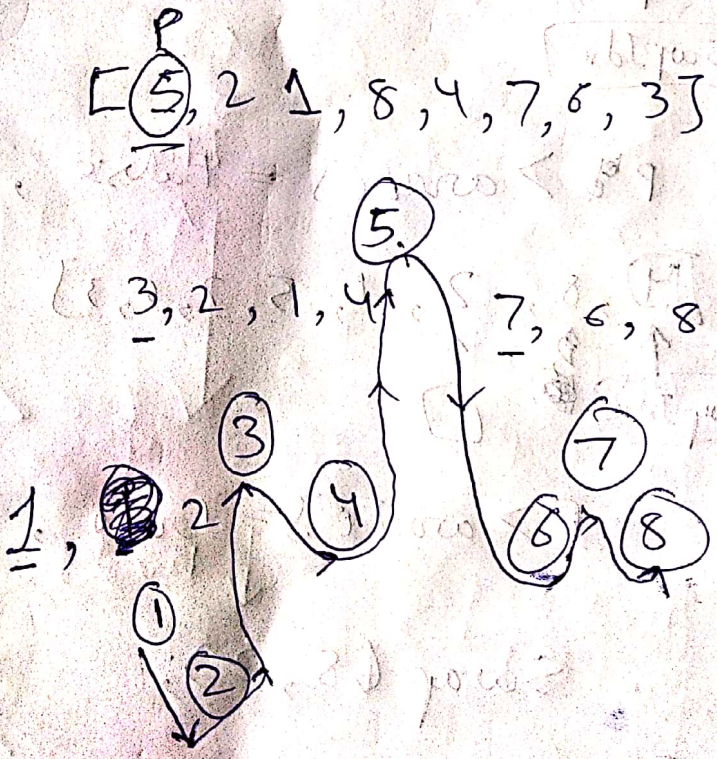
Quick Sort

Ex:- arr = [5, 2, 1, 8, 4, 7, 6, 3]

Working:-

Pivot Element : Any random element from the element we'll take. But for the sake of convenience we'll take the first element of the array.

Then move all the elements less than the Pivot element to its Left Hand side & all the elements greater than the Pivot element to its Right Hand side.



Scanned with CamScanner

Pivot Helper Function

~~function pivot (arr, startIdx = 0)~~

Steps:- In arr:- [4, 8, 2, 1, 5, 7, 6, 3]

(P)

Output:- [2, 1, 3, 4, 5, 7, 6, 8]

<P (Corder (doesn't matter))

(P) with Indx=3

>P (Corder doesn't matter.)

(i) [4, 8, 2, 1, 5, 7, 6, 3]

P, i, SwapIdx

P > arr[i] = False, i++

(ii) [4, 8, 2, 1, 5, 7, 6, 3]

SwapIdx, 8, i

P > arr[i] = True, SwapIdx++, i++

Swap (8, 2);

(iii) [4, 2, 8, 1, 5, 7, 6, 3]

SwapIdx, i

P > arr[i] = True, SwapIdx++, i++

Swap(arr(swapIdx, arr[i]) = Swap (8, 1)

Scanned with CamScanner

(iv) [4, 2, 1, 8, 5, 7, 6, 3]

~~P~~ swapIdx i

P > arr[i] = false, i++

(v) [4, 2, 1, 8, 5, 7, 6, 3]

swapIdx i

(vi) [4, 2, 1, 8, 5, 7, 6, 3]

swapIdx i

(vii) [4, 2, 1, 8, 5, 7, 6, 3]

swapIdx i

P > arr[i] = true, swapIdx++, i++

swap(arr(swapIdx), arr(i)) = swap(8, 3)

(viii) [4, 2, 1, 3, 5, 7, 8]

P swapIdx

swap(P, arr[swapIdx]) = swap(4, 3)

(ix) [3, 2, 1, 4, 5, 7, 8] ← output

P, swapIdx

swapIdx = 3 ∴ 4 is at the sorted Position of the array.

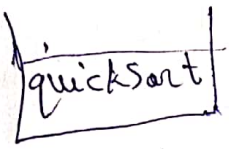
Scanned with CamScanner

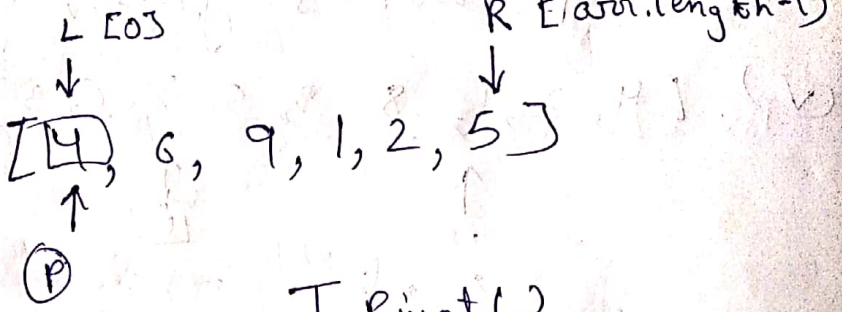
Quick Sort Implementation

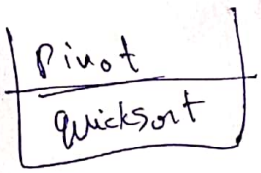
Ex:- arr [4, 6, 9, 1, 2, 5];

Code Implementation Visualisation :-

Stack

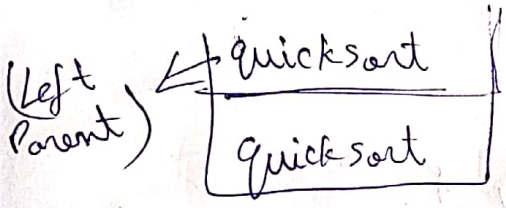


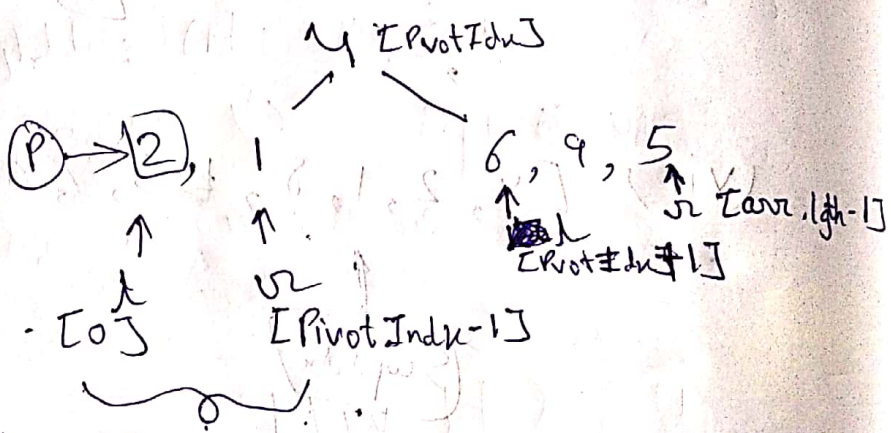


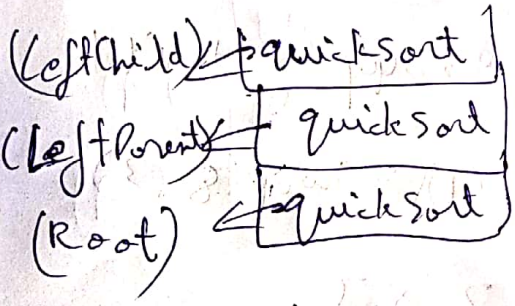


Pivot()

[2, 1, 4, 6, 9, 5]





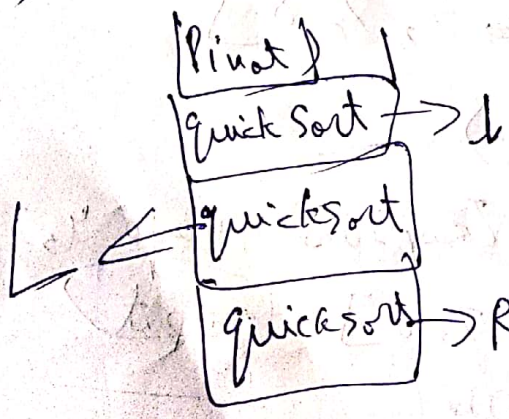


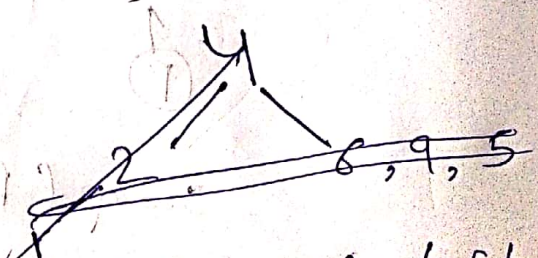
quicksort() → (leftchild)

Since, L=0 < R = PivotIdx-1 = 1

So,

Pivot()

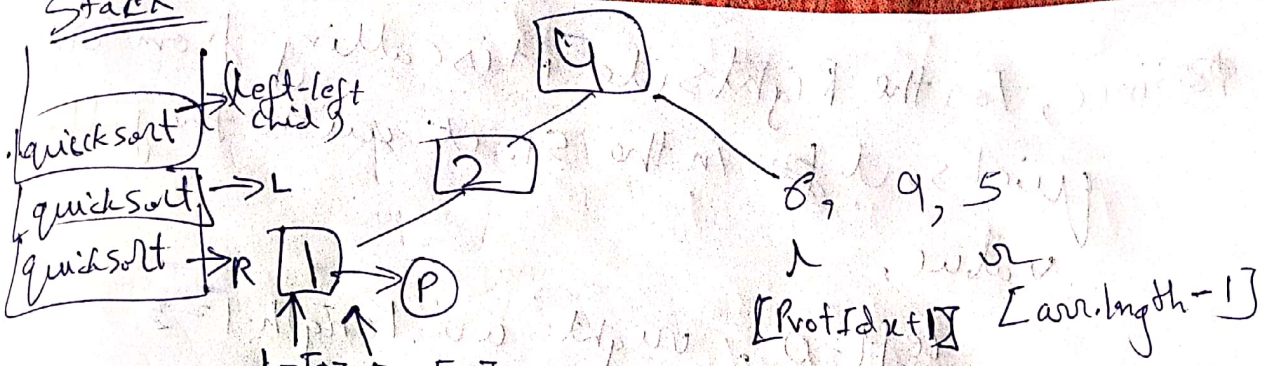




After pivot() to leftchild

Scanned with CamScanner

Stack



quicksort() → (left-left Child

Since, L=0 & r=0 → ~~(L<r=False)~~

Hence, return arr → (Base case)

[1, 2, 4, 6, 9, 5] to the previous quicksort which is waiting in the stack

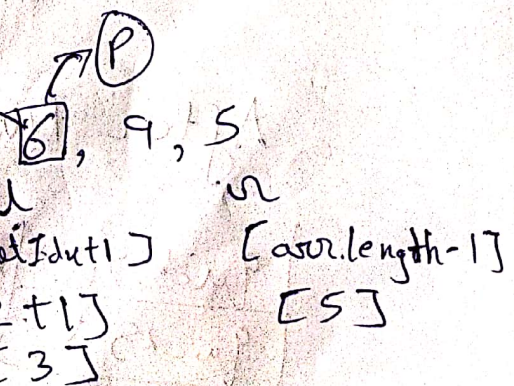
i.e. return back to the Left Parent Quick Sort.

Now, [1, 2, 4, 6, 9, 5]

Left is Sorted

Note:- Root/Main/Original Quick Sort is still waiting in the Stack.

Now, its time for the ~~Left~~ Right Parent element which would call from root quicksort.

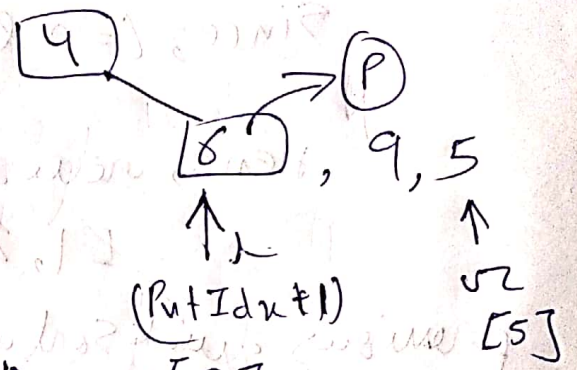


Scanned with CamScanner

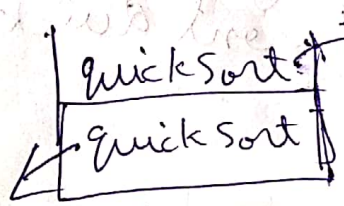
Since, For the Right Side it is calling from root quicksort in the Stack with the following value,

left = 0, right = arr.length-1 = 5

PivotIdx = 2



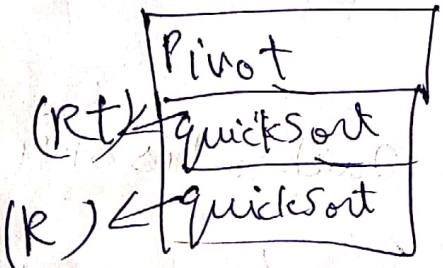
Stack



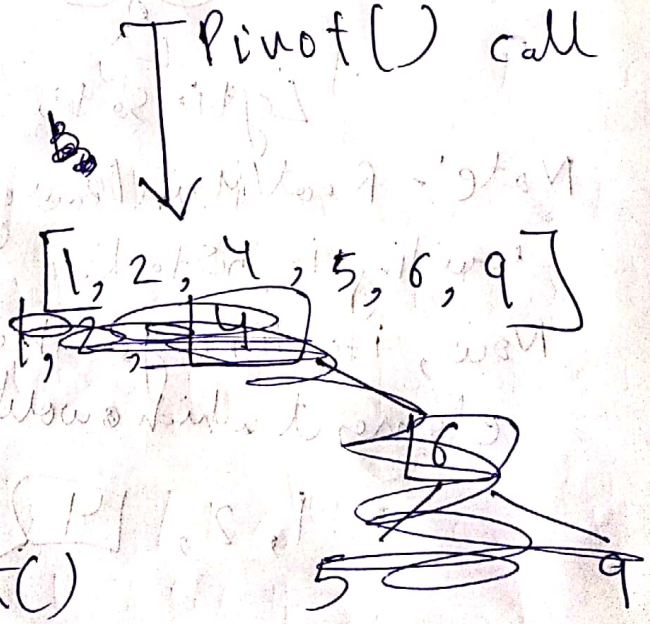
quickSort() → left = 3, vr = 5

Since, l > vr = True

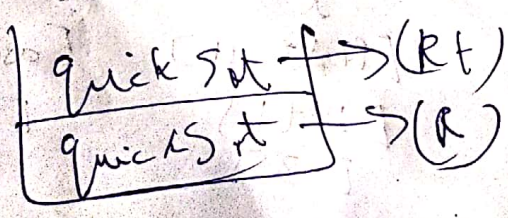
So,



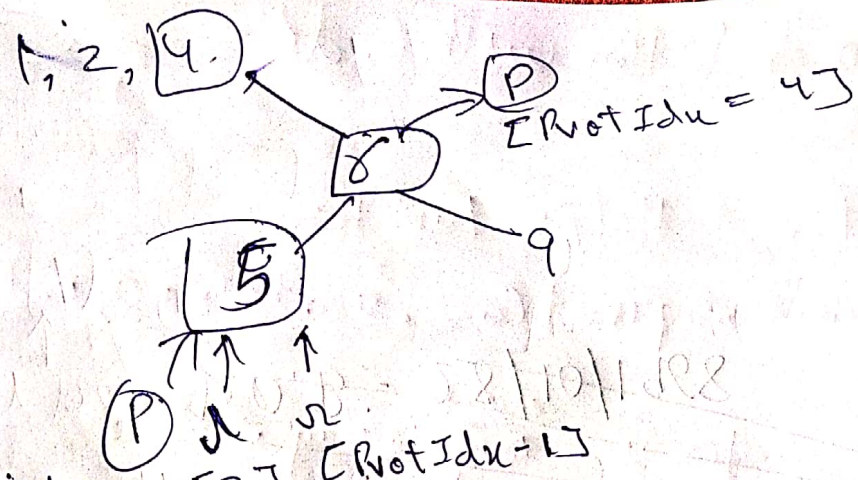
Pivot() call

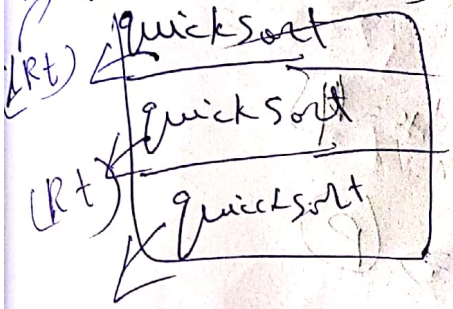


After Pivot()



Scanned with CamScanner





quicksort() → l=3, r=3

Since, l<r = False

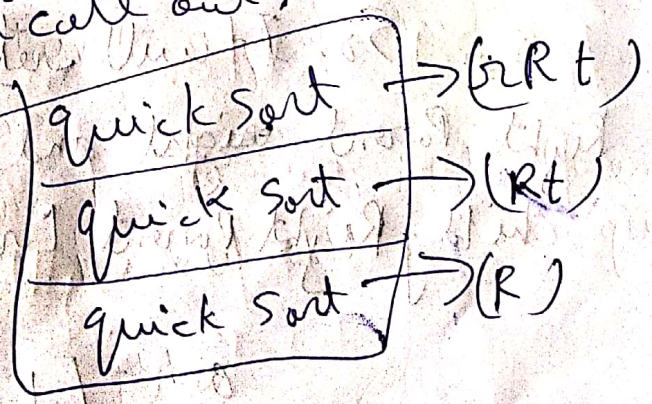
we'll return an arr :-

arr = [1, 2, 4, 5, 8, 9]

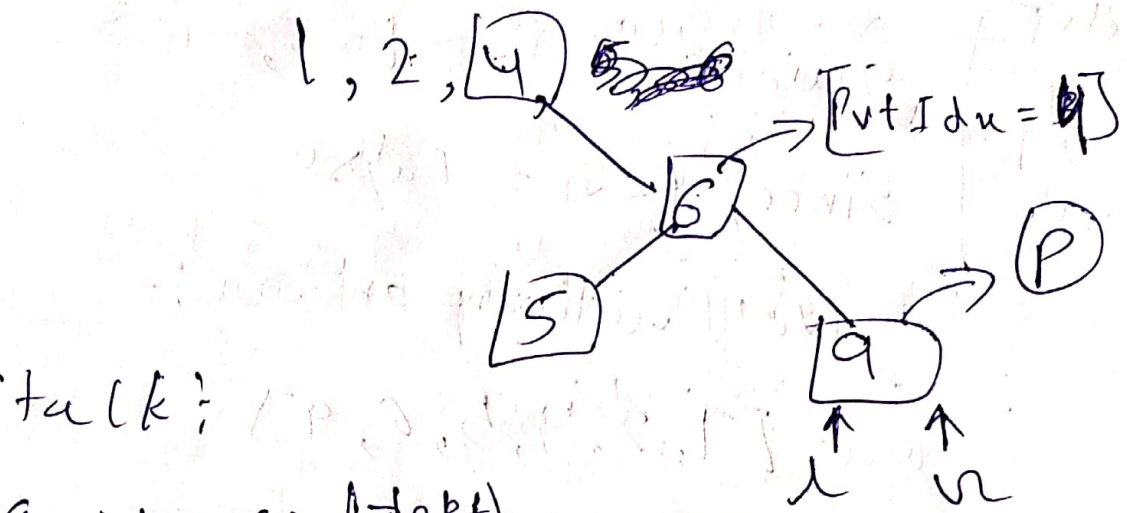
As, it return an arr, left-Right Parent of quicksort will Pop out.

And move to the "right child of Right Parent" of quicksort will push into the Stack and call out.

Stack :-



~~Value would be~~



Stack:

quicksort() → (RRt)
quicksort() → (Rt)
quicksort() → (R)

Implement quicksort()

l=5, r=5 → l<r = False

So, It will return an arr to the Right Parent ~~right~~ and ~~arr~~ PoPout right child Right Parent (rRT) from the stack.

arr = [1, 2, 4, 5, 6, 9]

Stack would look like this,

quicksort → (R)

Now, waited ~~Root~~ QuickSort for long time in the stack will ~~will~~ get their returned array i.e. [1, 2, 4, 5, 6, 9] & ~~finally~~ finally POP out from the stack.

And Print the Sorted Array

[1, 2, 4, 5, 6, 9]

Soln

Implementation with Code:-

I // Pivot Helper Function

```
function pivot(arr, startIdx=0, endIdx=arr.length-1)
{
    function swap(array, i, j) {
        let temp = array[i];
        array[i] = array[j];
        array[j] = temp;
    }

    let pivot = arr[startIdx];
    swapIdx = startIdx;

    for(let i=startIdx+1; i<arr.length; i++){
        if(pivot > arr[i]){
            swapIdx++;
            swap(arr, swapIdx, i);
        }
    }
```

II // Quick Sort Function

```
function quickSort(arr, left=0, right=arr.length-1){
    if(left < right){
        let pivotIdx = pivot(arr, left, right);
    //Left
        quicksort(arr, left, pivotIdx-1)
    //Right
        quickSort(arr, pivotIdx+1, right);
    }
```

Scanned with CamScanner

return arr;

};

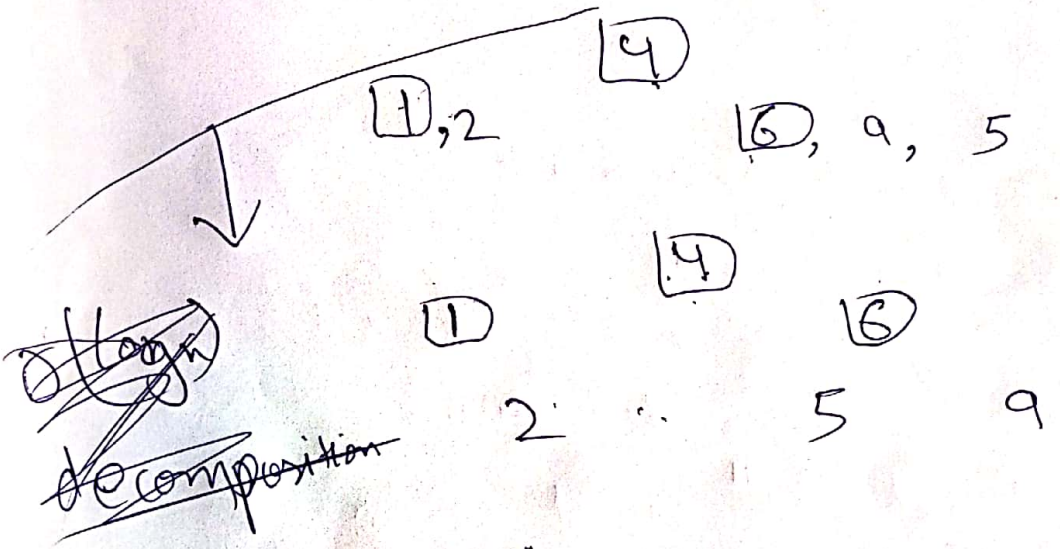
quickSort([4, 6, 9, 1, 2, 5]);

Output:-

[1, 2, 4, 5, 6, 9]

Working

[4, 6, 9, 1, 2, 5]



~~O(log n)~~

~~decomposition~~

Time Complexity:-

Best: O(n log n)

Avg: O(n log n)

Worst: $O(n^2)$ → (when array is sorted)

To avoid this we have to ~~pick~~ pivot middle element of the array instead of First.

Scanned with CamScanner