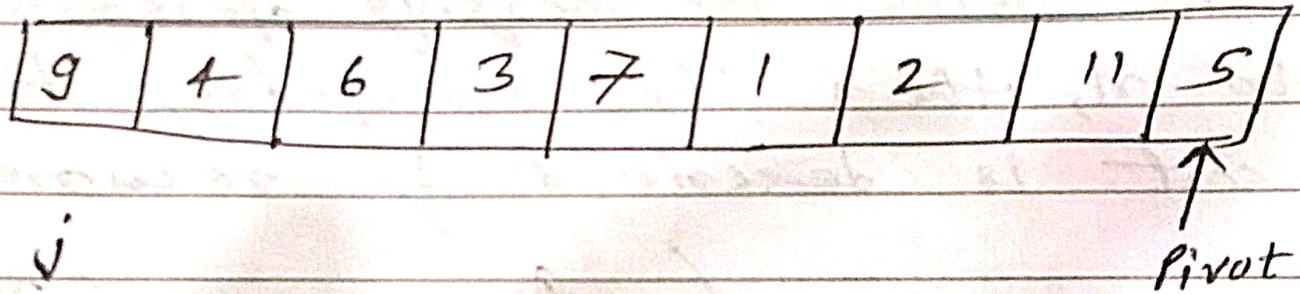
i location is ~~de~~ less of one location of j

Quick Sort

→ Partition
→ Quicksort.

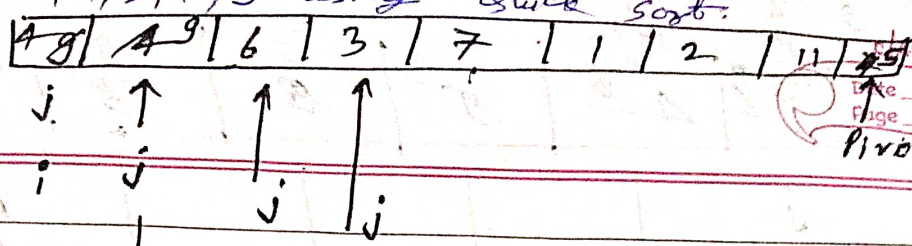


i = set as pointer to NULL

j = set pointer to list of element either from left, right side processing

- Increment for every element of j & proceed upto pivot element
- Increment of i only if a[i] < a[pivot].

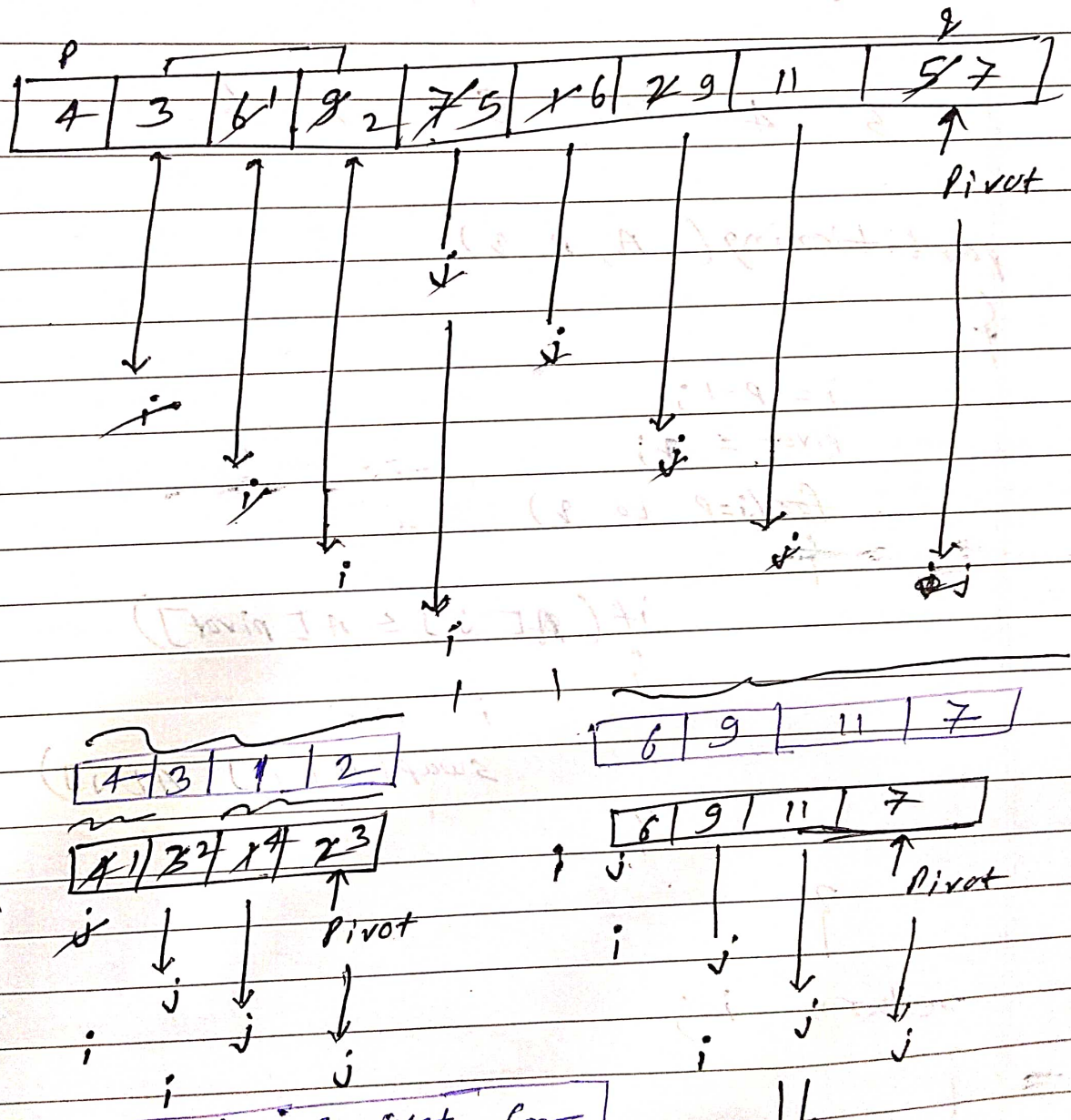
Q. Sort 9, 4, 6, 3, 7, 1, 2, 11, 5 using Quick Sort.

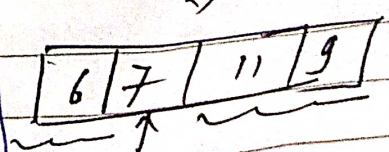


(1) i is only increment if a[j] ≤ a[pivot], then swap the elements a[i] and a[j];

(2) j is increment every time till it reaches to pivot location element

(3) If a[j] == a[pivot], then swap a[i] & a[pivot]





1. Initial setting of j & pivot for an Array A, and setting of i = j-1
Process: 1. Compare A[j] & A[Pivot] as a[j] ≤ a[pivot], then increment i = i+1 and swap of A[i] & A[j].
2. j is incremented till it doesn't reach to pivot & increment in i due to A[j] == A[Pivot]
3. If A[j] == A[Pivot] then swap A[i] & A[pivot].

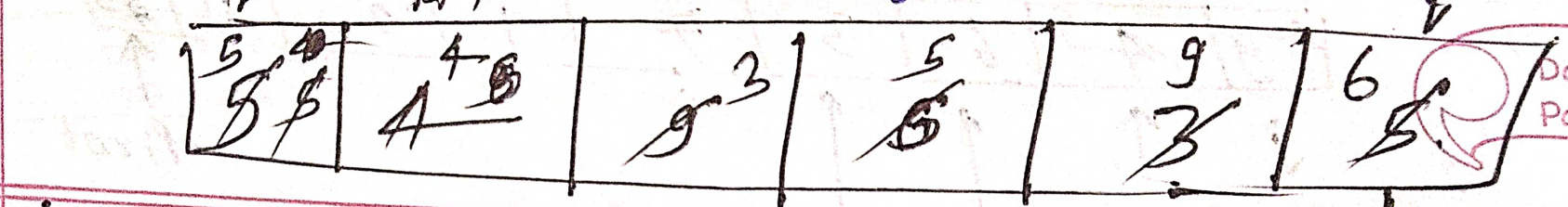
Algorithm Quick sort

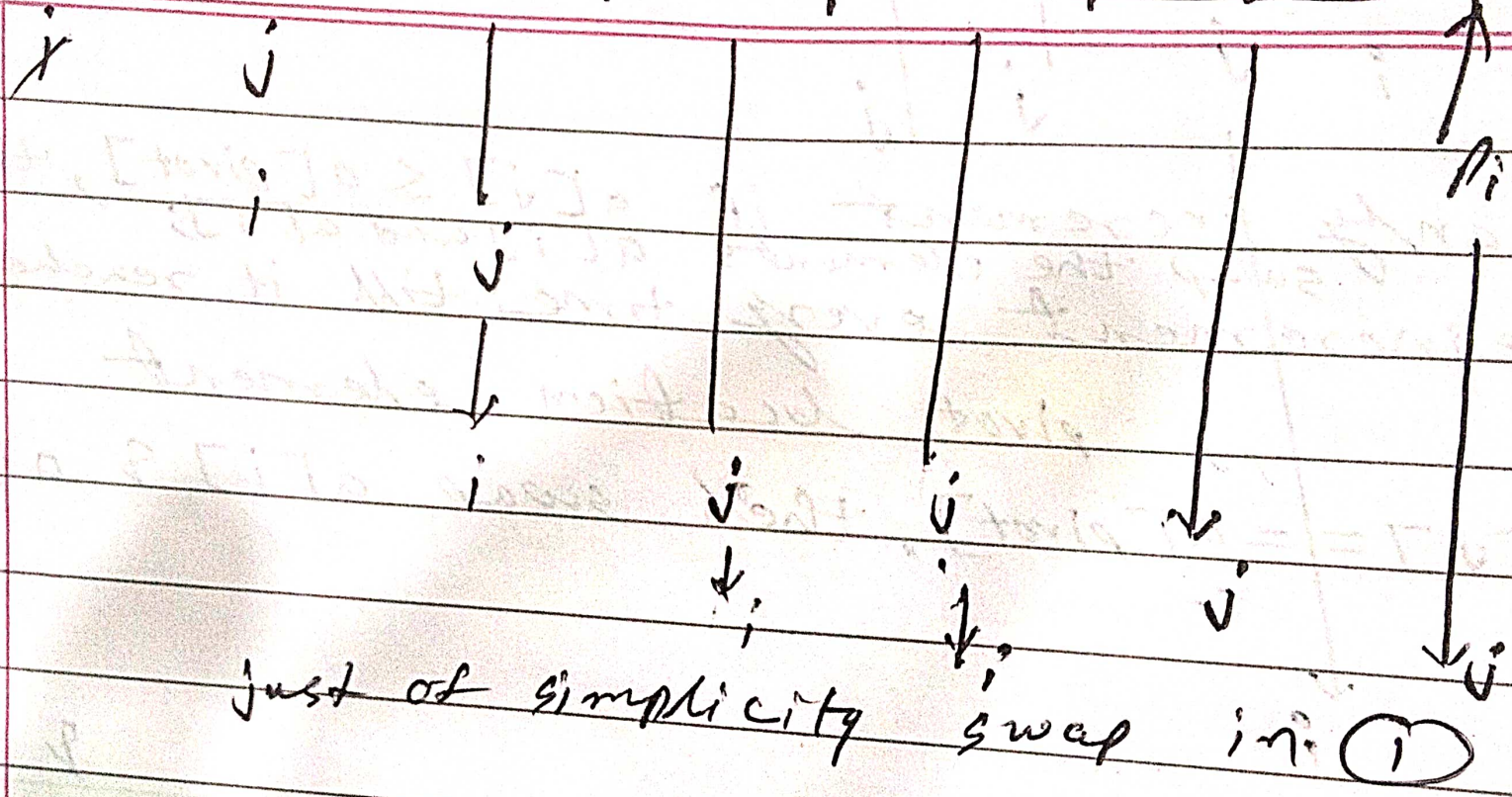
```
QuickSort( p, q)
{
    if( p < q) then
    {
        // Divide P into two subproblems
        j = Partition( A, p, q)
        // j is the position of the partitioning element.
        // Solve the subproblem
        QuickSort( p, j-1);
        QuickSort( j+1, q);
    }
}
```

p is j setting
@q is Pivot setting

```
partitioning( A, p, q)
{
    i = p-1;
    pivot = q;
    for(j=p to q)
    {
        if( A[j] ≤ A[pivot])
        {
            i = i+1;
            swap(A[i], A[j])
        }
    }
    return i;
}
```

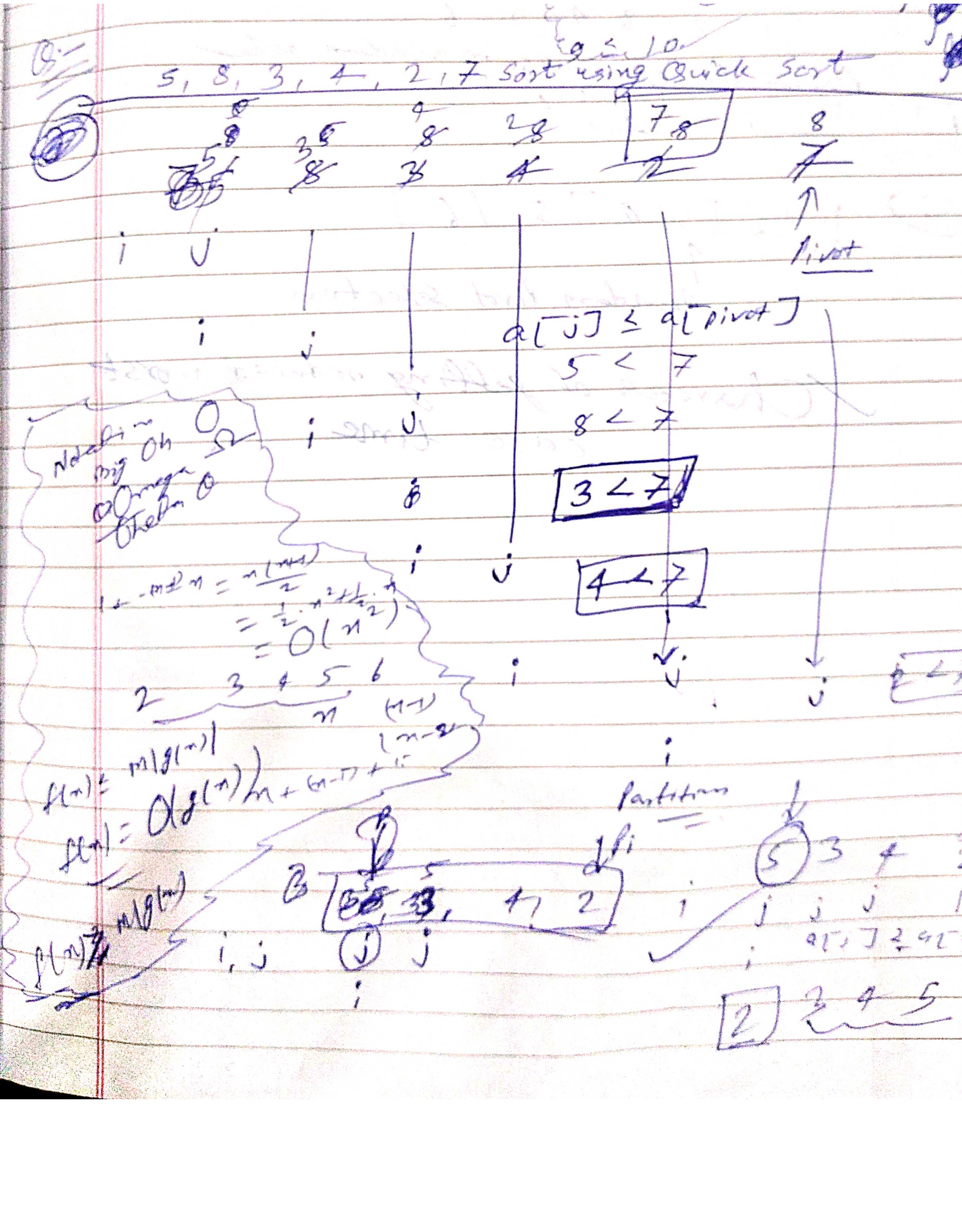
Q. 5 4 9 6 3 5 Sort using Quick sort





5 4 3 5 9 6

Q:- 5, 8, 3, 4, 2, 7 Sort using Quick Sort



Worst Case ~~Radix~~ Bucket Sort

$O(n^2)$

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

Randomized Quick Sort

1 2 3 4 5 6

Place the elements in random order

(I) | 1 | 3 | 2 | 6 | 4 | 5 |

(II) | 1 | 2 | 3 | 4 | 5 | 6 |

↑ Random Pivot selection.

Chances of getting reduced worst case time.

Complexity

1. Average Case Complexity $O(n \log_2 n)$
2. Worst Case $O(n^2)$. when elements are sorted to apply for quick sort.