Sorting

① Selection Sort - finds the smallest element & puts it to first place of the array continues the same till sorted

1- get an input array

2- use a for loop an outer for loop mainly for Swapping the min element in the array to its right place ie to ith place everytime (start of every round of outerloop

ex: i=0- arr[0]= 1st min
arr[1] = 2nd min
arr[2]= 3rd min
etc

3- keep a variable to store the index of the min'e elements position initially at i assuming that ith is the Smallest in every iteration.

4- now in the inner for loop compare every element in the array with mini
if you find a smaller element then update mini with that index

5- now if(mini!=i) that ~~that~~ means if mini=i then the smallest element is in the right place but if mini!=i i.e theres a smaller element in the array

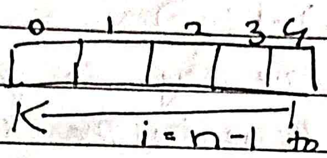
so just swap
(arr[i], arr[mini])

__/__/____

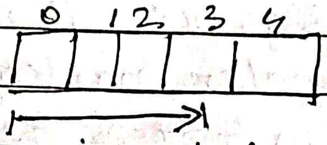
② Bubble sort

1- first you get an input array then run two loop on the array ~~[illegible]~~ outer & inner.

ex:



i = n-1 to 0 outer



j = 0 to i-1 inner

↓

Cuz to compare j with j+1
when j=3 j+1=4

you compare j with j+1 if j > j+1 then swap the two

- basically yore bubbling the largest element to the last of the array in every iteration.
- moving it to the last by pushing it one step ahead everytime

③ <u>Insertion sort</u>

- Basically you start the comparision from 1stelement (i.e i = 1-n) with all the elements before it which is (j = i-1 to 0)
- ith element is the key so you start comparing the jth elements with the key
- if jth element is greater it moves ahead
- if jth element is smaller than key it ~~places~~ then the key is placed in front of the smaller element
- in one whole complete j loop the key remains same only when i increments key changes

ex:

0	1	2
2	3	1

i = 1   j = 0   key = 3

↓ no change in loop

2	3	1
0	1	2

j + 1 = key
0 + 1 = 3

1 < 3 ✓

0	1	2
2	3	1

i = 2   j = 1   key = 1

↓

1 < 2 ✓

2		3
0	1	2

i = 2   j = 0   key = 1

↓

1	2	3
0	1	2

- Looks for larger numbers & then pushes it ahead & then places the key such that if x is the key all elements before x will be smaller than x.

4 Quicksort

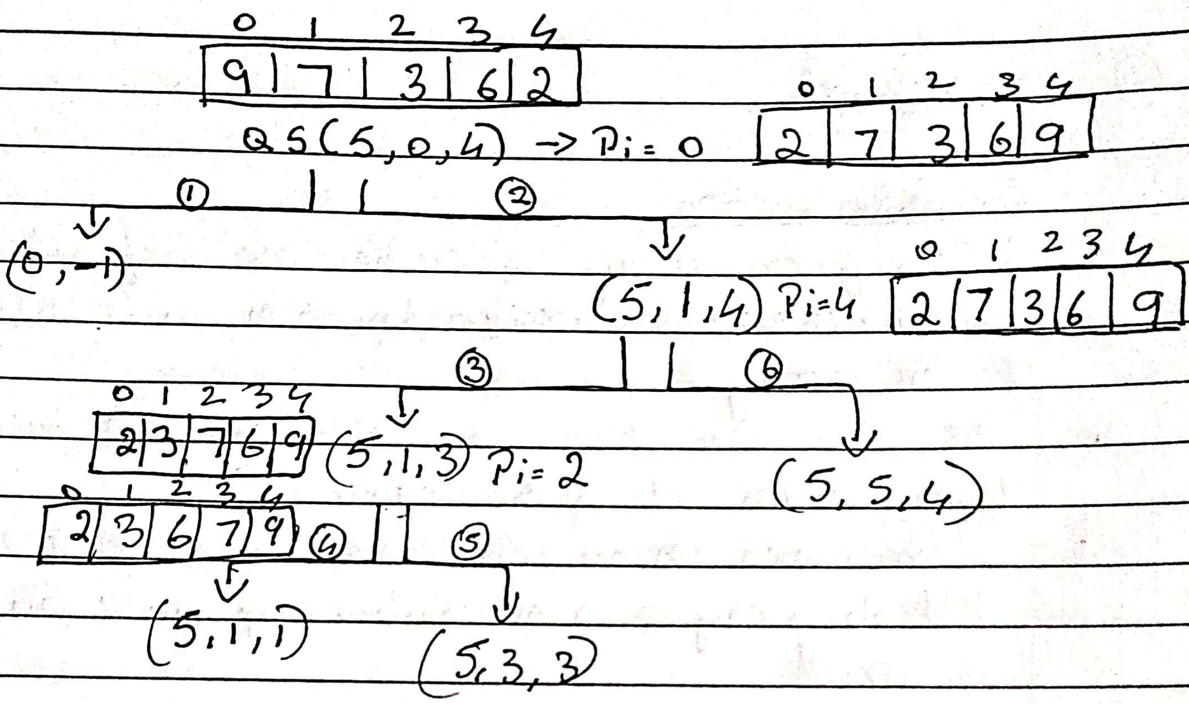
Recursive function

- for a given array from the main function call the quicksort recursive function with start & end of the array
- The recursive function calls partition function from which it gets P (Pindex)
- Then based on the Pindex divide the arrays into leftsub array & right subarray until the array is sorted.

Partition function

- it is called from the quicksort function with start & end
- arr[end] is kept as the Pivot
- Pindex is kept as a tracker to ~~p~~ fill smaller elements in the array to the start of the array & position the pivot in its right place (Pindex = start)
- here all the elements are compared with the pivot if the element is smaller it is shifted to begining & Pindex increases (for whole array)
- then for last updated Pindex position the Pivot is placed where the smaller elements than Pivot will be on its left.

looks
~~compares~~ smaller elements & starts filling from start & then places the pivot


0 1 2 3 4
9 | 7 | 3 | 6 | 2
QS(5,0,4) -> Pi = 0
0 1 2 3 4
2 | 7 | 3 | 6 | 9
①
②
(0,-1)
(5,1,4) Pi=4
0 1 2 3 4
2 | 7 | 3 | 6 | 9
③
⑥
0 1 2 3 4
2 | 3 | 7 | 6 | 9
(5,1,3) Pi= 2
(5,5,4)
0 1 2 3 4
2 | 3 | 6 | 7 | 9
④
⑤
(5,1,1)
(5,3,3)

__/__/__

⑤ Merge function

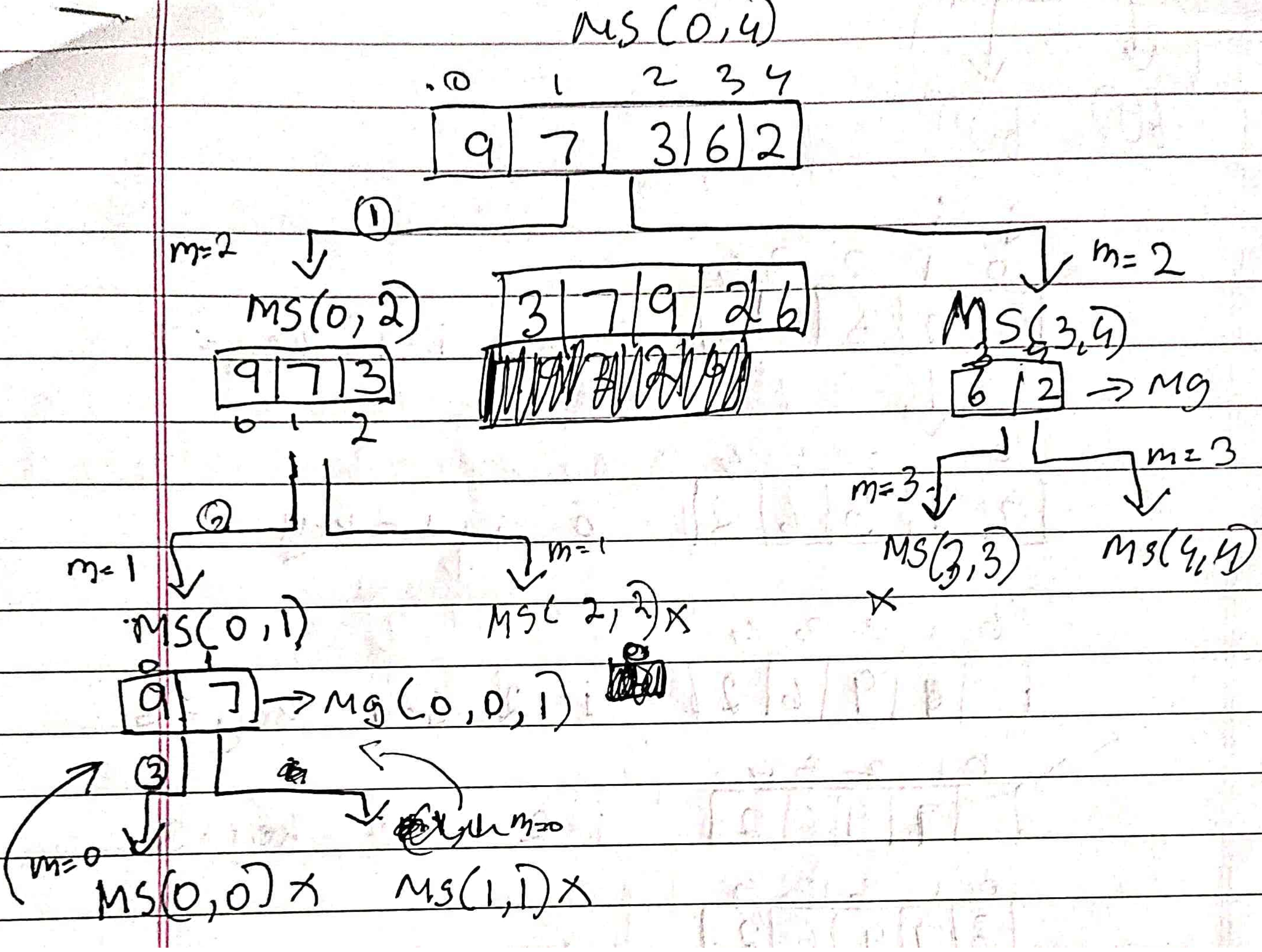
Recursive function

- It takes the whole array from main
- gets the mid point of the array
- divides the array into left & right subarray based on midpoint (left - start to mid)(right ~~&~~ - mid+1 to end)
- and for each subarray merge function is called.

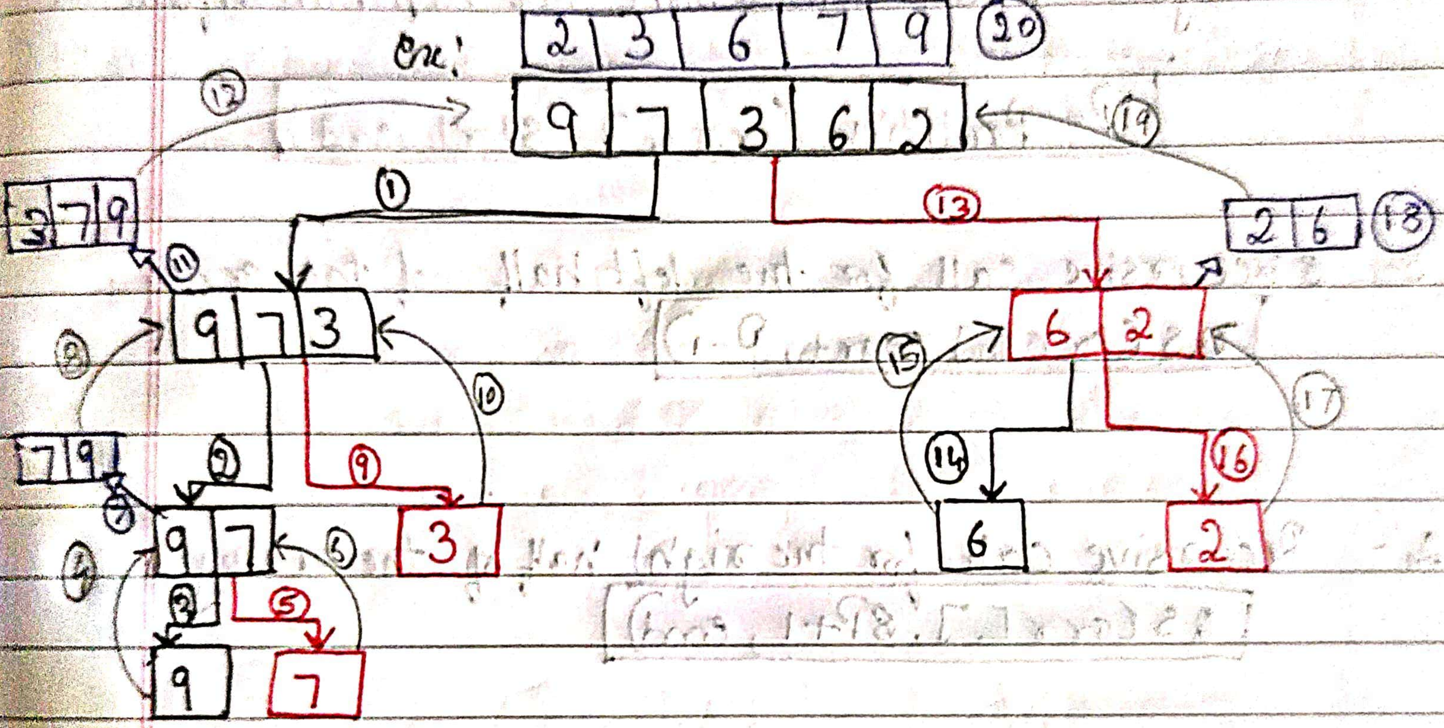
Merge funtion

- The merge function gets two subarrays and compares the two subarrays
- but even if it just gets one of the subarray right or left it treats that single subarray itself as two smaller subarrays like first to mid & mid+1 to end in a single subarray.
- now it compares the two parts and puts the smaller elements in ~~&~~ the beginning of the temp array and then the remaing larger elements
- now after temp array is filled & sorted it is now copied back to original array.

:- looks for smaller elements & then copies those small elements in temparray



MS(0,4)
0 1 2 3 4
9 7 3 6 2
1
m=2
MS(0,2)
9 7 3
0 1 2
3 7 9 2 6
m=2
MS(3,4)
6 2 → Mg
m=3
m=3
MS(3,3)
×
MS(4,4)
2
m=1
m=1
MS(0,1)
MS(2,2) ×
9 7 → Mg(0,0,1)
3
m=0
m=0
MS(0,0) ×
MS(1,1) ×



Black - first recursive call
Pencil - returning
red - second recursive call
blue - Sorted part of array