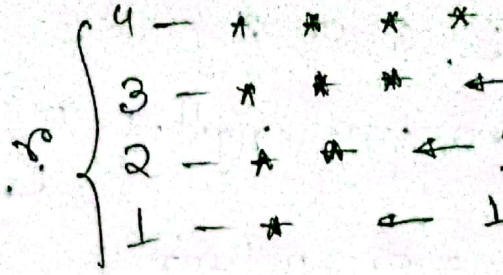
Pattern Printing :-



so, if (r = 0) return;

if (c < r) {

print (*)

triangle (r, c+1)

}

else, new line

triangle (r-1, 0)

Bubble sort: → each pass (swap if previous > next)

```
arr = {4, 3, 6, 2, 8}
bubble(arr, arr.length-1, 0)
                    ↳ index

if (r == 0) return;

if (c < r) {
        if (arr[c] > arr[c+1]) {
            swap(arr[c], arr[c+1])
        };
        bubble(arr, c, c+1);
    }
    else, bubble(arr, r-1, 0);
```

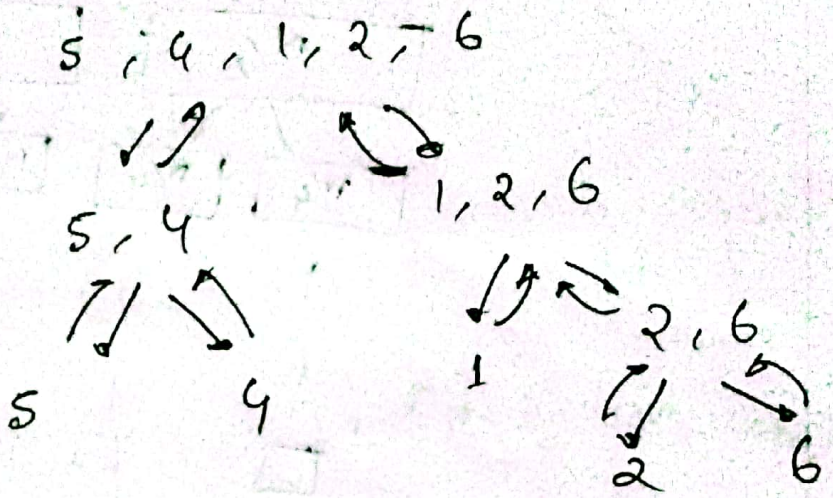
Selection Sort : each pass → max element right pos.

sort(arr, length, 0, 0)
r c max

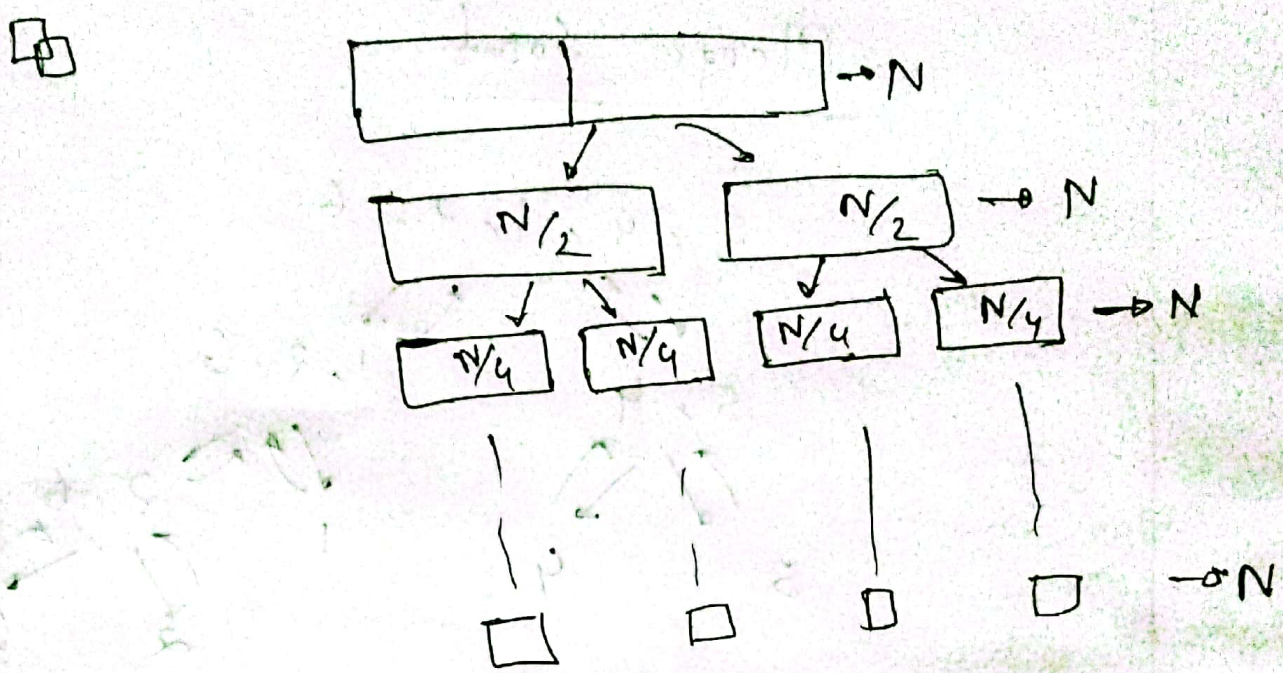
```
if(r==0)    return

if (c<r) {
        if(arr(c) > arr[max]) {
            sort(arr, r, c+1, c)
        }
        else {
            sort(arr, r, c+1, max)
        }
    }
else{
    swap(arr(max), arr(r-1)
    sort(arr, r-1, 0, 0)
}
```

Merge Sort

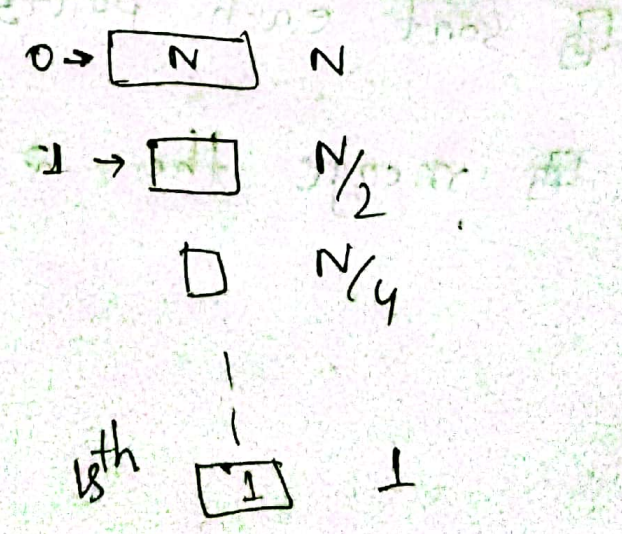


- First divide 2 parts
- sort each parts
- merge the 2 parts



At every level, N element are being merged.

Levels



$$\therefore \quad 1 = \frac{N}{2^k} \Rightarrow \boxed{k = \log_2 N}$$

∴ Total Time Complexity : $O(N * \log N)$

Space Complexity :

Auxiliary Space : $O(N)$

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

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

from Akra-Bazzi,

$$a = 2, \; b = \frac{1}{2}$$

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

$$T(x) = x + x \int_1^x \left(\frac{u+1}{u^{1+1}}\right) du$$

$$= x + x \int_1^x \left(\frac{1}{u} - \frac{1}{u^2}\right) du$$

$$= x + x \left[\log u + \frac{1}{u}\right]_1^x$$

$$= x + x \left[\log x + \frac{1}{x} - 1\right]$$

$$= x + x \log x + 1 - x.$$

$$= x \log x$$

$$\therefore T.C = O(N * \log N)$$

Quick Sort

- Pick Pivot

  In left side of pivot bring all elements that are < pivot and in right side of pivot bring all elements that are > pivot

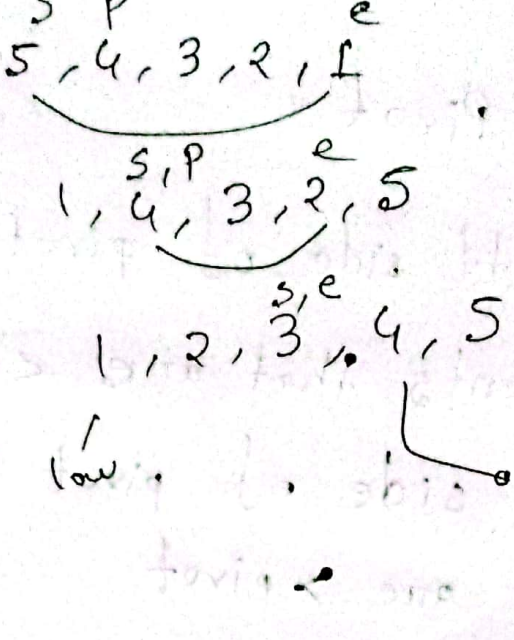
- 5, 4, 1, 3, 2

  Let's pivot = 4

  1, 3, 2, [4], 5

  so, each pass pivot element will come at it's right position

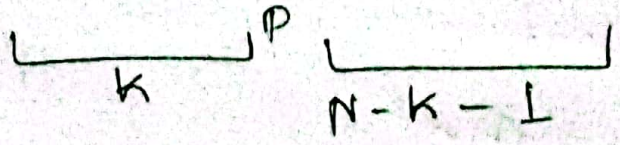
4. Put pivot at it's correct position



while ~~(~~ arr[s] < p
s++;

while → arr[e] > p
e--;

5. pivot →
1. random
2. corner
3. mid



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

worst case: pivot → largest / smallest

$k = 0$

$$T(N) = T(N-1) + O(N)$$
$$= O(N^2)$$

Best case: pivot → mid

$$T(N) = T(N/2) + T(N/2) + O(N)$$
$$= 2T(N/2) + O(N)$$
$$= O(N * \log N)$$

→ Quick sort

- Not stable
- In-place (preffered over merge sort because merge sort taking O(N) extra space)
- Merge sort is beffers in linked lists due to memory allocation.

Hybrid Sorting Algorithms (Tim sort)

Merge Sort + Insertion Sort

↓

works beffers for partially sorted arrays