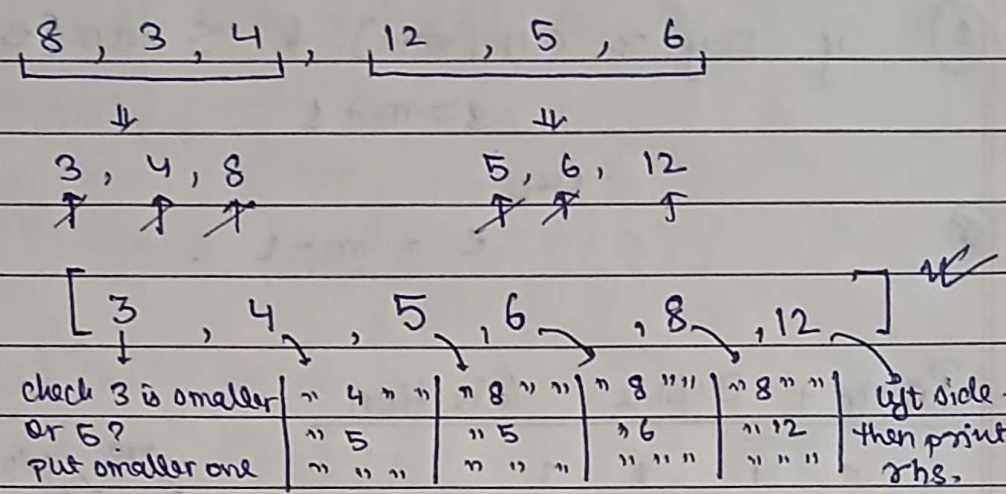
last pattern Q, just simple diff. is that rather print just check & swap.

- Q, check code for selection sort.

18/07/24

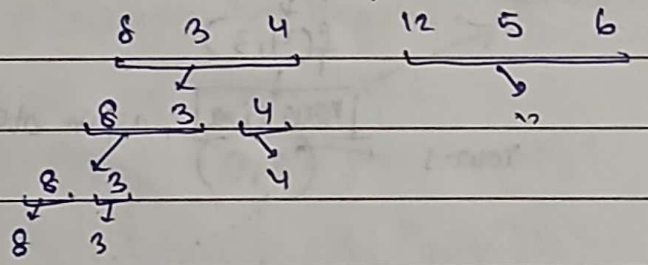
* MERGE SORT

→ ~~Merge~~ sort LHS and RHS of array middle
→ merge them by checking individually.



Steps :-

1. Divide array into 2 parts
2. Get both parts sorted via recursion
3. merge the sorted part.



Q. what Arrays.copyOfRange( ) does?

→ sol: 1 ctrl + click to check

sol 2 copies the specified range into a new array.

→ check code!. + 4 mostly check for merge in plain code!

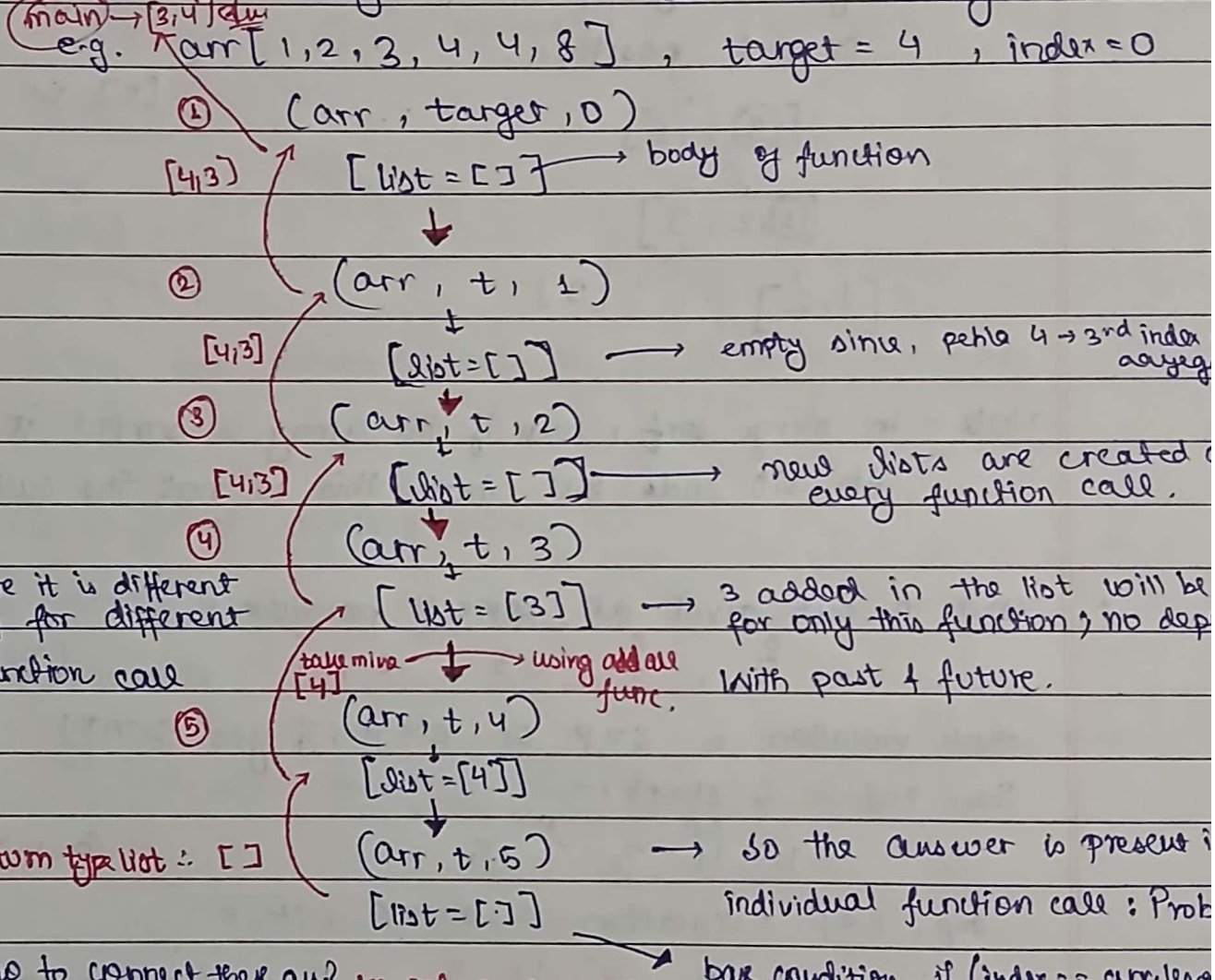
---

*2. (Continuing return ArrayList) Passing it in body (not Argument

↳ very Imp.

Goal: return the list but don't take in argument.

∴ Problem: Every call will have new arrayList.



∵ here it is different obj for different function call

→ So the answer is present i individual function call : Prob

How to connect these ans? see red pen answers.

base condition if (index == arr.leng return list }

CHECK CODE!

* Quick Sort

- Pivot: choose any element → after first pass
  all the elements < pivot will be on LHS of pivot
  and " " " > " " " " RHS " " .

eg. 5, (4), 3, 2, 1 , let pivot = 4

↓

3, 2, 1, (4), 5

Note:- after every pass pivot will be surely at correct position.

[5] ✓

[(3), 2, 1]

[(1), 2, 3]

[1, 2] [3]

Note:- in merge sort, even if the array is sorted it goes till ends but here this is not the case.

- How to put pivot at correct position?

s p e
5, 4, 3, 2, 1 (p=4)

check violation is s>p or e<p, if yes SWAP!
then end-- & start++

∴ 1 s,p 4 3 2 p 5 (p=4)

s=p, e<p ∴ violation ∴ swap, s++, e--

1 2 s,e 3 4 5, ∴ pivot got correct pos.

low

no need to swap if both same index

high

s p e
[1, 2, 3]

s,e

e s

ends ends

[1, 2, 3]

(low, end) (start, high)

[s & e] → for swapping

[low & high]

main

tells us which part of array you're working on.

- How to pick pivot?

→ taking pivot at last element:

{10, 80, 30, 90, 40, 50, (70)}

partition around 70

{10, 30, 40, (50)} {90, (80)}

{10, 30, (40)} {} {} {90}

{10, (30)} {} {}

{10} {}

So, which one to pick at last? :- 1. Random element

2. Corner "

3. middle "

How to select which one? :-

→ Complexity analysis of pivot pos^n.

let N elements :. [k] [p] [n-k-1]

∴ $T(N) = T(K) + T(n-K-1) + O(N)$ (for pivot changing at every pass)

recurrence rel^n of Quick sort.

- worst case :- when pivot element is either smallest or largest in array.

when K=0.

eg. 3, 5, 8, 9, 20, 34, 18, (66)

(n-1) calls rather dividing into 2.

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

$\therefore$ $\boxed{T(N) = T(N-1) + O(N)}$

complexity for this $\rightarrow$ $= O(N^2)$ $\rightarrow$ check Space & Time Comples. notes for derivation.

- Best case : when pivot is middle element so all divided into 2.

$K = \frac{N}{2}$ $\quad \therefore \quad T(N) = T\left(\frac{N}{2}\right) + T\left(\frac{N}{2}\right) + O(N)$

$$\boxed{T(N) = 2T\left(\frac{N}{2}\right) + O(N)}$$

(r.r. relation of merge sort)

Best case complexity : $O(N \log N)$

- Imp. points :-

→ not stable sorting algorithm

→ In-place, thats why preferred for arrays instead of merge sort, ∵ M.S. takes $O(N)$ extra space for copying.

→ M.S. is better in L.L. due to not conts. memory allocation.

- Hybrid sorting algorithms (Tim Sort) :-

⇒ Merge sort + Insertion sort

∵ it works well with partially sorted data