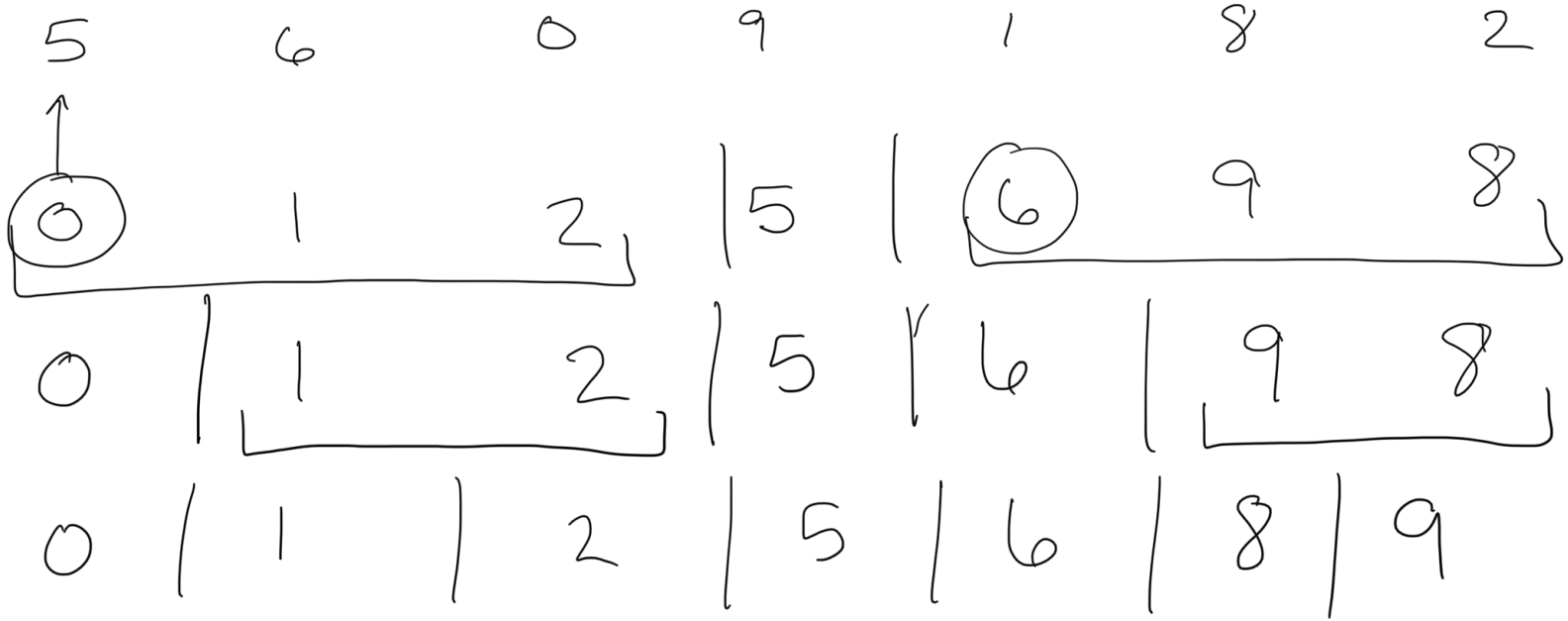
Sorting V

- QuickSort Overview
- 3-Way QuickSort
- QuickSort Analysis
- QuickSort vs. MergeSort

Quicksort Overview

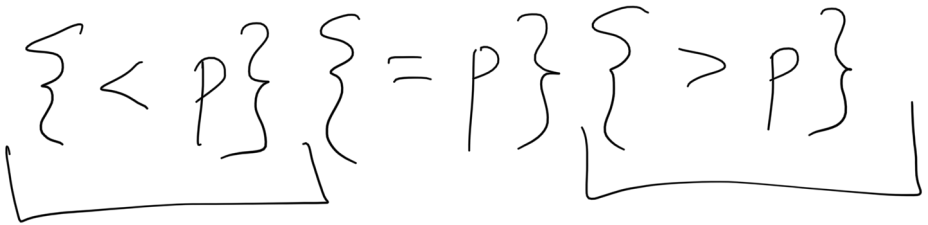
- Paradigm: Divide & Conquer
  - Dividing: divide the array by pivoting it around some random value
  - Solving: automatic w/ the recursion
  - Combining: " " " "
- Randomization: pick a random pivot

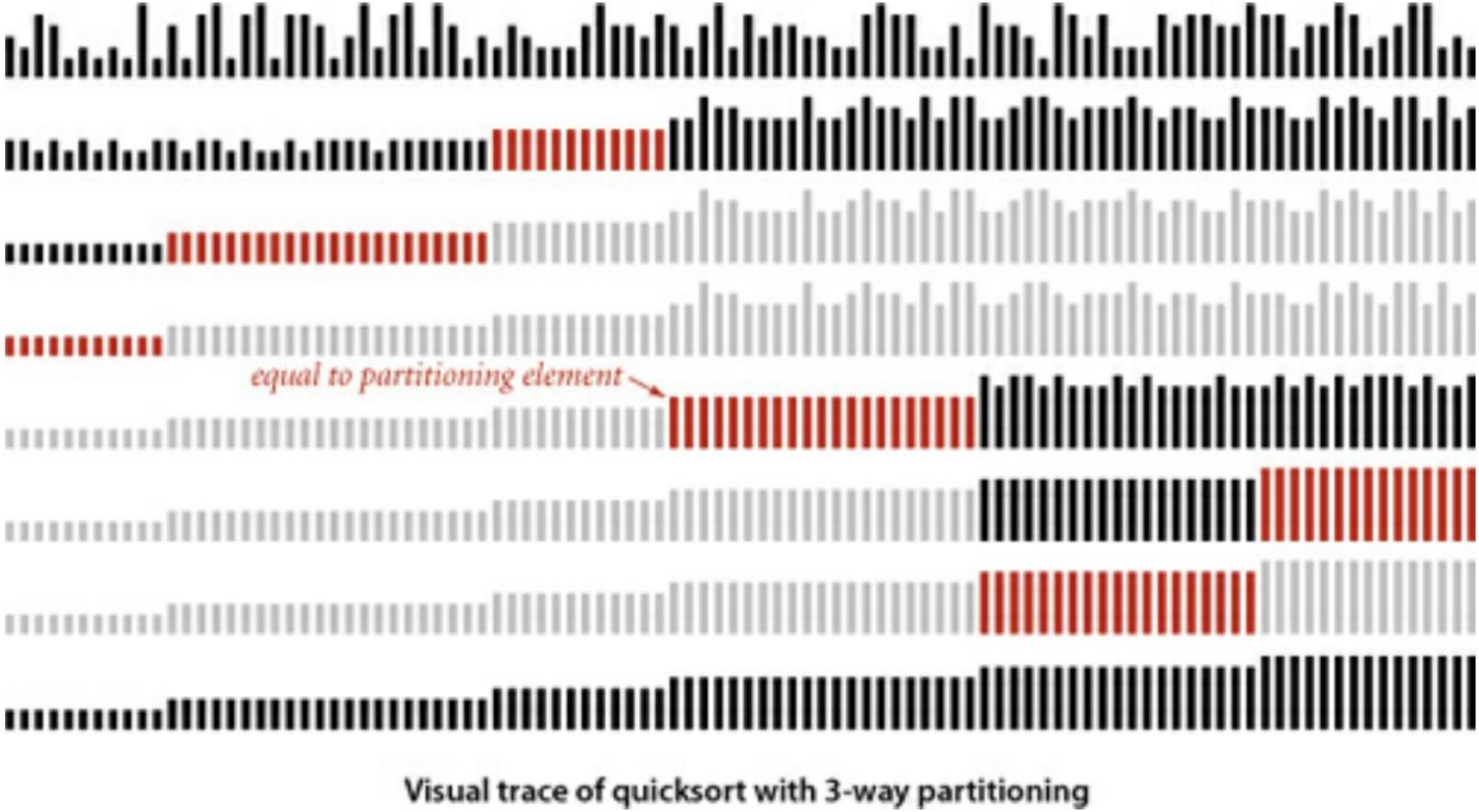
Example



3-way Quicksort

- Instead of {≤} p {>}, we divide into three sets: {<}{=}{>}.
- This can help the runtime if there are a lot of equal elements, but it won't change the overall big-Oh of the runtime.



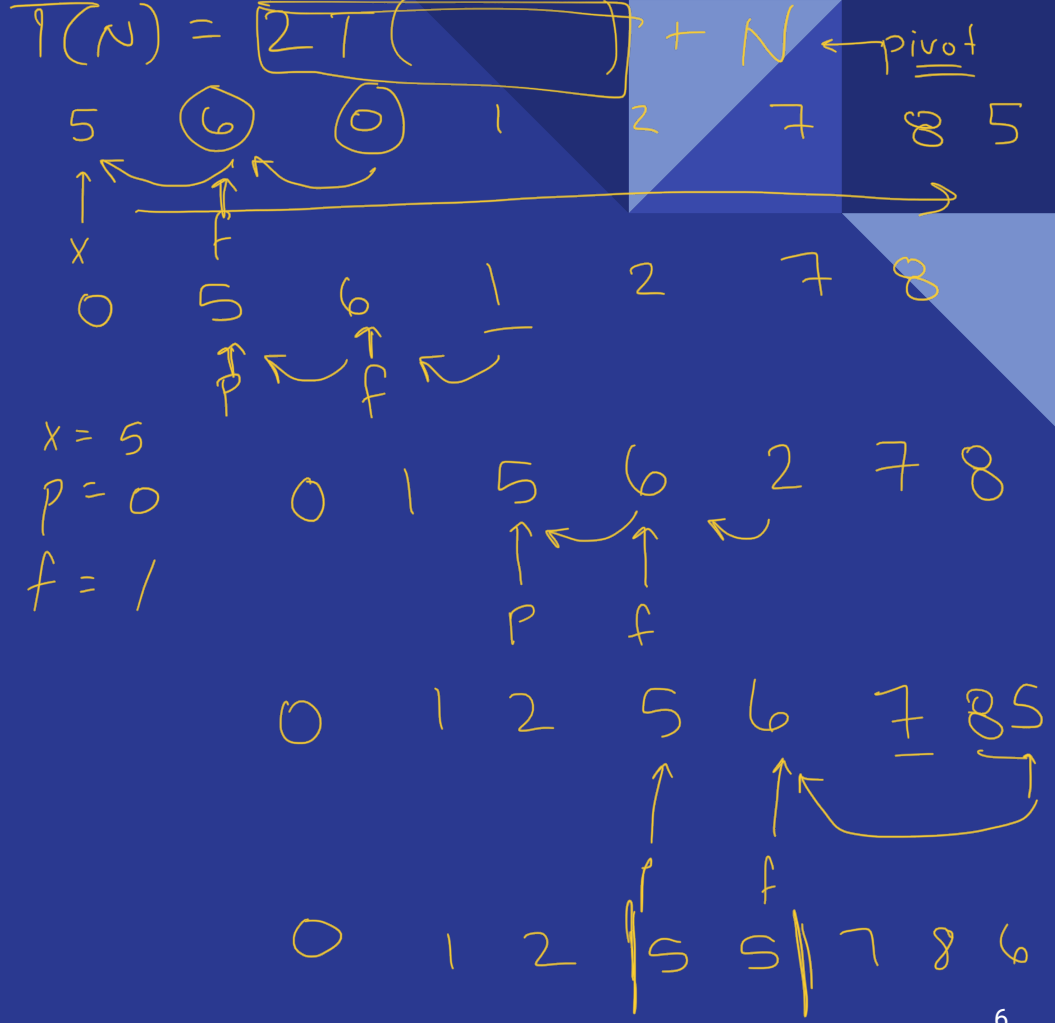


Visual trace of quicksort with 3-way partitioning

Picture from Sedgewick and Wayne

```
Algorithm quickSort
Input: an array A of size N
Output: A, sorted
pivot(A, 0, N-1)

procedure pivot (Array A, int i, int j)
    if(i >= j)
        return
    if(i < 0 || i >= A.length || j >= A.length)
        return
    int x = A[i]
    int p = i;
    int f = i + 1;
    for(int k = f; k <= j; k++)
        if(A[k] < x)
            swap A[f] and A[k]
            swap A[p] and A[f]
            p++
            f++
        else if(A[k] == x)
            swap A[f] and A[k]
            f++
    end for
    pivot(A, i, p-1)
    pivot(A, f, j)
end pivot
```



Best-Case Runtime

pivot is median

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

trivial best case : all the same element

(3-way QS) $\longrightarrow O(N)$

Worst-Case Runtime

min or max as pivot

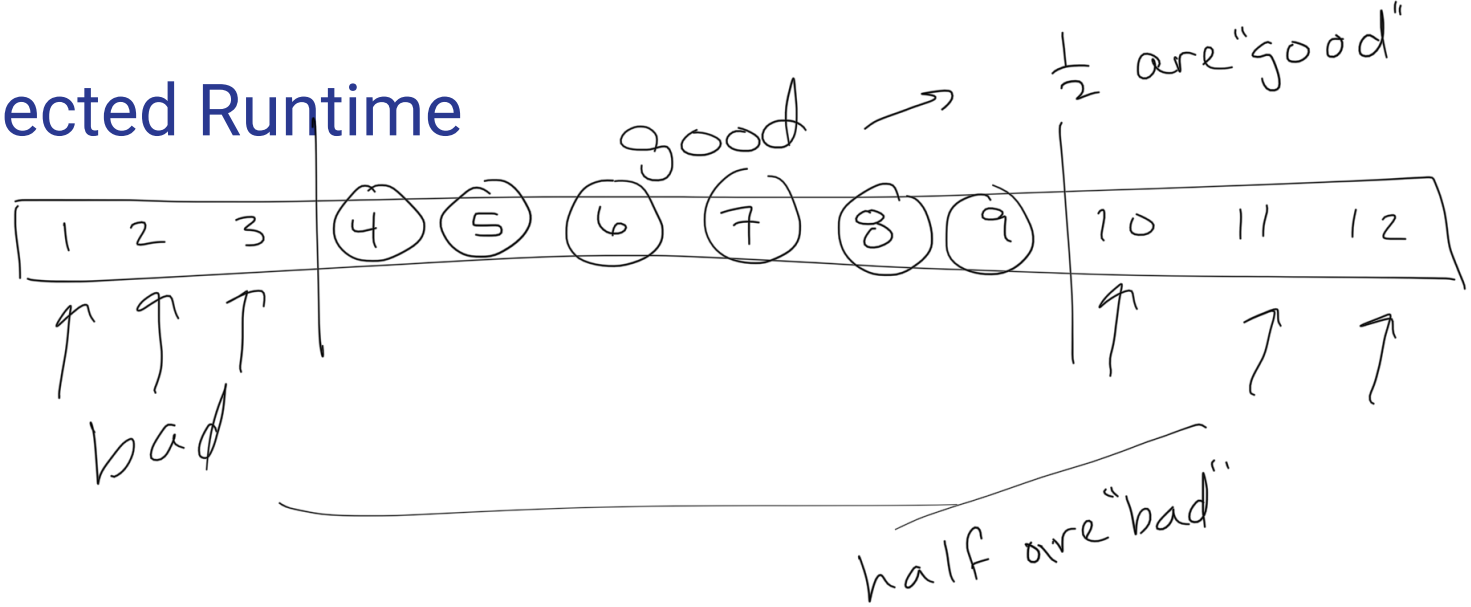
[ ] min [N - 1]

[N - 1], max [ ]

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

$$\Rightarrow O(N^2)$$

Expected Runtime



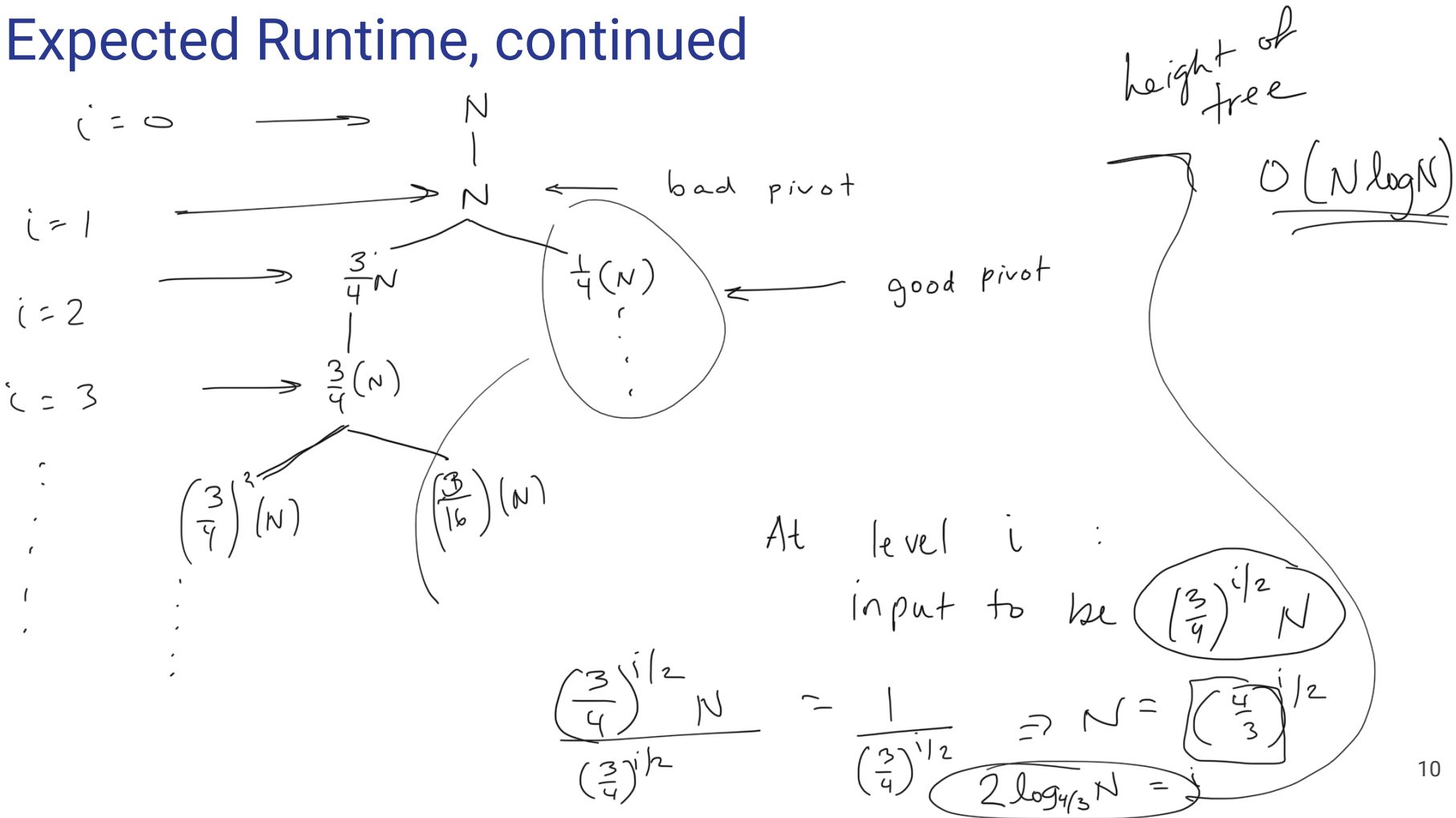
$P(\text{good pivot}) = \frac{1}{2}$

$P(\text{bad pivot}) = \frac{1}{2}$

expected # of good pivots : $\frac{1}{2}$ of them

expected # of bad pivots : $\frac{1}{2}$ of them

Expected Runtime, continued



Quicksort vs. Mergesort

mergesort

O(N) → combining merge

second array

→ extra space is : O(N)

Quicksort

dividing → pivot

O(N)

in place

→ extra space :

O(log N)

Quicksort Summary