MergeSort

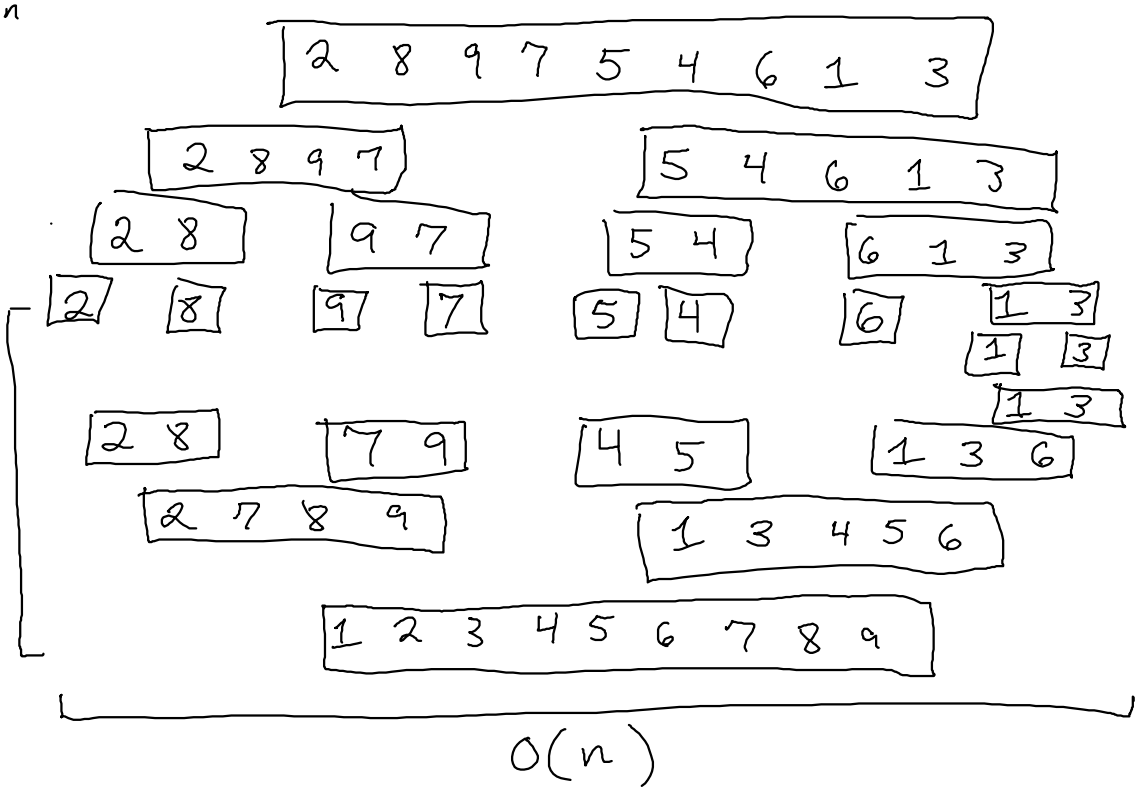
```
Function MergeSort(A, n):
    If n ≥ 2:
        X, Y ← Split(A, n)
        MergeSort(X, ⌊n/2⌋)
        MergeSort(Y, ⌊(n+1)/2⌋)
        Merge(X, ⌊n/2⌋, Y, ⌊(n+1)/2⌋, A)
    End If
End Function
```

$O(n \log n)$

$2^? = n$

$? = \log_2 n$

$O(\log n)$



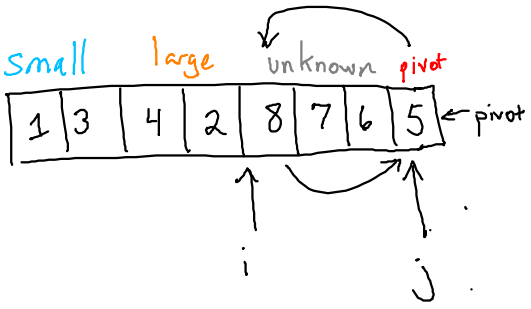
Sel Sort: n^2 100000 orders 10000000000

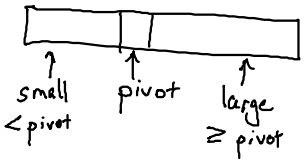
Merge Sort: 1000 $n \log n$ $1000 \cdot 100000 \cdot 17$

QuickSort ← in-place

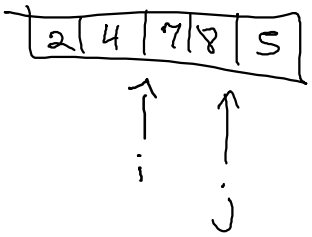
Function Partition(A, start, end): ← inclusive

Goal is to arrange array into two parts






```
Function Partition(A, start, end):
    pivot ← A[end]
    i ← start
    j ← start
    While j < end:
        If A[j] < pivot:
            // small
            Swap A[j] with A[i]
            i ← i+1
            j ← j+1
        Else:
            j ← j+1
        EndIf
    EndWhile
    Swap A[end] with A[i]
    Return i
End Function
```



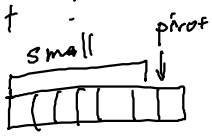
→ Lomuto Partition

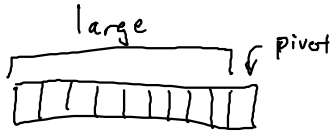
Alt Hoare Partition

```
Function QuickSort(A, start, end):
    If end - start > 0:
        pivotIndex ← Partition(A, start, end)
        QuickSort(A, start, pivotIndex - 1)
        QuickSort(A, pivotIndex+1, end)
    EndIf
End Function
```

Complexity of QuickSort:

Randomized





Worst Case: $O(n^2)$

Expected Case: $O(n \log n)$

1 2 3 4 5 6

* RQS: at the start of partition, swap A[end] w/ a random element

Function Crambler Sort (A, n):

Flip a coin

If coin is heads:

A is magically sorted

Else

try again

best case $O(1)$

worst case $O(\infty)$

$$\frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 2 + \frac{1}{8} \cdot 3 + \cdots = \sum_{i=1}^{\infty} \frac{1}{2^i} \cdot i = 2$$