Lecture 05: Sorting by Divide and Conquer

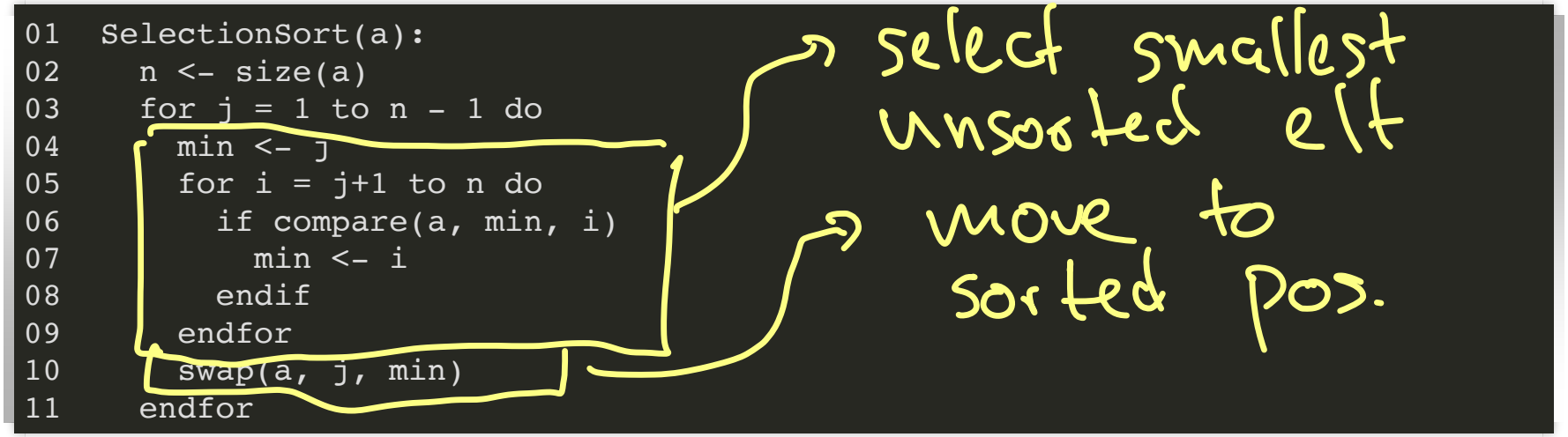
COSC 311 *Algorithms,* Fall 2022

Overview

1. Efficiency of Sorting
2. Review of Big O
3. Sorting by Divide and Conquer: MergeSort

Previously

Careful analysis of *correctness* of SelectionSort



Today

Focus on *efficiency*

- How many elementary operations does a procedure perform?

Questions about SelectionSort

```
01  SelectionSort(a):
02    n <- size(a)
03    for j = 1 to n - 1 do
04      min <- j
05      for i = j+1 to n do
06        if compare(a, min, i)
07          min <- i
08        endif
09      endfor
10      swap(a, j, min)
11    endfor
```

How many comparisons?

$n + (n-1) + (n-2) + \cdots + 1 = n!$

$(n-1) + (n-2) + (n-3) + \cdots + 1 = \frac{n(n-1)}{2} \approx \frac{1}{2}n^2$

How many swaps?

$n-1$ (1 per outer iteration)

$\frac{n}{2}$ swaps is best possible for any array

How many "elementary" operations (including arithmetic)?

???

Abstracting Away Unknowns

Uncertain Parameters:

1. precise running times of elementary operations
2. overhead associated with executing algorithm

Assume:

1. elementary operations take constant (but unknown) time
2. overhead of starting computation is negligible for large inputs

Focus on **asymptotic growth** of # of operations as function of input size

Big O Notation

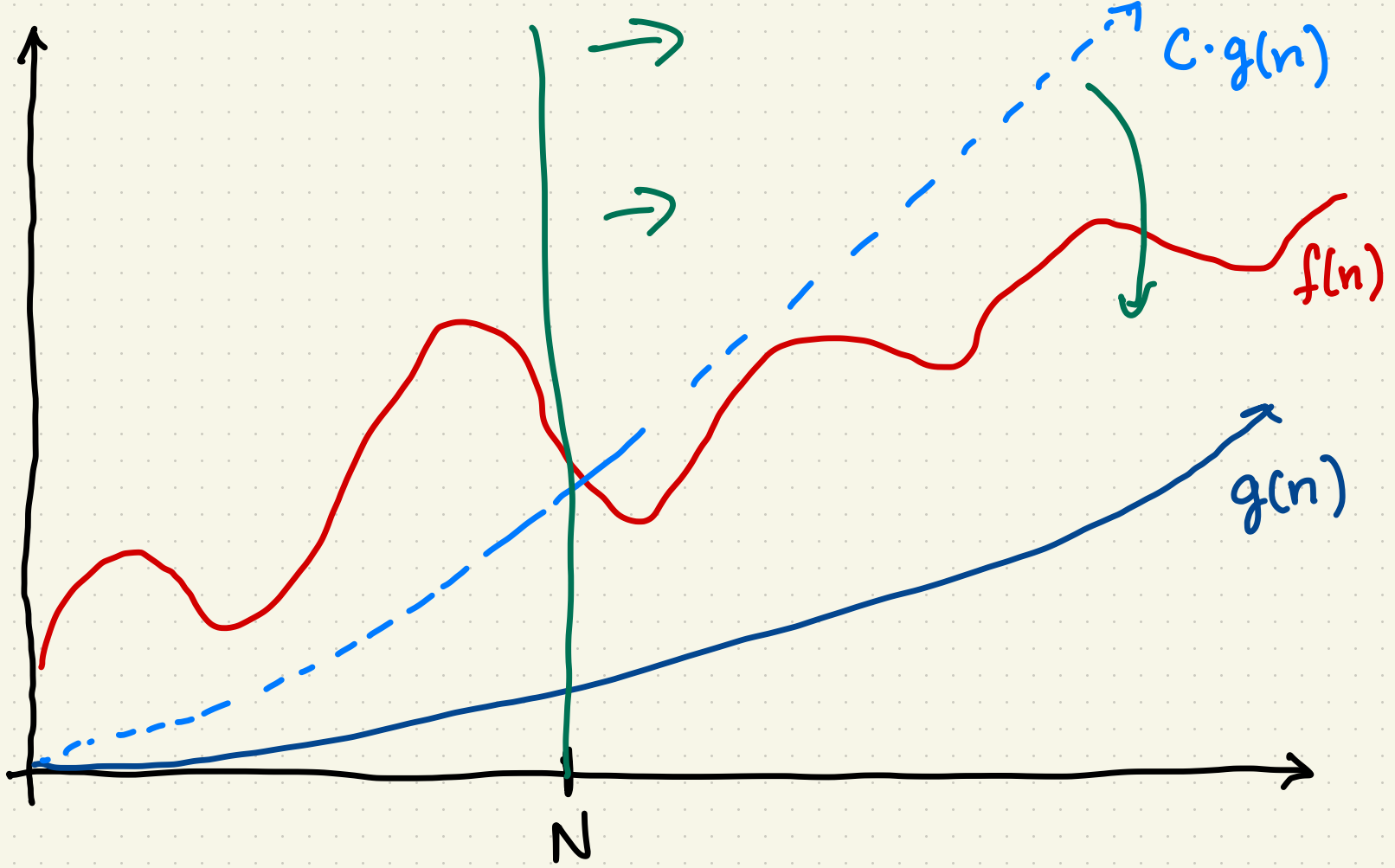
Qualitative measure of function growth:

1. ignore constant factors
2. ignore values on small inputs

Formally: f, g functions, write $f = O(g)$ if

- exists $C > 0$ and $N \geq 1$ such that
  - $f(n) \leq Cg(n)$ for all $n \geq N$

Big O in Pictures


C·g(n)
f(n)
g(n)
N

Properties of O

1. all constants are $O(1)$
2. if $f(n) \leq g(n)$ for all n, then $f = O(g)$
   - if $a \leq b$ then $n^a = O(n^b)$
   - if $a < b$ then $n^b \neq O(n^a)$
3. if $f = O(g)$, then $a \cdot f = O(g)$
4. if $f = O(g)$ and $g = O(h)$, then $f = O(h)$
5. if $f, g = O(h)$, then $f + g = O(h)$
6. if $f_1 = O(g_1)$ and $f_2 = O(g_2)$ then $f_1 \cdot f_2 = O(g_1 \cdot g_2)$

any constant

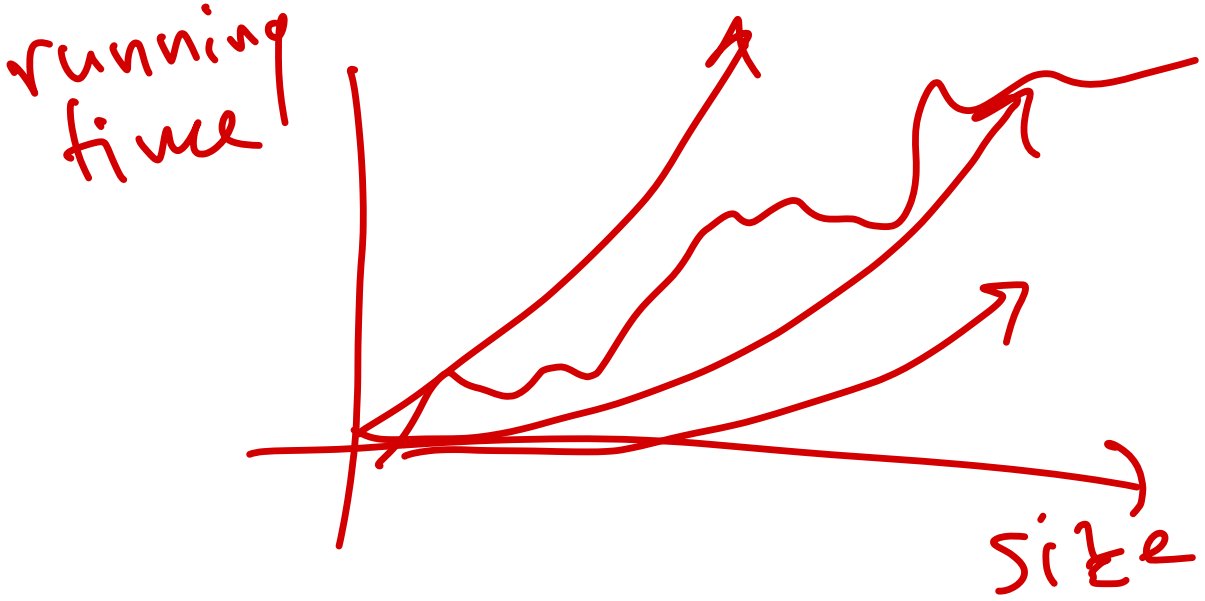
$f(n) = \underbrace{10n^3 + 14n^2 - 37n + 100,000}_{O(n^3)}$

$= O(n^3)$

Running time of SelectionSort in O?

```
01  SelectionSort(a):
02    n <- size(a)
03    for j = 1 to n - 1 do
04      min <- j
05      for i = j+1 to n do
06        if compare(a, min, i)
07          min <- i
08        endif
09      endfor
10      swap(a, j, min)
11    endfor
```

O(1) O(n) O(n) $O(n^2)$



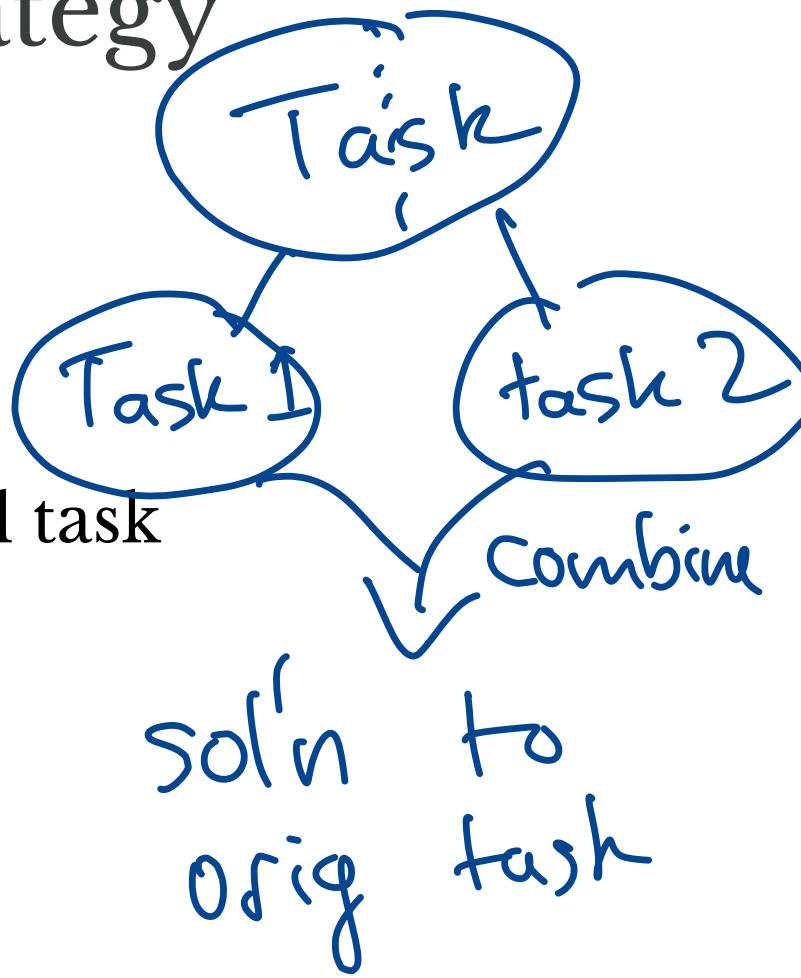
Faster Sorting?

Divide and Conquer

Divide and Conquer Strategy

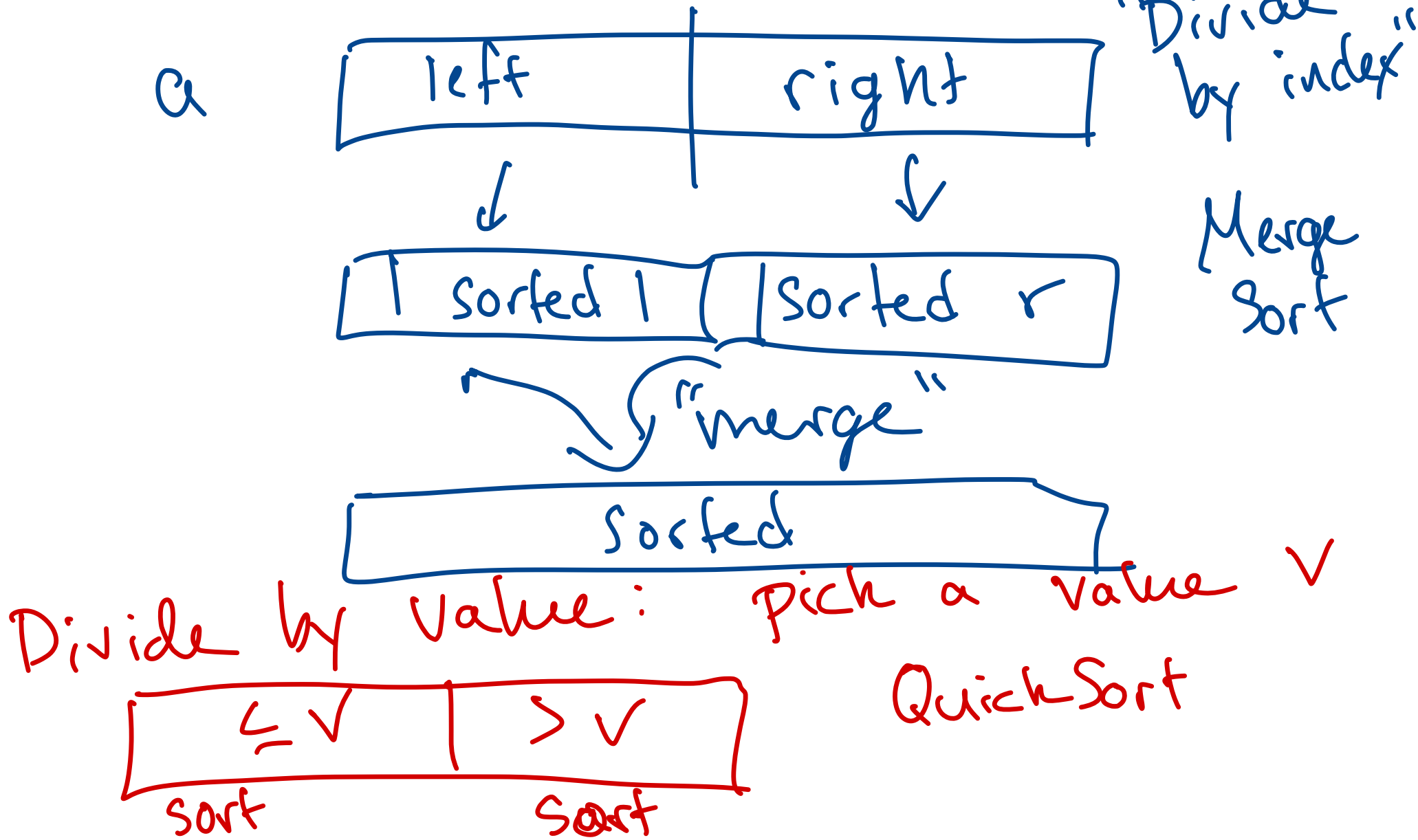
Given a computational task:

1. Divide the task into sub-tasks
   - smaller instances of *same task*
2. Solve the sub-tasks recursively
3. Combine solutions to solve original task



Question

How could divide the sorting task into sub-tasks?



Sorting by D&C

1. Divide by *index*
    - MergeSort
2. Divide by *value*
    - QuickSort
3. Divide by *bit representation*
    - RadixSort

Dividing by Index: MergeSort

As before, a an array of size n

- access/assignment by index (not just compare/swap)

MergeSort algorithm:

1. divide a into halves $m = (n + 1)/2$
2. sort $a[1..m - 1]$ recursively
3. sort $a[m..n]$ recursively
4. merge $a[1..m - 1]$ and $a[m..n]$ to form sorted array

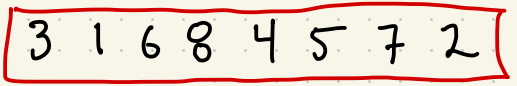
Pseudocode

```
# sort values of a between indices i and j-1
MergeSort(a, i, j):
  if j - i = 1 then
    return
  endif
  m <- (i + j) / 2
  MergeSort(a,i,m)
  MergeSort(a,m,j)
  Merge(a,i,m,j)
```

Base case

Illustration of MergeSort

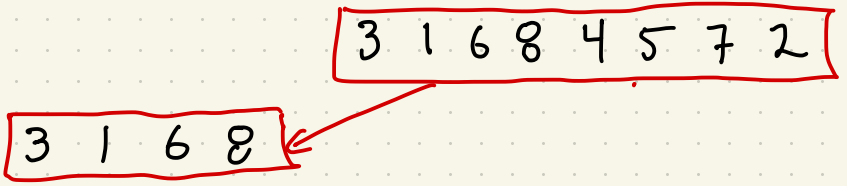
MergeSort:

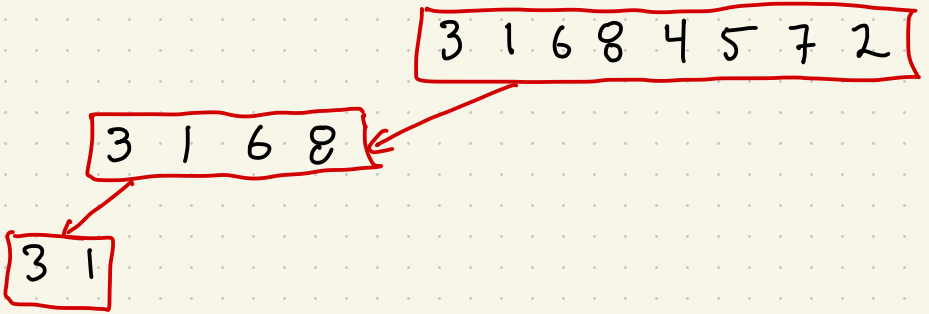


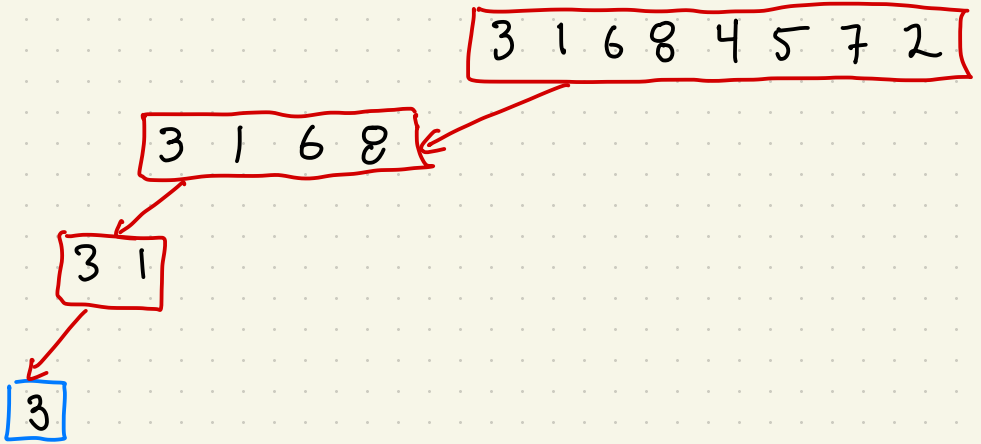
method call

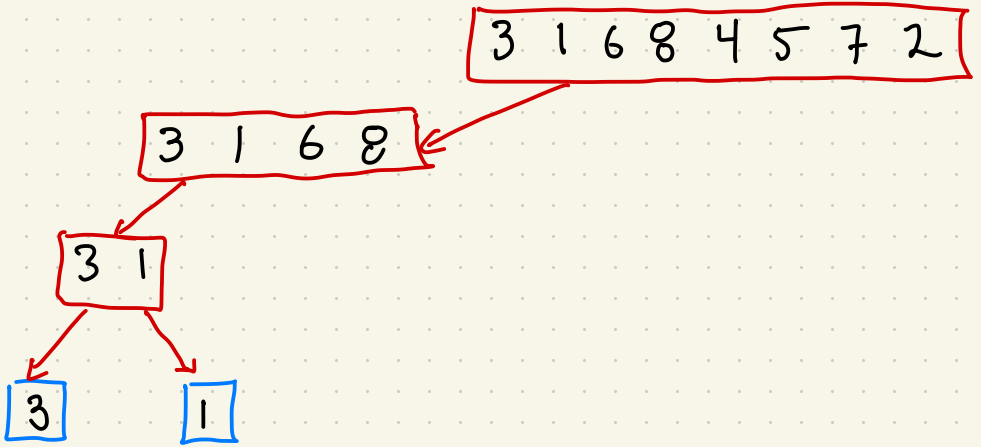
return value

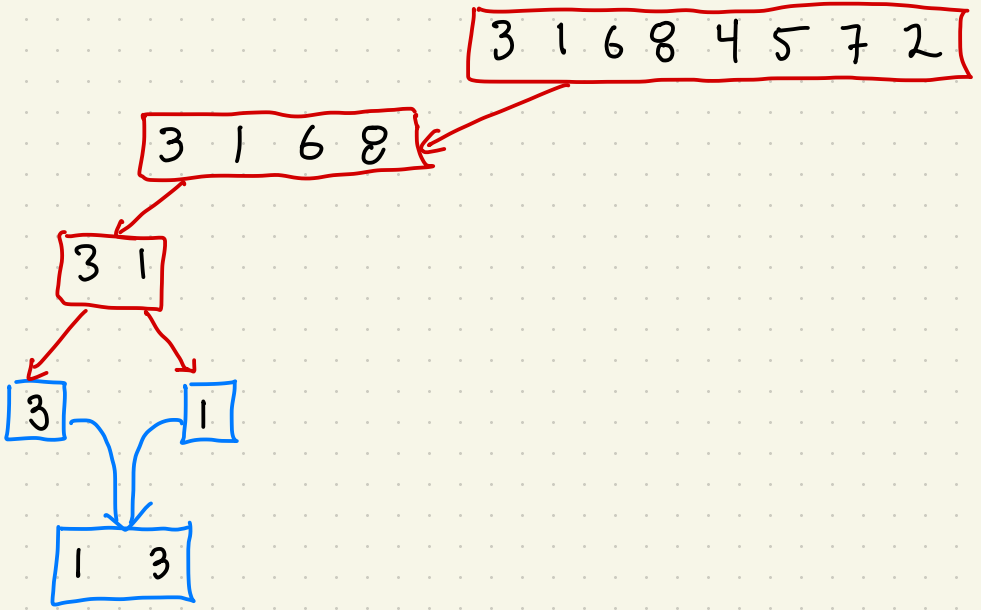
merge

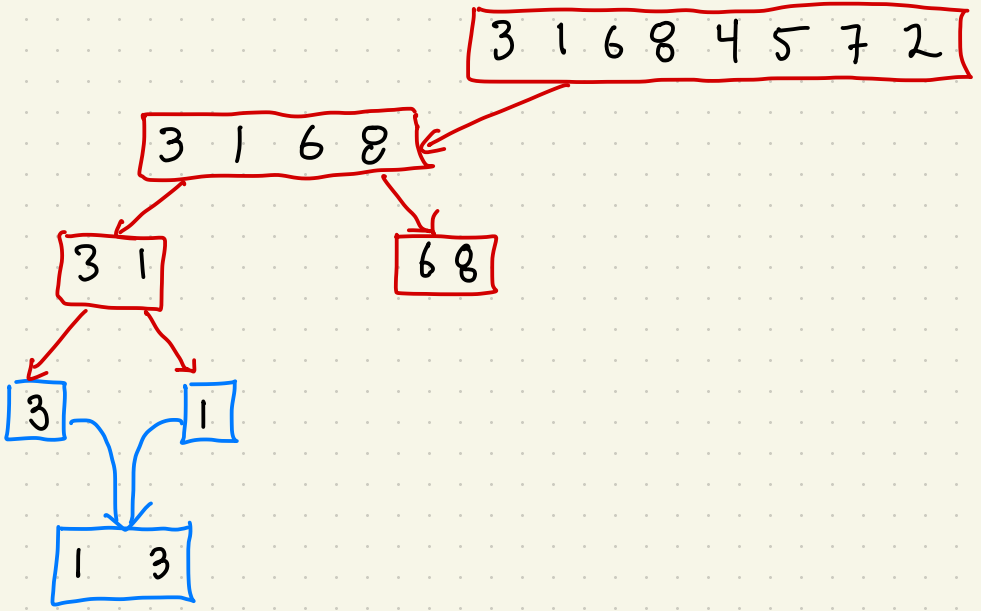

3 1 6 8 4 5 7 2
3 1 6 8

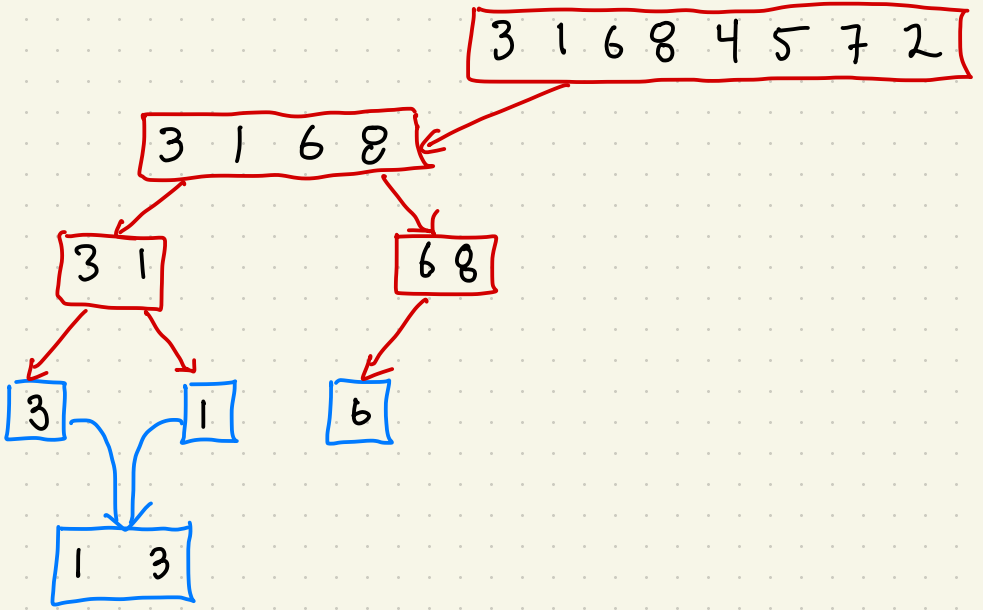

3 1 6 8 4 5 7 2
3 1 6 8
3 1

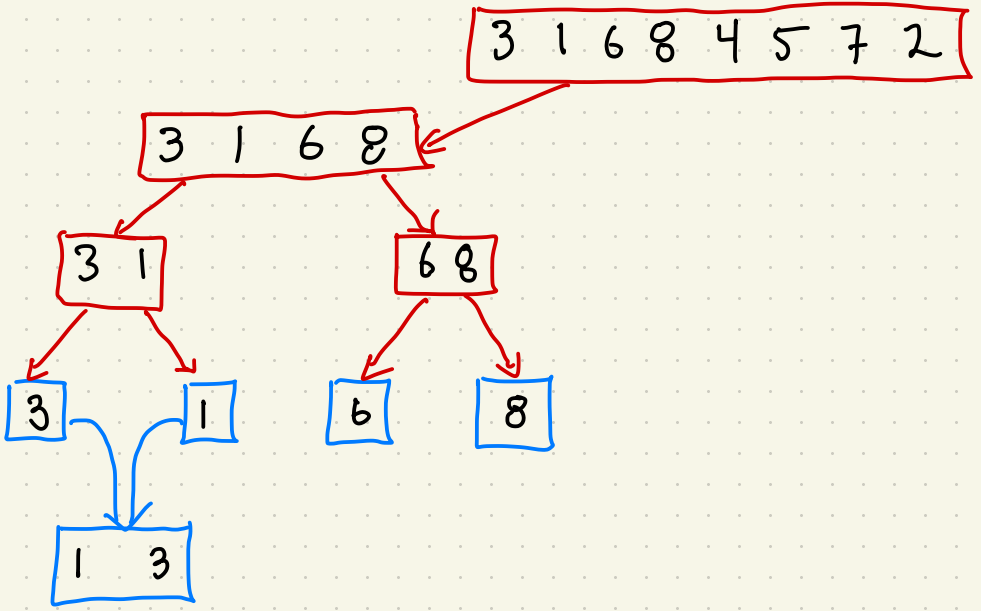

3 1 6 8 4 5 7 2
3 1 6 8
3 1
3

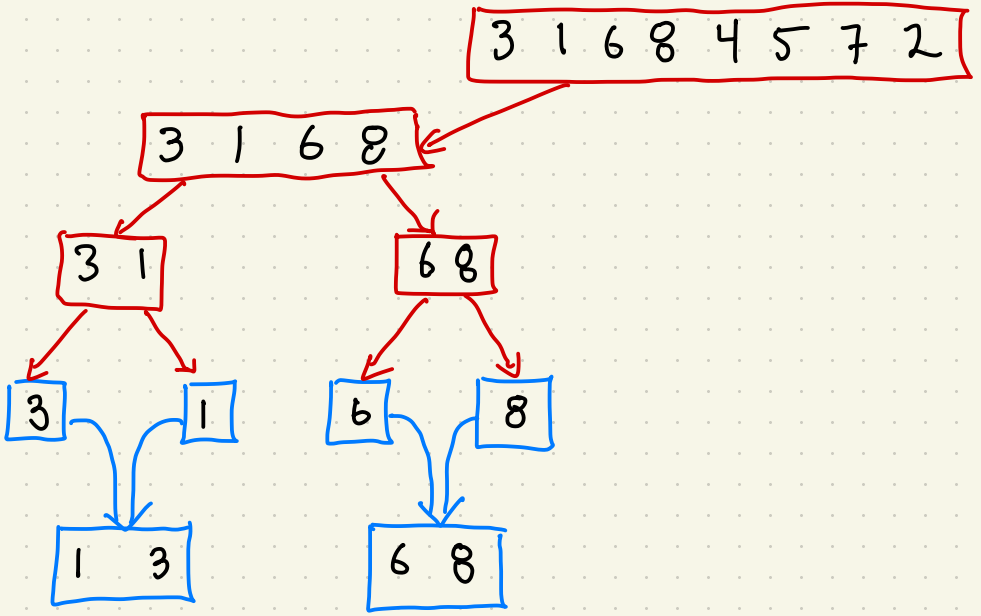

3 1 6 8 4 5 7 2
3 1 6 8
3 1
3
1

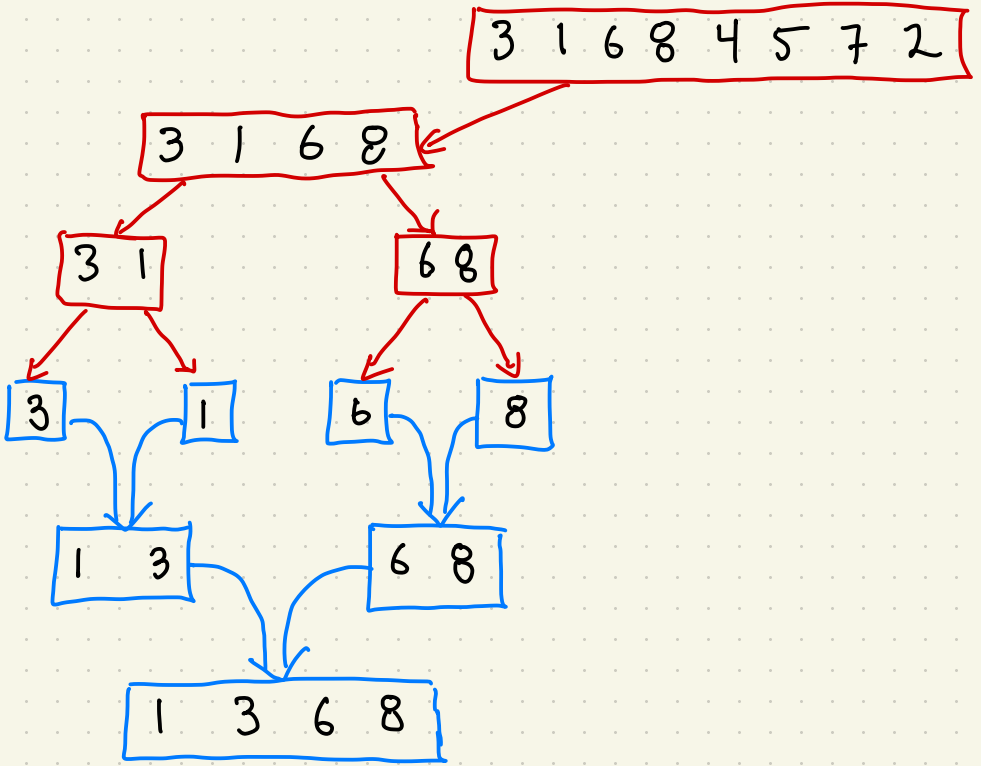

3 1 6 8 4 5 7 2
3 1 6 8
3 1
3
1
1 3

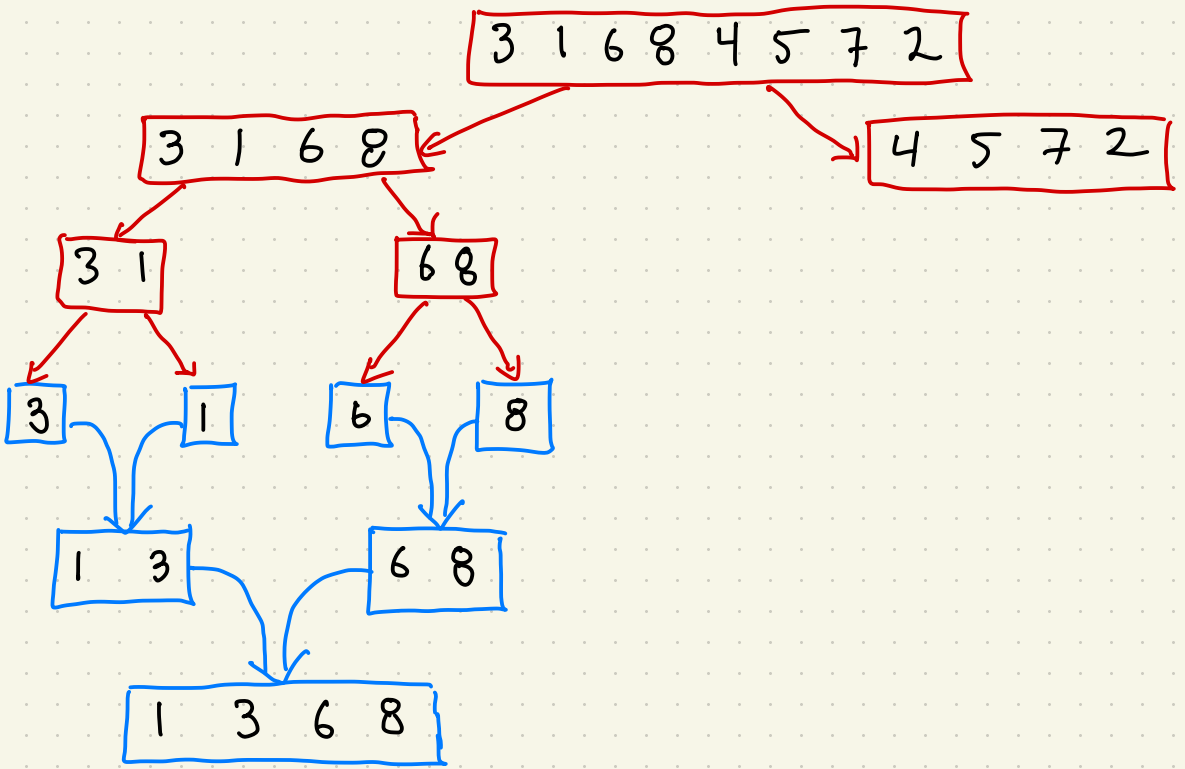

3 1 6 8 4 5 7 2
3 1 6 8
3 1
6 8
3
1
1 3

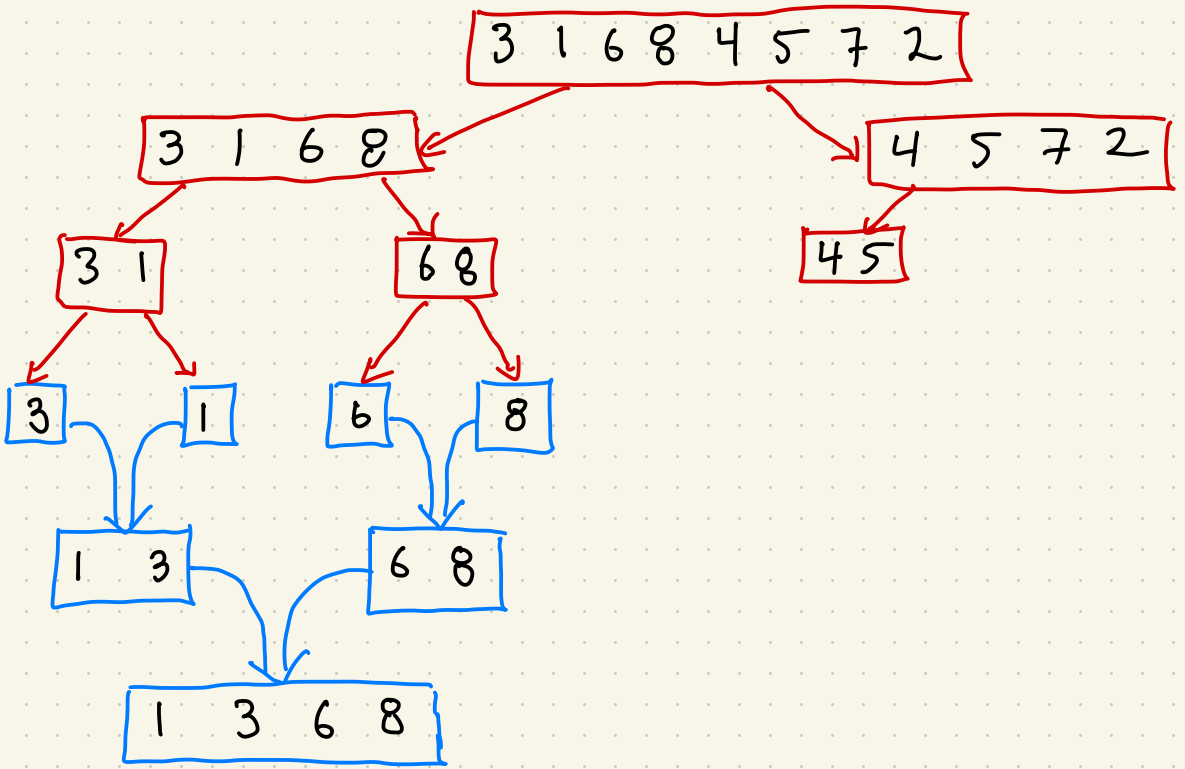

3 1 6 8 4 5 7 2
3 1 6 8
3 1
6 8
3
1
6
1 3

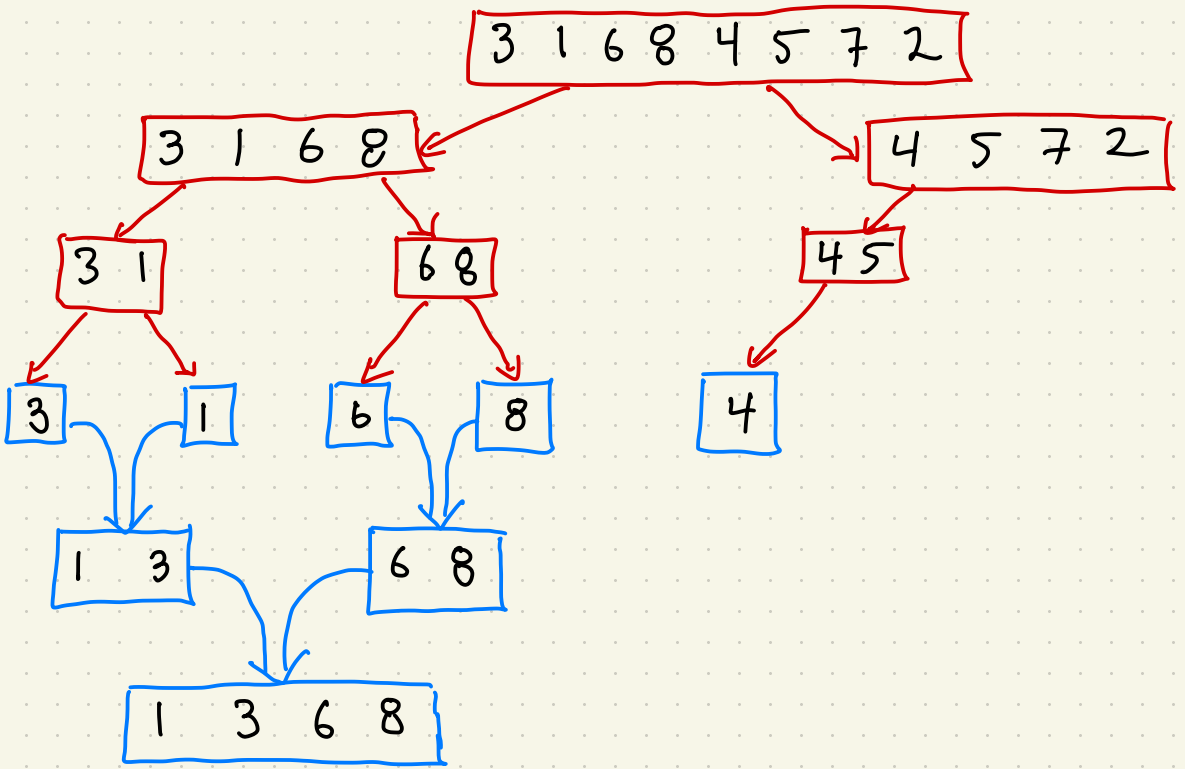

3 1 6 8 4 5 7 2
3 1 6 8
3 1
6 8
3
1
6
8
1 3

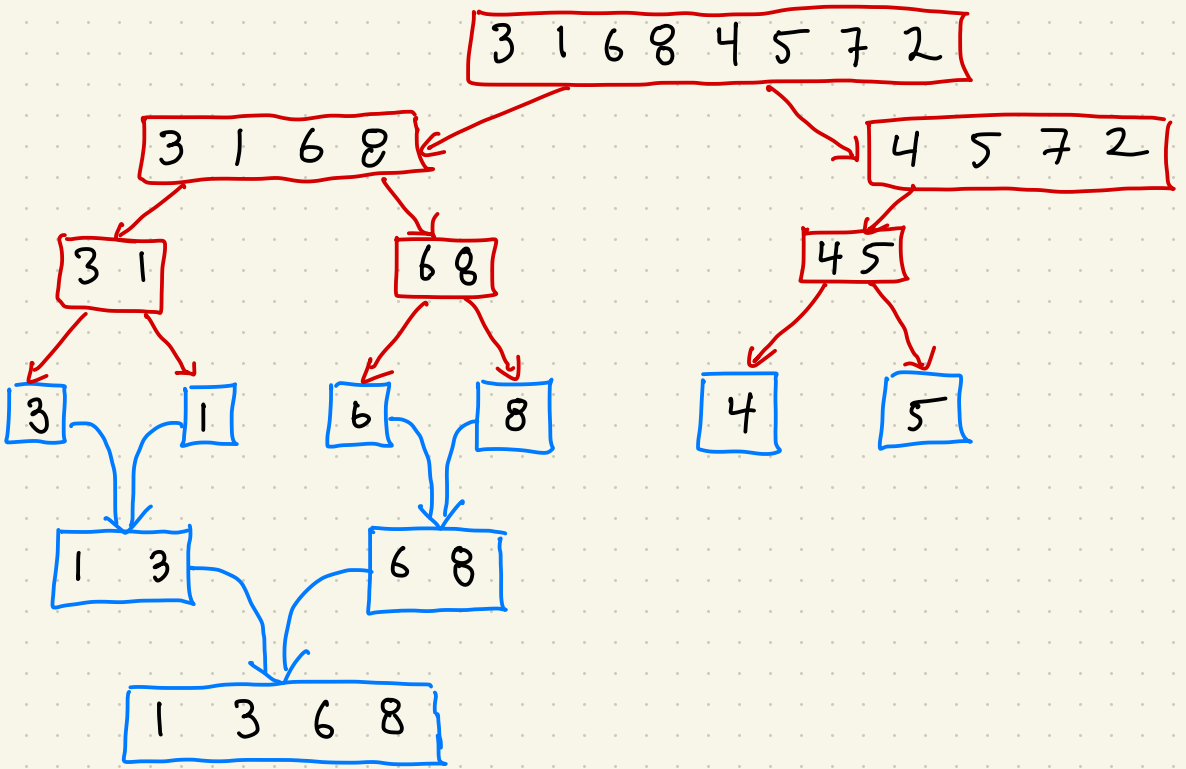

3 1 6 8 4 5 7 2
3 1 6 8
3 1
6 8
3
1
6
8
1 3
6 8

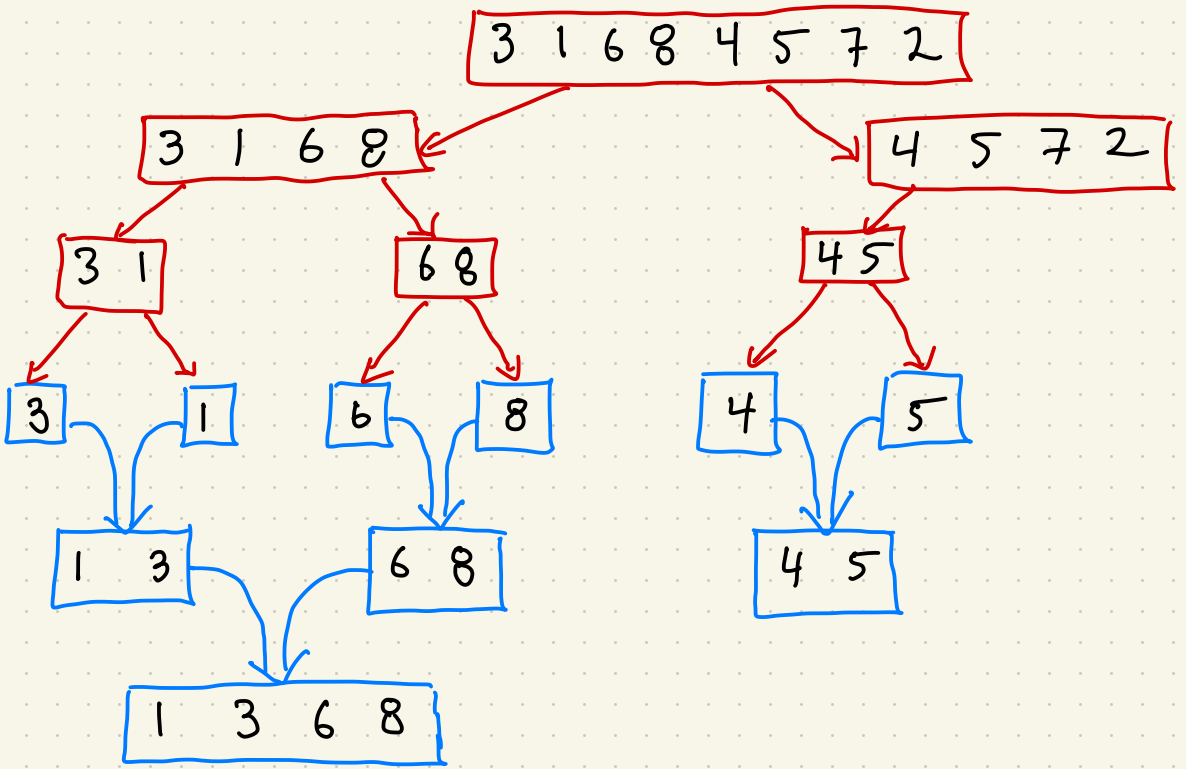

3 1 6 8 4 5 7 2
3 1 6 8
3 1
6 8
3
1
6
8
1 3
6 8
1 3 6 8

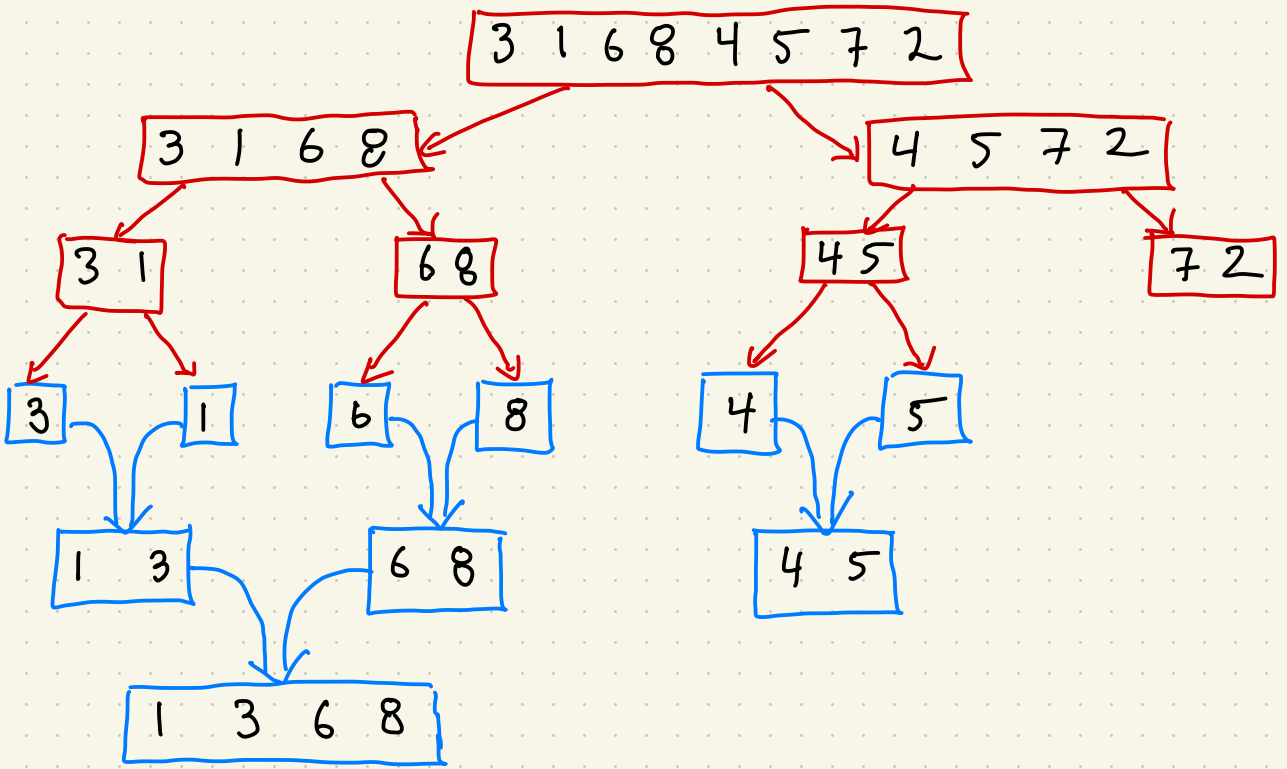

3 1 6 8 4 5 7 2
3 1 6 8
4 5 7 2
3 1
6 8
3
1
6
8
1 3
6 8
1 3 6 8

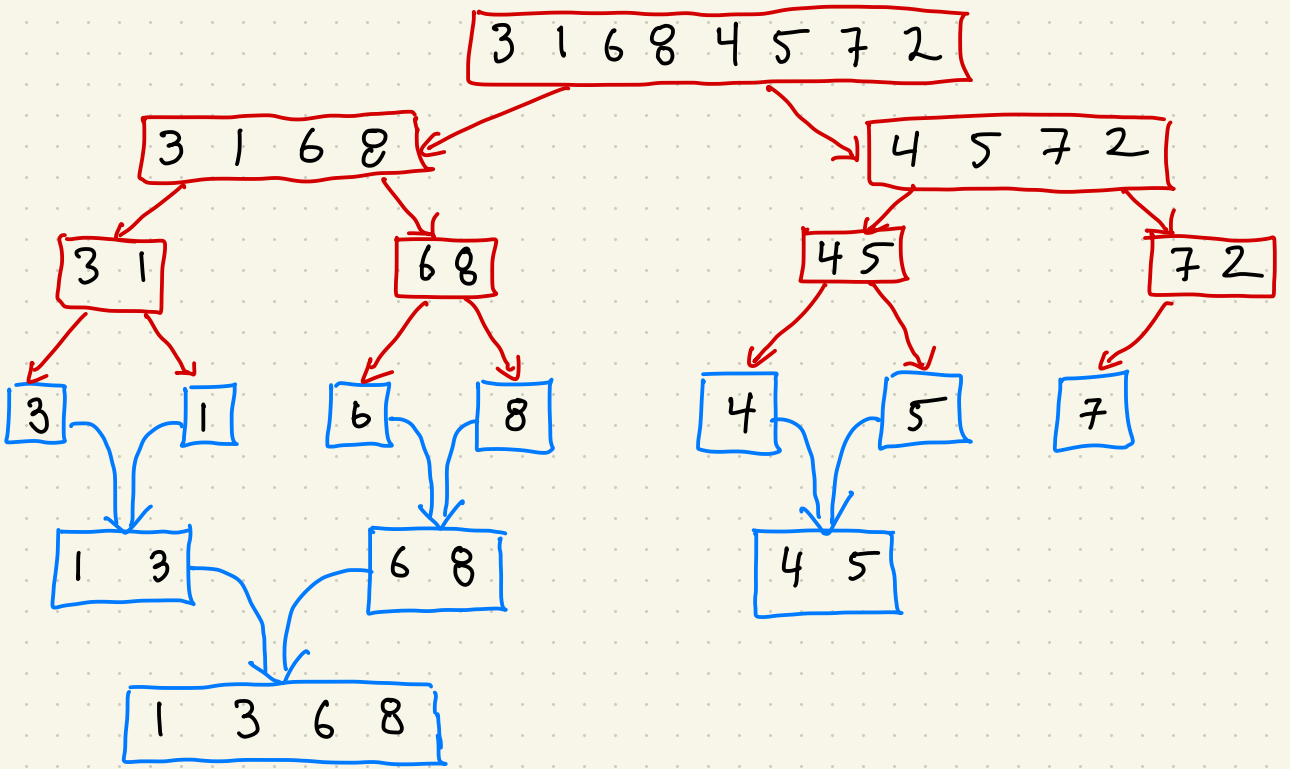

3 1 6 8 4 5 7 2
3 1 6 8
4 5 7 2
3 1
6 8
4 5
3
1
6
8
1 3
6 8
1 3 6 8

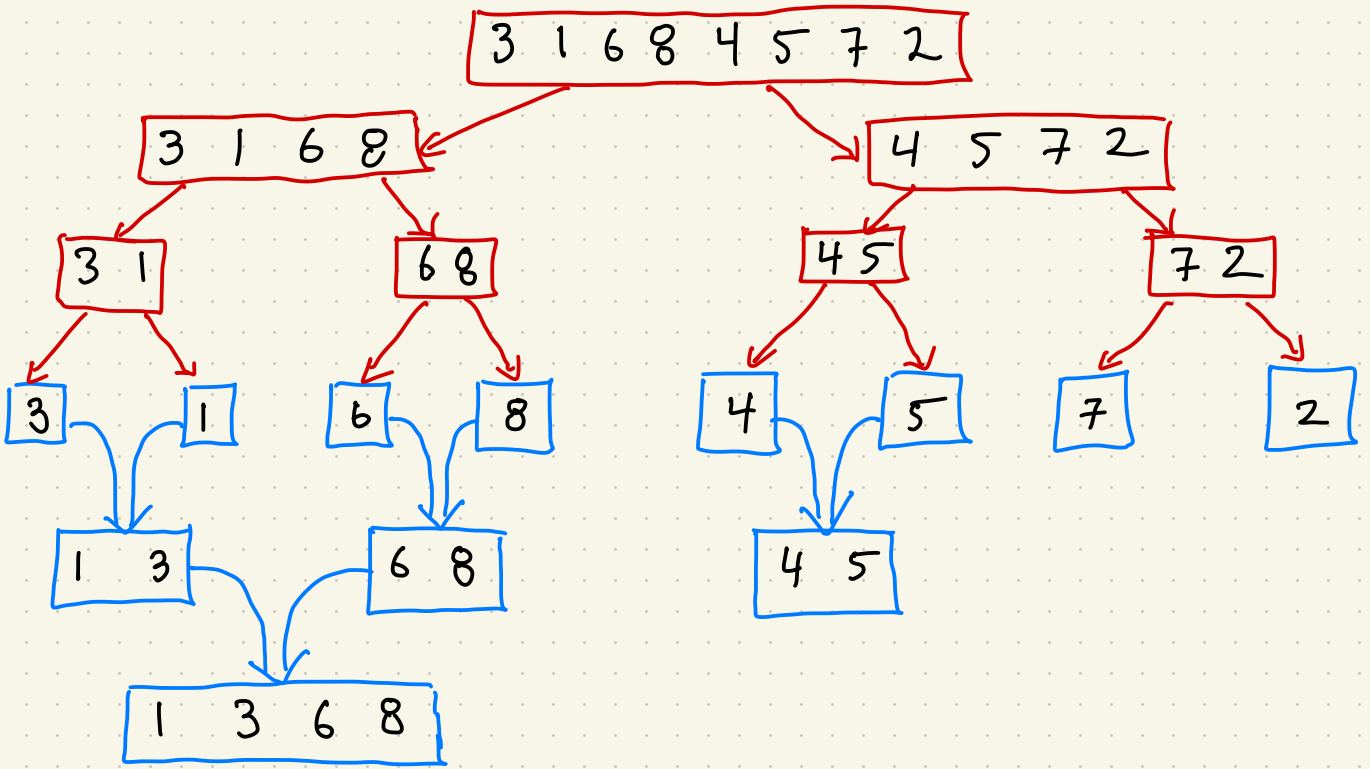

3 1 6 8 4 5 7 2
3 1 6 8
4 5 7 2
3 1
6 8
4 5
3
1
6
8
4
1 3
6 8
1 3 6 8

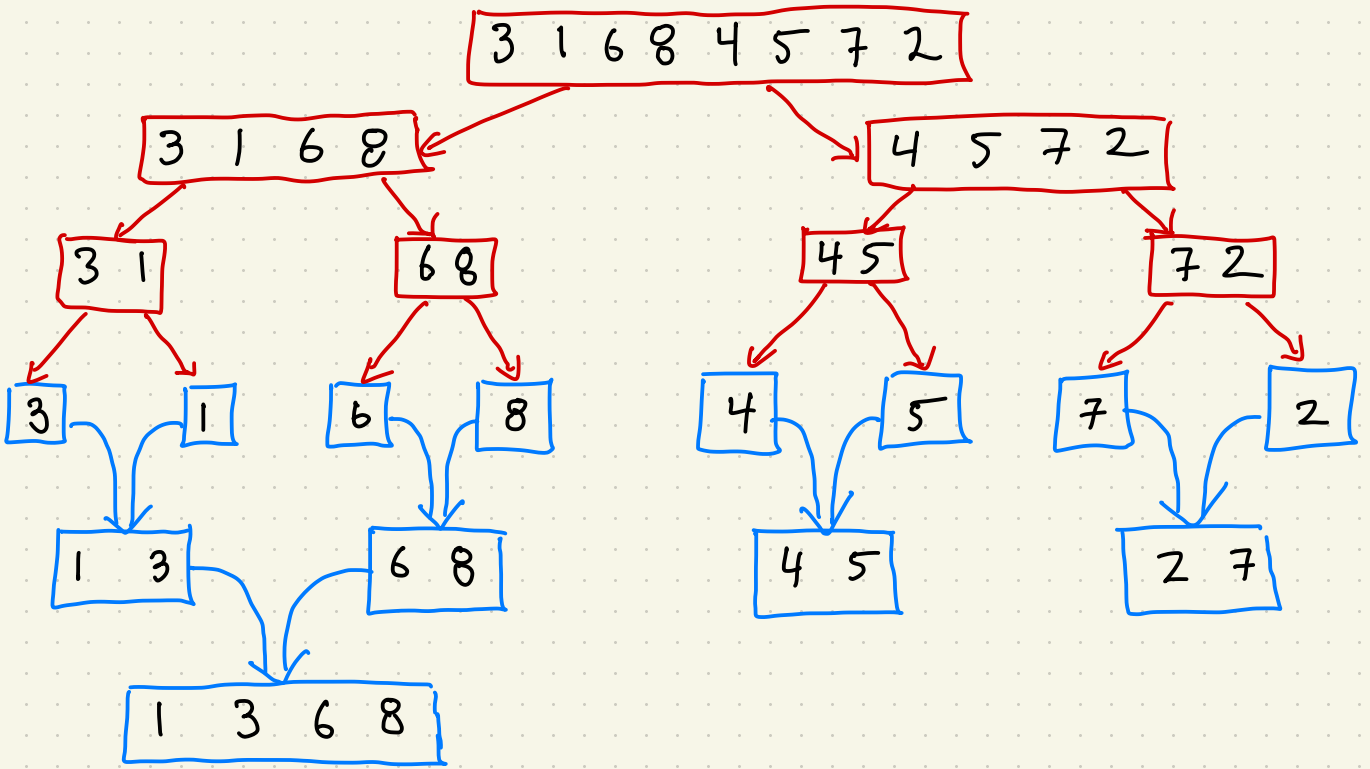

3 1 6 8 4 5 7 2
3 1 6 8
4 5 7 2
3 1
6 8
4 5
3
1
6
8
4
5
1 3
6 8
1 3 6 8

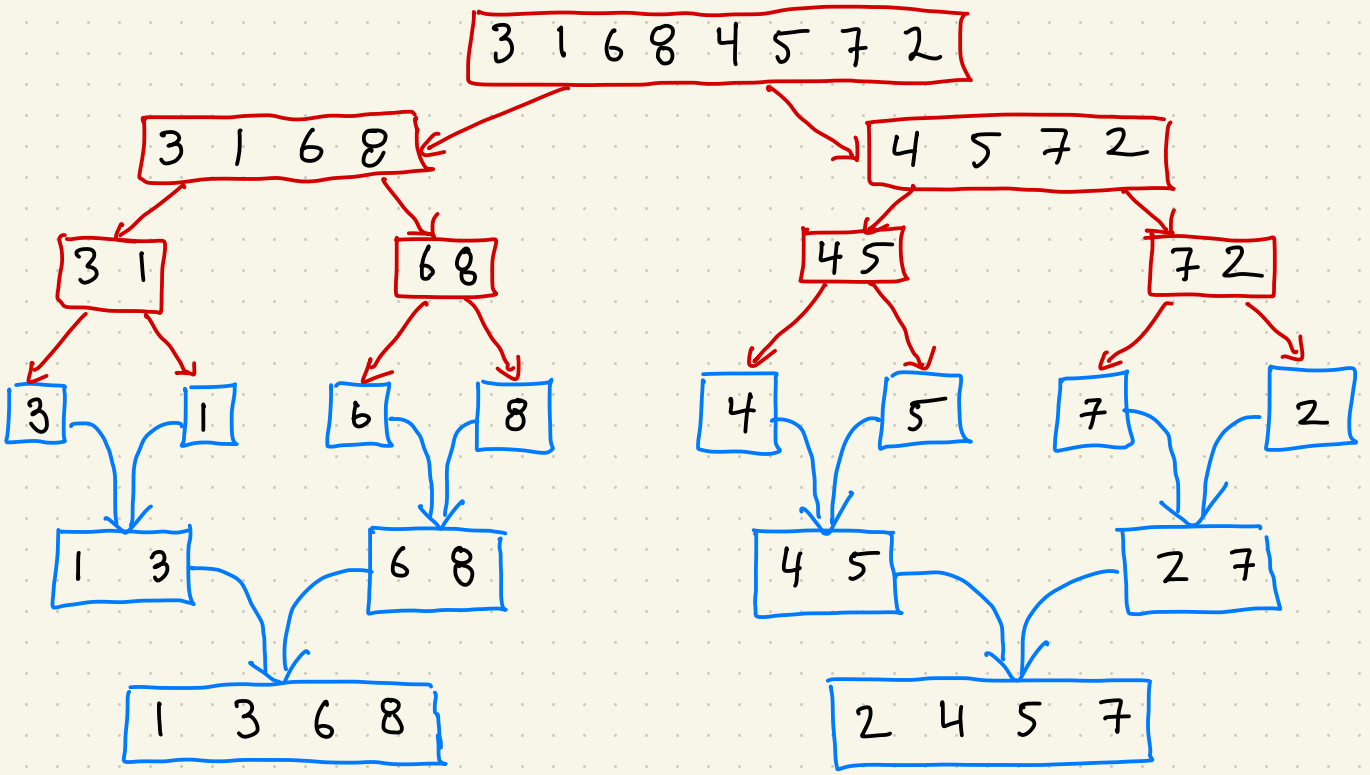

3 1 6 8 4 5 7 2
3 1 6 8
4 5 7 2
3 1
6 8
4 5
3
1
6
8
4
5
1 3
6 8
4 5
1 3 6 8

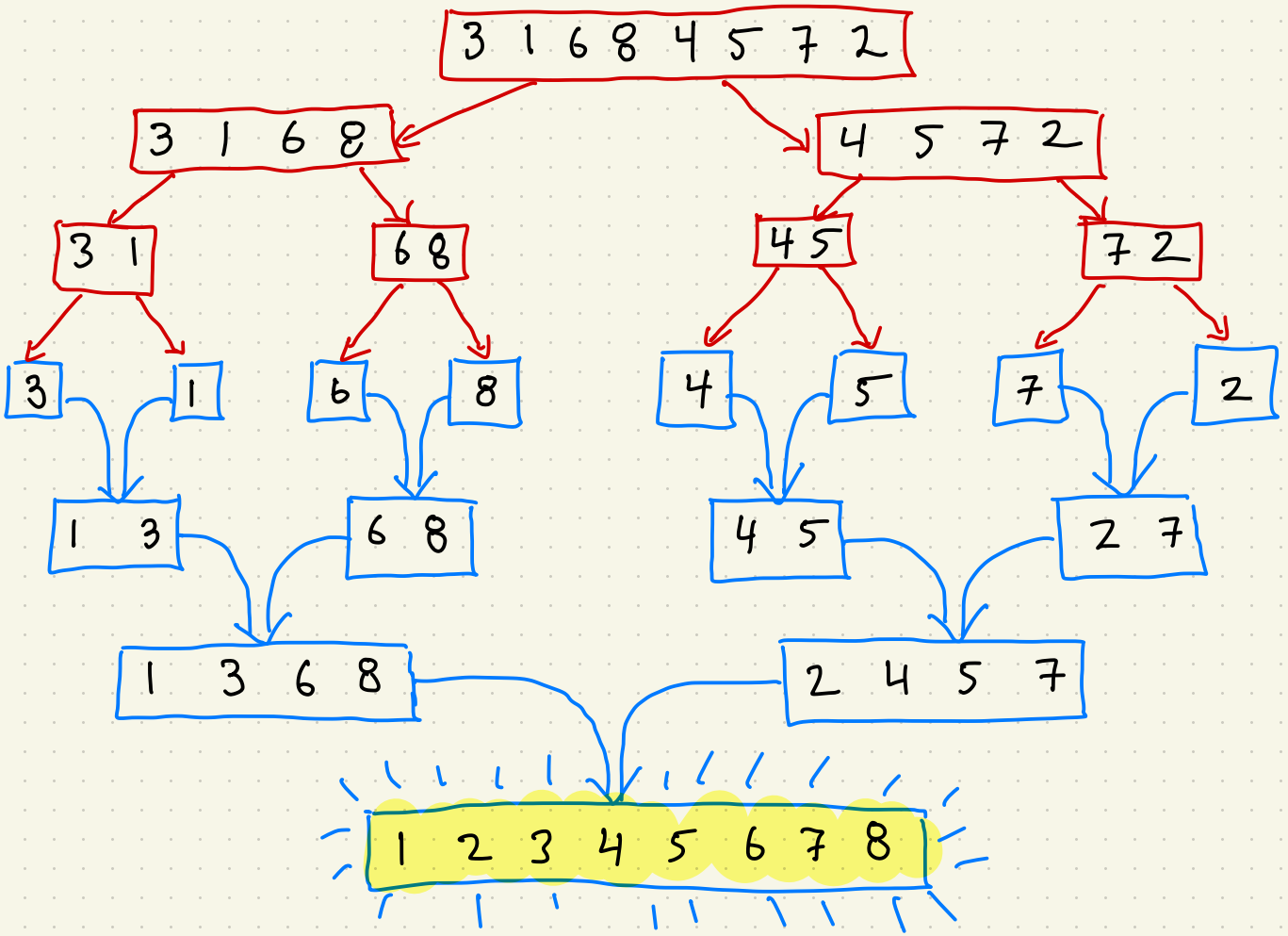

3 1 6 8 4 5 7 2
3 1 6 8
4 5 7 2
3 1
6 8
4 5
7 2
3
1
6
8
4
5
1 3
6 8
4 5
1 3 6 8


3 1 6 8 4 5 7 2
3 1 6 8
4 5 7 2
3 1
6 8
4 5
7 2
3
1
6
8
4
5
7
1 3
6 8
4 5
1 3 6 8


3 1 6 8 4 5 7 2
3 1 6 8
4 5 7 2
3 1
6 8
4 5
7 2
3
1
6
8
4
5
7
2
1 3
6 8
4 5
1 3 6 8

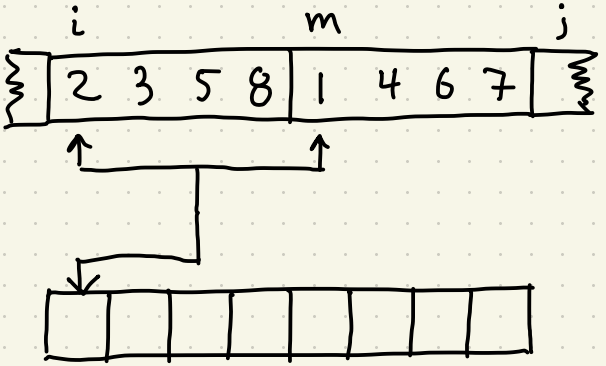

3 1 6 8 4 5 7 2
3 1 6 8
4 5 7 2
3 1
6 8
4 5
7 2
3
1
6
8
4
5
7
2
1 3
6 8
4 5
2 7
1 3 6 8


3 1 6 8 4 5 7 2
3 1 6 8
4 5 7 2
3 1
6 8
4 5
7 2
3
1
6
8
4
5
7
2
1 3
6 8
4 5
2 7
1 3 6 8
2 4 5 7

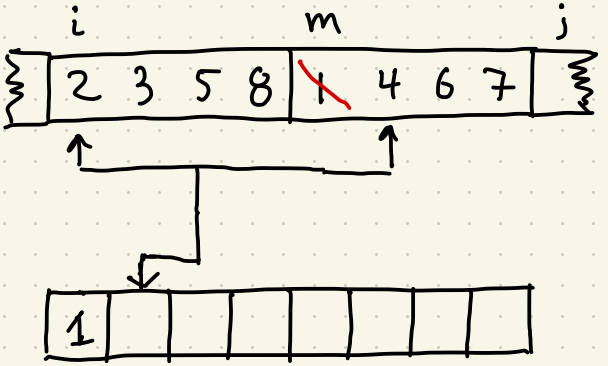

3 1 6 8 4 5 7 2
3 1 6 8
4 5 7 2
3 1
6 8
4 5
7 2
3
1
6
8
4
5
7
2
1 3
6 8
4 5
2 7
1 3 6 8
2 4 5 7
1 2 3 4 5 6 7 8

Merge Procedure

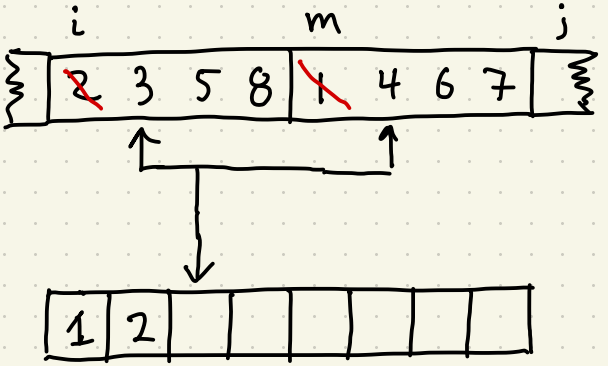
Merge



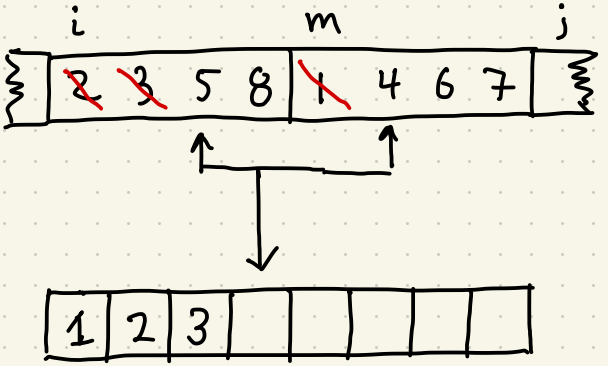
Merge



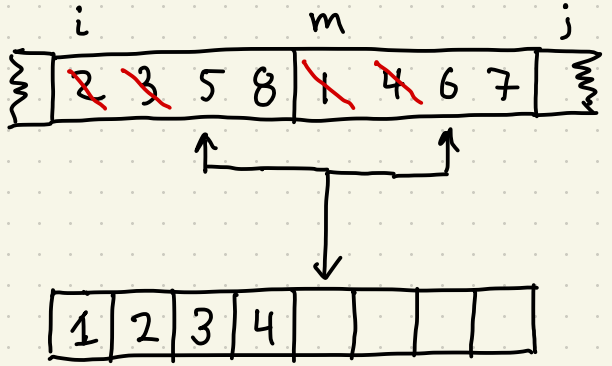
Merge



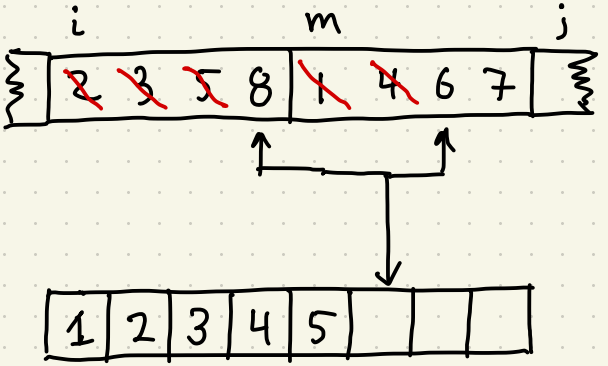
Merge



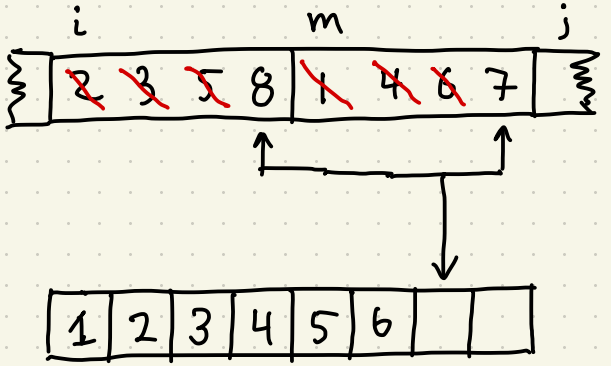
Merge



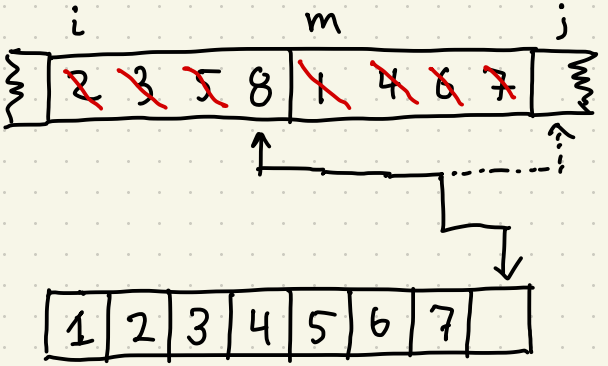
Merge



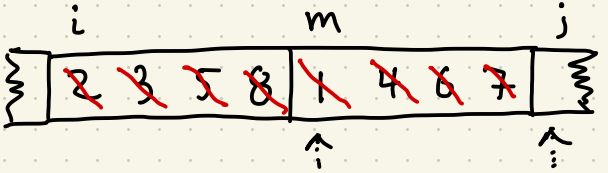
Merge



Merge



Merge



1	2	3	4	5	6	7	8

Correctness of MergeSort

Establish two claims:

Claim 1 (merge). If $a[i..m-1]$ and $a[m..j]$ are sorted, then after Merge(a, i, m, j), $a[i..j]$ is sorted.

- Argued on lecture ticket!

Claim 2. For any indices $i < j$, after calling MergeSort(a, i, j), $a[i..j]$ is sorted.

- How to argue this? (assume Claim 1 is true...)

Pseudocode Again

```
00  # sort values of a between indices i and j-1
01  MergeSort(a, i, j):
02    if j - i = 1 then
03      return
04    endif
05    m <- (i + j) / 2
06    MergeSort(a,i,m)
07    MergeSort(a,m,j)
08    Merge(a,i,m,j)
```

Inductive Claim

Consider MergeSort(a, i, j), define $k = j - i$ to be *size*

$P(k)$: for every $k' \leq k$, MergeSort(a, i, j) with size k' succeeds

Base case $k = 1$:

Inductive step $P(k) \implies P(k+1)$: