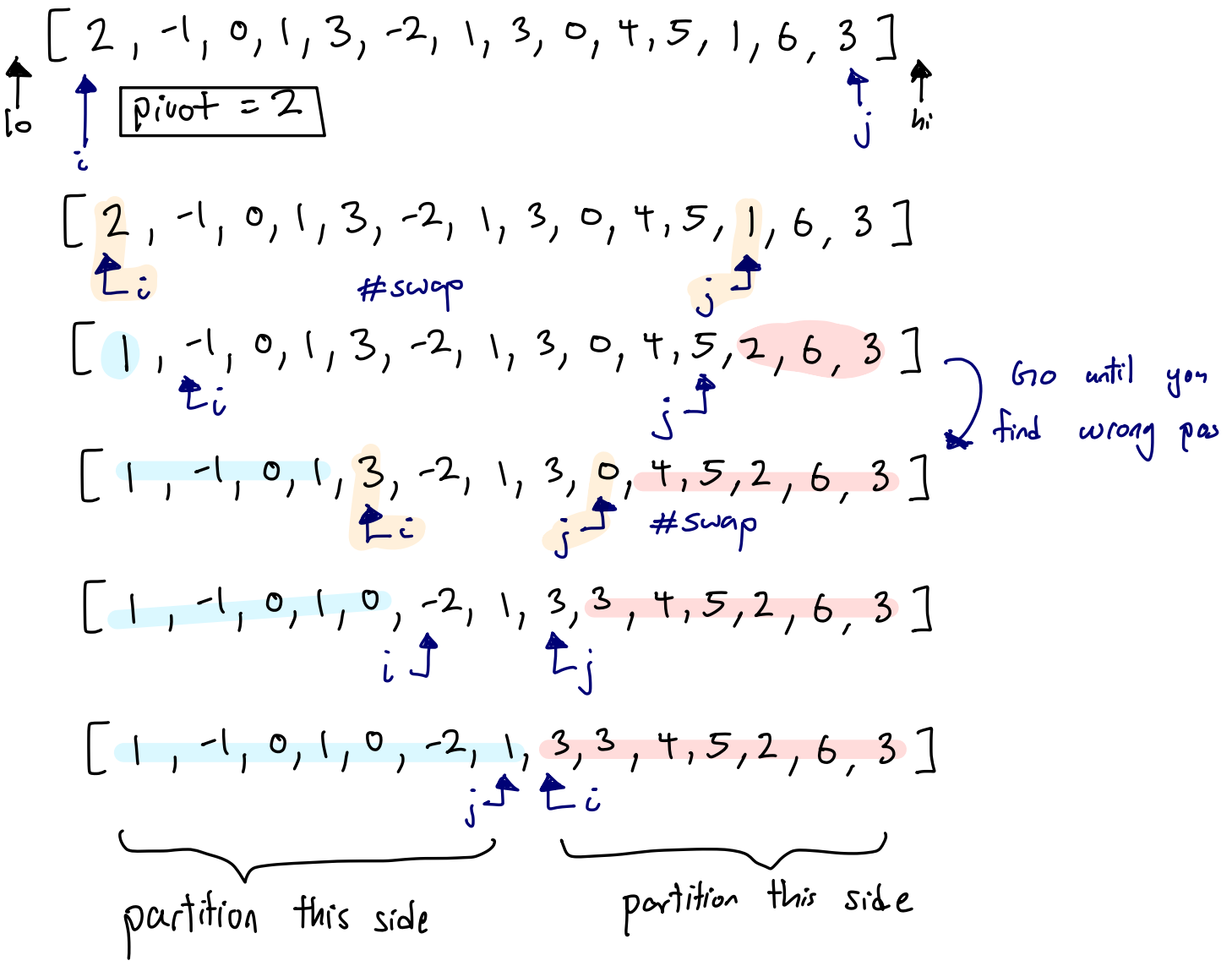
Quicksort

```
def partition(arr, low, high):
    pivot = arr[low]
    i = low - 1
    j = high + 1
    while (True):
        i += 1
        while (arr[i] < pivot):
            i += 1
        j -= 1
        while (arr[j] > pivot):
            j -= 1
        if (i >= j):
            return j
        arr[i], arr[j] = arr[j], arr[i]

def quickSort(arr, low, high):
    if (low < high):
        pi = partition(arr, low, high)
        quickSort(arr, low, pi)
        quickSort(arr, pi + 1, high)
```

Hoare's

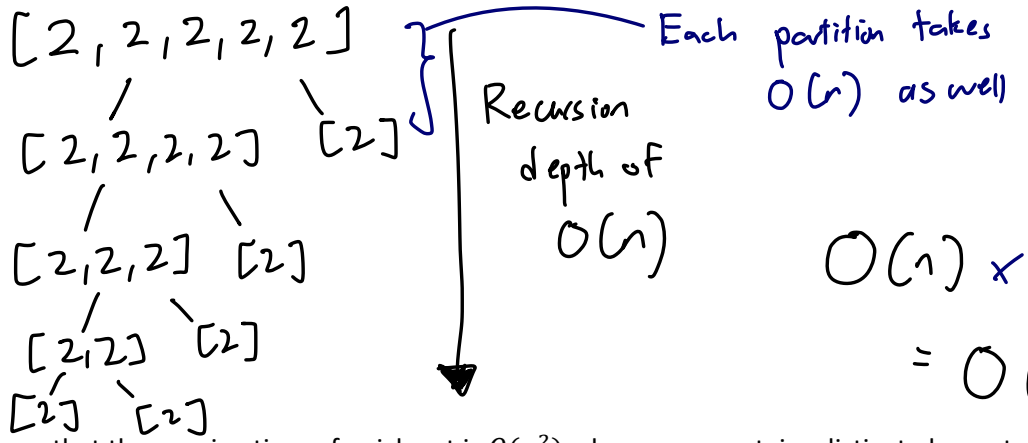


Partition cost → O(n) time complexity

Hoare's > Lomuto > Out-of-place

- Hoare's → less swaps → in-place
- Lomuto → in-place
- Out-of-place

What is the running time of quicksort when all elements of array has the same value?

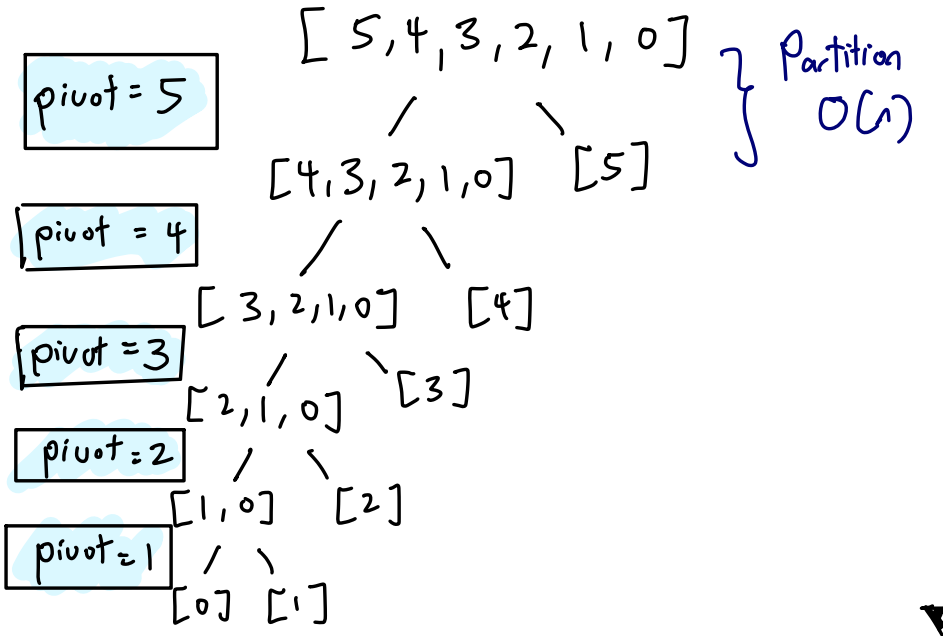


Could improve with DNF

$O(n) \times O(n)$

$= O(n^2)$

Show that the running time of quicksort is $O(n^2)$ when array contains distinct elements sorted in descending order.



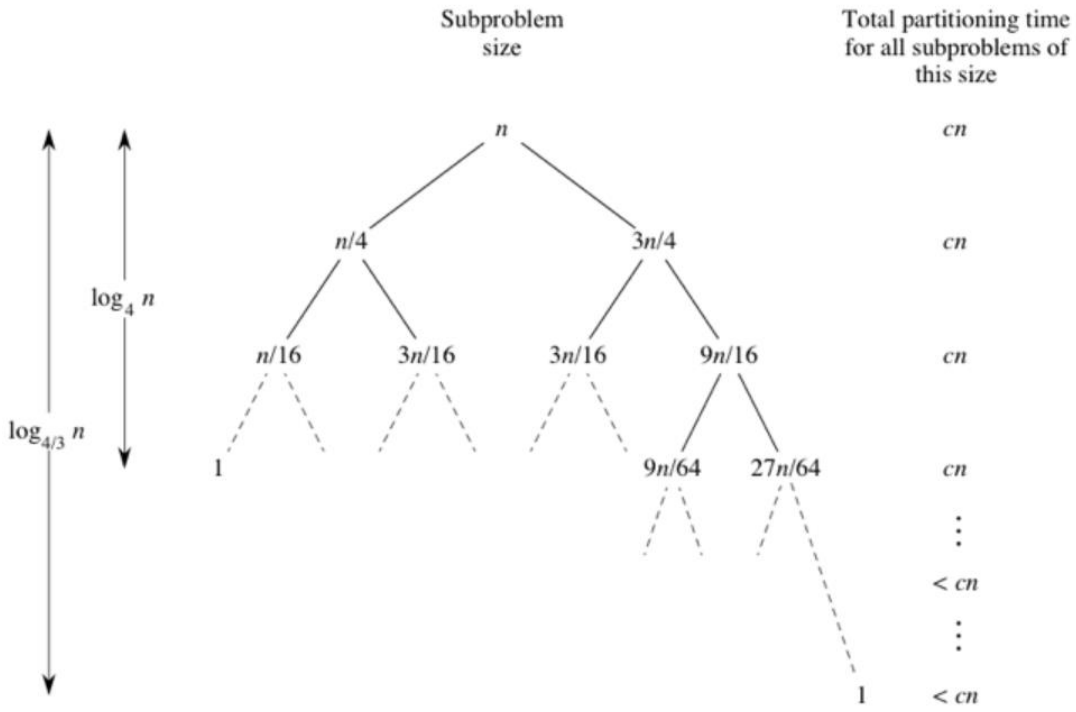
Recursive depth of $O(n)$

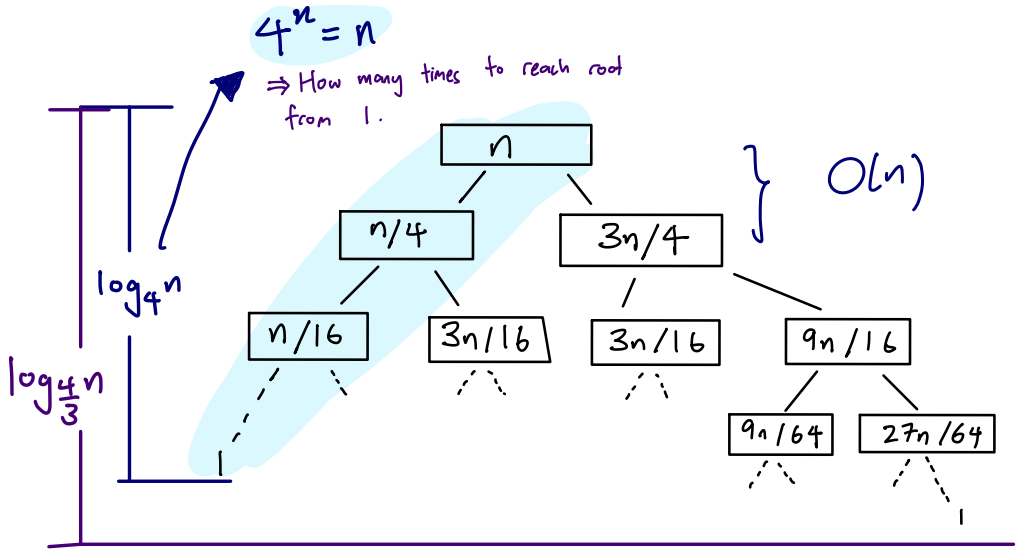
$O(n^2)$

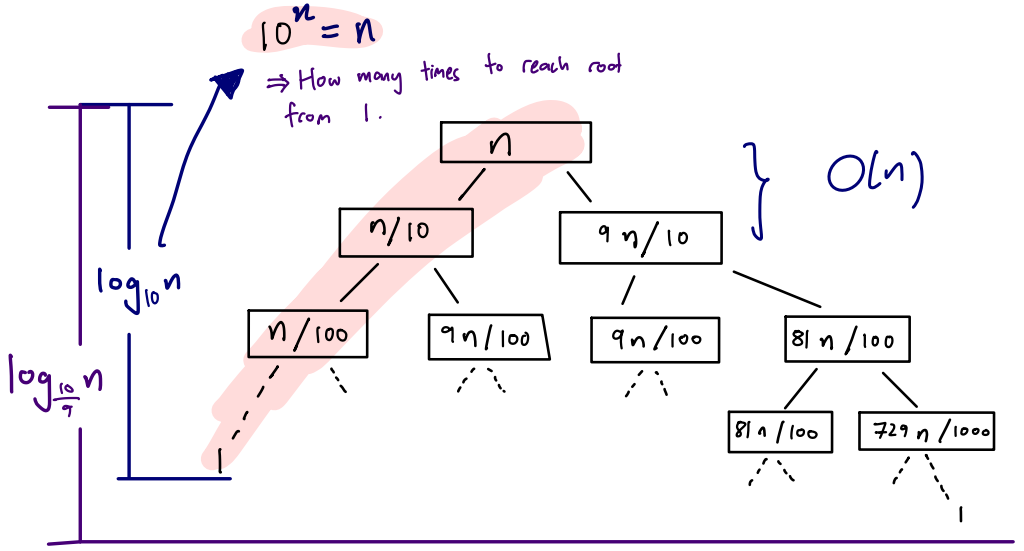
- Depends on pivot selection

Average time

[redacted] complexity analysis







Regardless if split by 25% : 75%, 10% : 90%, 1% : 99%, still can prove as $O(n \log n)$, just that it is $\log_{\frac{4}{3}} n$, $\log_{\frac{10}{9}} n$, $\log_{\frac{100}{99}} n$,

→ inefficient ? Yes.

→ $\log n$? Yes.

So, on average, even with bad splits, it is still $\log n$.

Dutch National Flag Algorithm

→ concept that improves partitioning

```
function PARTITION(array[1..n], pivot)
    lo = 1, mid = 1, hi = n
    while mid ≤ hi do
        if array[mid] < pivot then
            swap(array[mid], array[lo])
            lo += 1, mid += 1
        else if array[mid] = pivot then
            mid += 1
        else
            swap(array[mid], array[hi])
            hi -= 1
    return lo, mid
```

// Red case } less than

// White case } equals to

// Blue case } more than

pivot = 2

[2, 1, 0, 0, -1, 1, 3, 4, 2, 2, 3, 0, 1, -1, 4, 2]

[1, 0, 0, -1, 1, 0, -1, 2, 2, 2, 2, 4, 3, 4, 3]