Lecture 09: Lower Bounds and RadixSort

COSC 311 *Algorithms*, Fall 2022

Overview

1. Sorting Lower Bound
2. Binary Radix Sort

Last Time

We asked:

> *Can we sort n elements faster than $O(n \log n)$?*

I claimed: not really?

1. Decision trees
   - encode executions of algorithms on *set of inputs*
   - "decision" = response to compare operation
2. Sorting task requires that algorithm distinguishes all pairs of permutations

   $1, 2, \ldots, n$

   - decision tree must have many leaves
3. Conclude: sorting requires $\Omega(n \log n)$ compare operations

$\geq c \, n \log n$

const

Decision Trees Again

$= $ all perm. of $1, 2, \dots, n$

$|S_n| = n!$

$= n \cdot (n-1) \cdots 1$

Follow execution of $\boxed{A}$ on all inputs from S_n

Define a binary tree:

1. each node corresponds to a single compare operation
2. each node has two children corresponding to two possible outcomes of compare

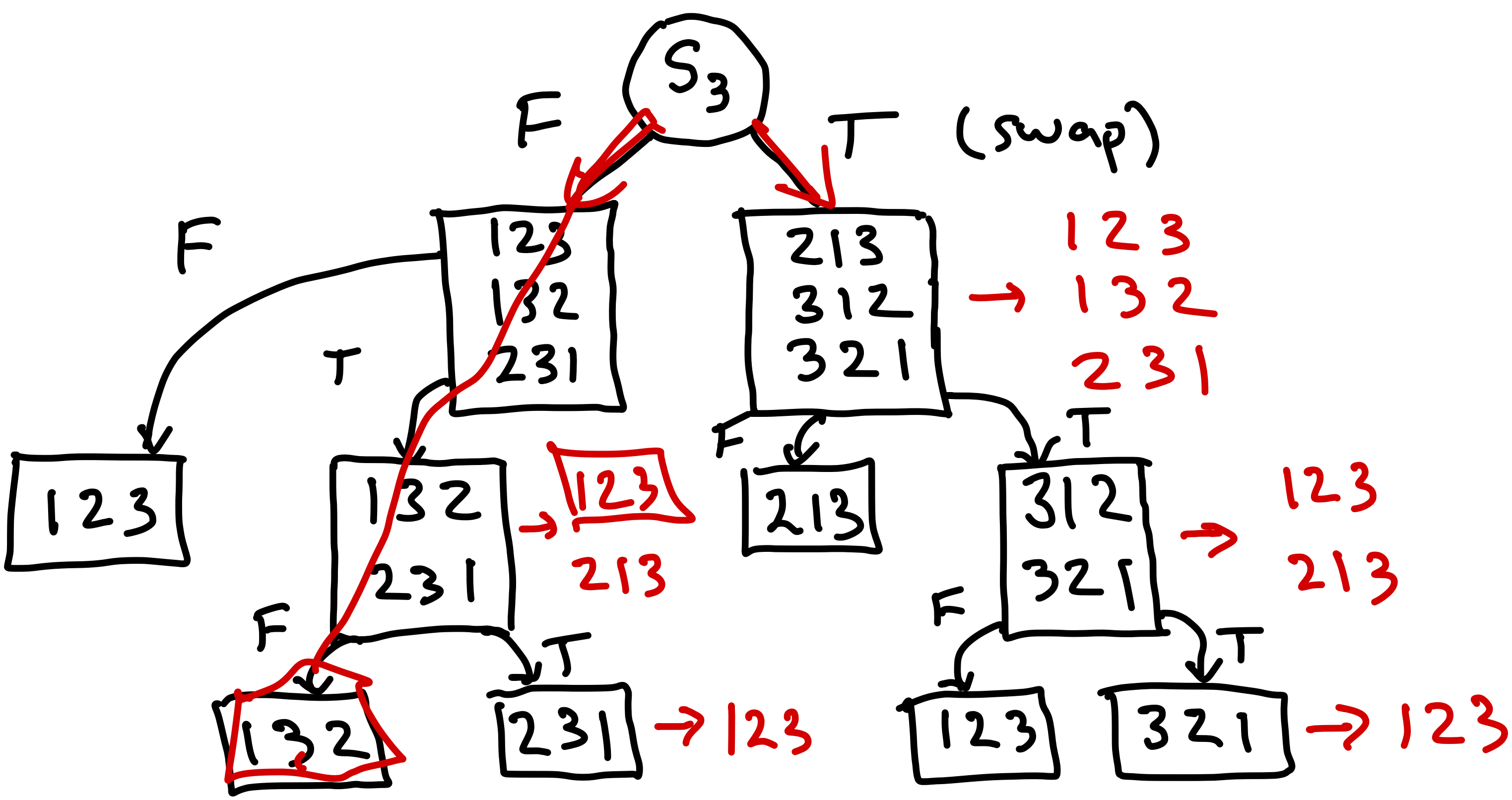
Form this tree for all comparisons made on all inputs in S_n

- label each node with inputs consistent with all compare outcomes

Decision Tree Example, n = 3

```
for i = 2 to n do
| j <- i
| while j > 1 and compare(a, j-1, j) do
| | swap(a, j, j-1);    j ← j-1
```

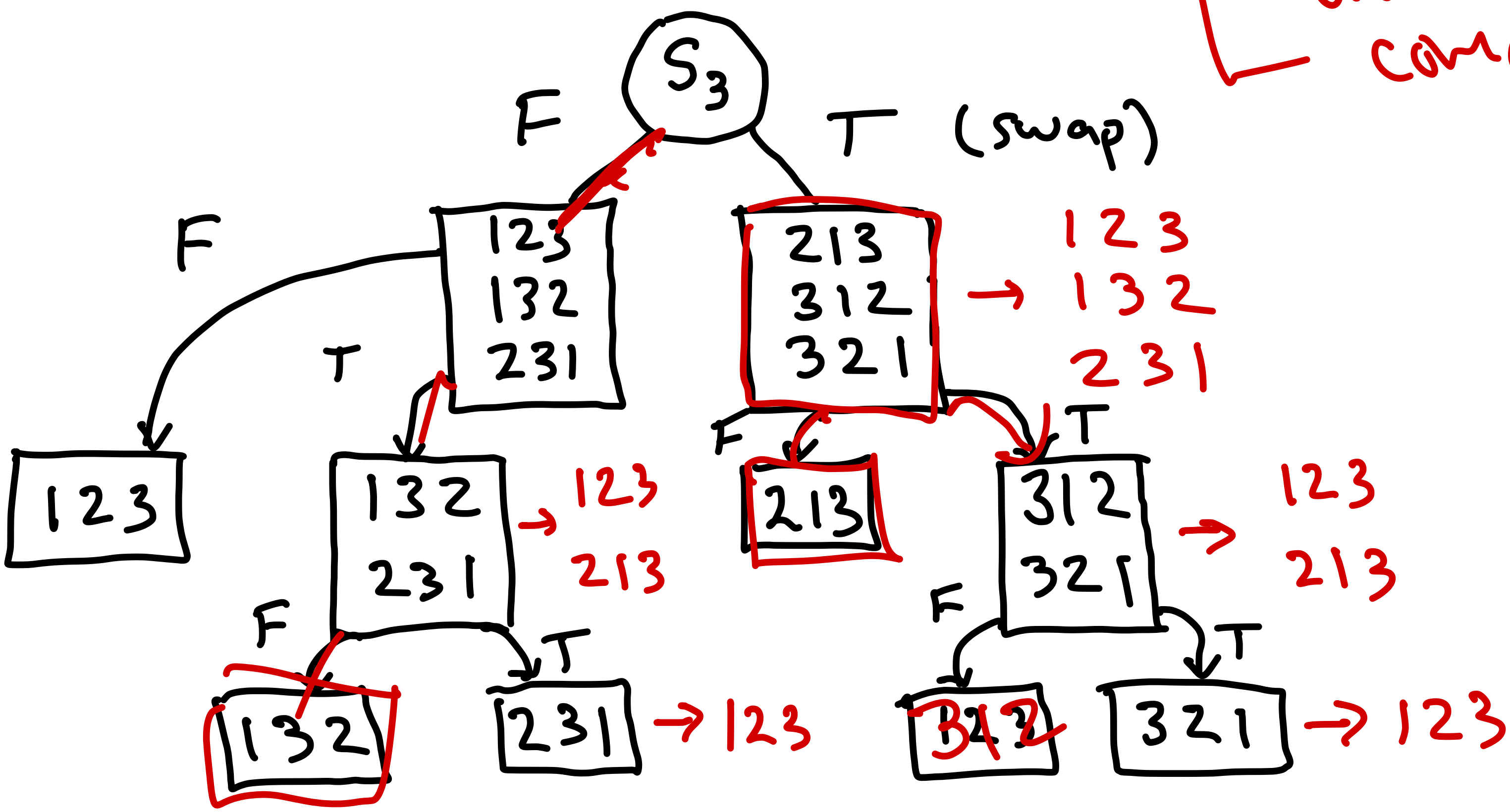
Compare(a,2,1)



Features of Decision Trees

1. *Depth* of tree = max # of compare operations on any input ← A calls compare(a,i,j)
2. Algorithm $\boxed{\text{distinguishes}}$ inputs a and b $\iff$ a and b label different leaves

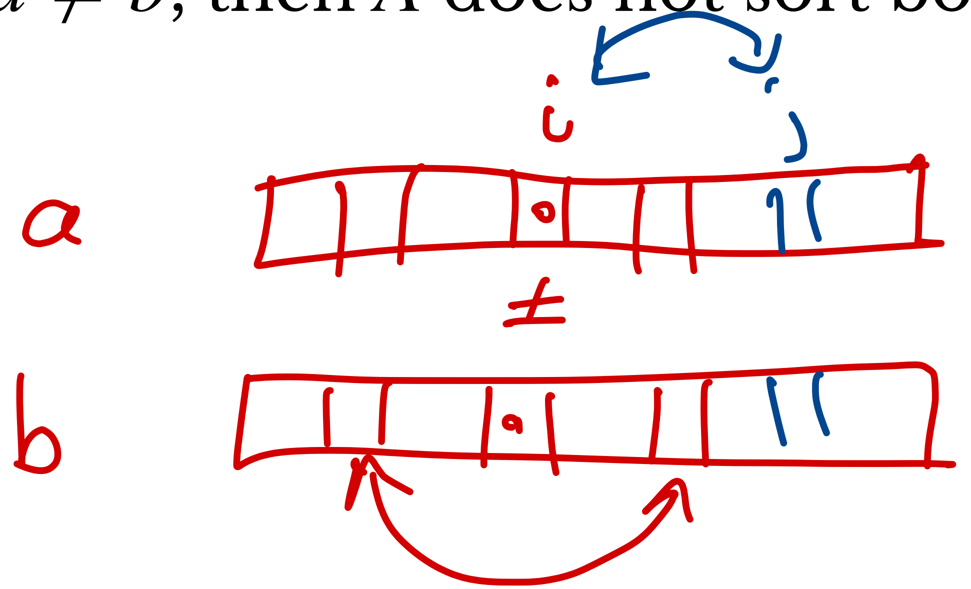
≠ and compare(b,i,j)



Indistinguishability Claim

Claim. If A does not distinguish permutations a and b with $a \neq b$, then A does not sort both a and b.

Why?



If $a \neq b$ before swap, then $a \neq b$ after.

$\Downarrow$

after "sorting" $a \neq b$

but not possible

If sorted

$a = 1, 2, \ldots, n$

$b = 1, 2, \ldots, n$

$\Rightarrow$ $a = b$

$|S_n| = n!$

Indistinguishability Consequence

Consequence. If A sorts all arrays of size n, then every leaf of A's decision tree is labeled with a single permutation array.

Why?

If A sorts, A distinguishes all pairs of perms

$\Rightarrow$ all permutations sent to distinct leaves.

$\Rightarrow$ $n!$ leaves.

How Big is Decision Tree?

How many leaves must a correct decision tree have?

$\geq n!$

How deep must decion tree be?

Bin. tree w/ depth d

how many leaves?

$n! \leq \#$ leaves $\leq 2^d$

$\Longrightarrow \quad n! \leq 2^d \Rightarrow d \geq \log n!$

Putting it All Together

$\log(ab) = \log(a) + \log(b)$

1. Consider any sorting algorithm A
2. Fix inputs S_n = permutations of size n
3. Decision tree must have at least $|S_n| = n!$ leaves
4. Decision tree must have depth at least $\log n!$
5. A must perform at least $\log n!$ comparisons

$a > b \Rightarrow \log a > \log b$

Claim. $\boxed{\log n!} = \Omega(n \log n)$

$\geq c\, n \log n$

$$\log(n!) = \log\left(n(n-1)(n-2)\cdots 2\cdot 1\right)$$

$$= \boxed{\log n + \log(n-1) + \cdots} + \log 2 + \log 1$$

stop at $\log(\frac{n}{2})$

$$\geq \underbrace{\log \frac{n}{2} + \log \frac{n}{2} + \cdots + \log \frac{n}{2}}_{n/2}$$

$$= \frac{n}{2} \log \frac{n}{2} \leftarrow \Omega(n \log n)$$

Conclusion

Theorem. Any algorithm that sorts all permutations of size n using only compare and swap operations requires $\Omega(n \log n)$ comparisons.

Question

The $\Omega(n \log n)$ lower bound critically assumes that array is only accessed and modified with compare and swap.

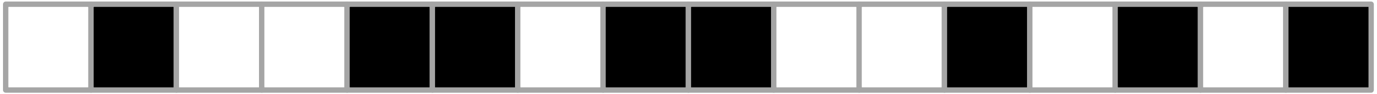
- What if we have more refined access?
- What if we see binary representation of elements?

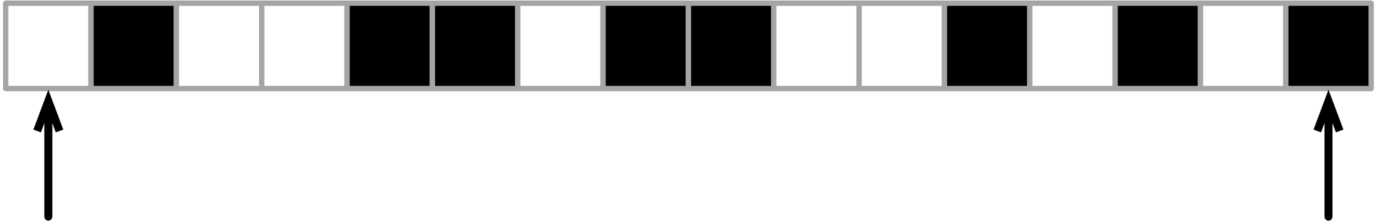
Sorting Binary Values

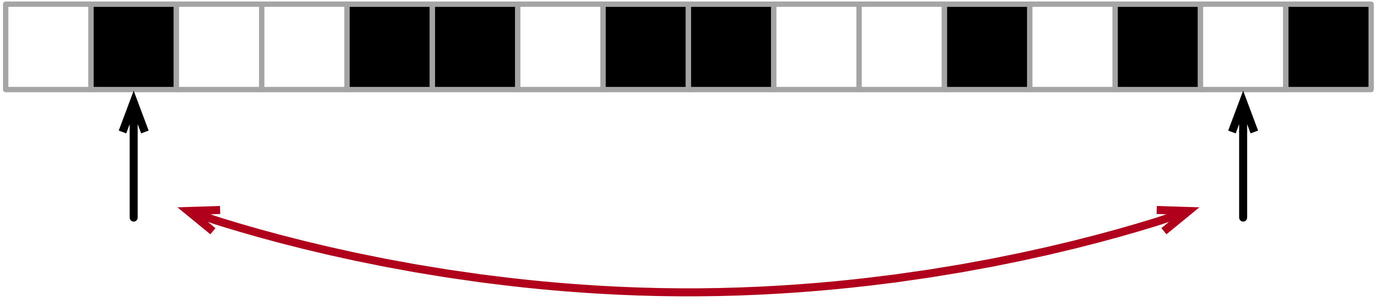
Observation. If a consists of only 0s and 1s, we can sort a in $O(n)$ time.

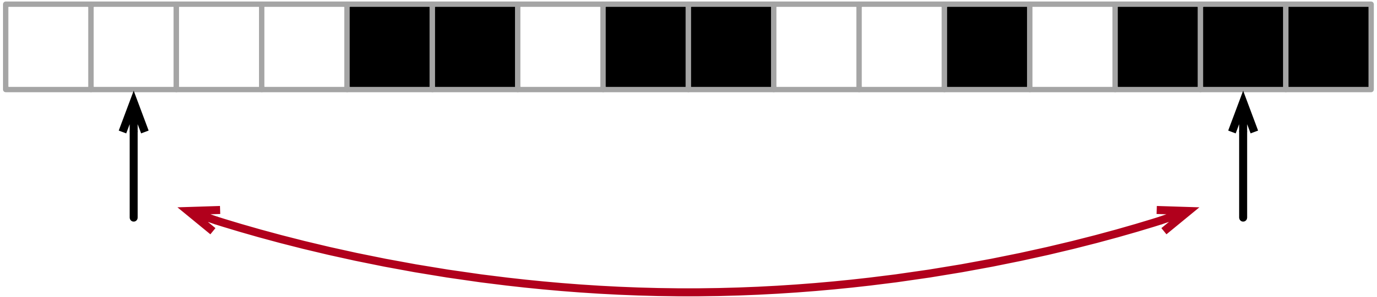
How? Split from quicksort

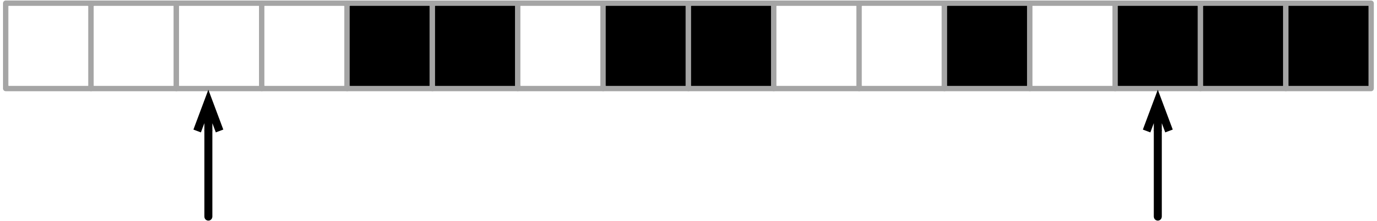
Binary Split Illustration

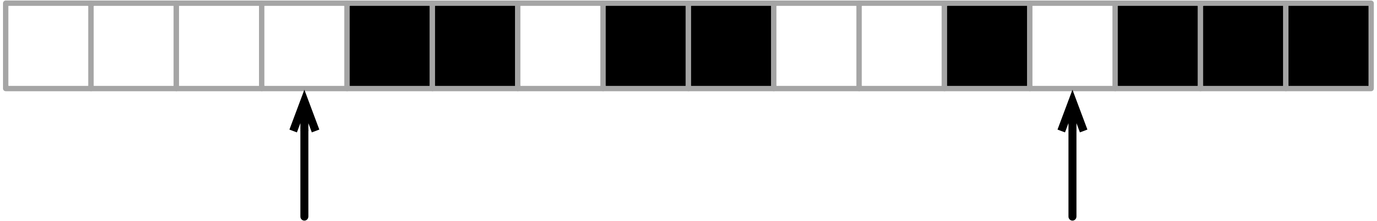


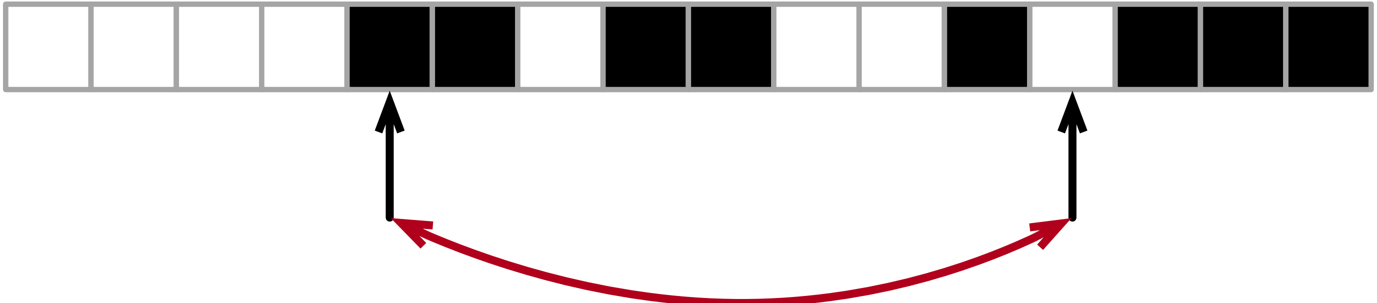


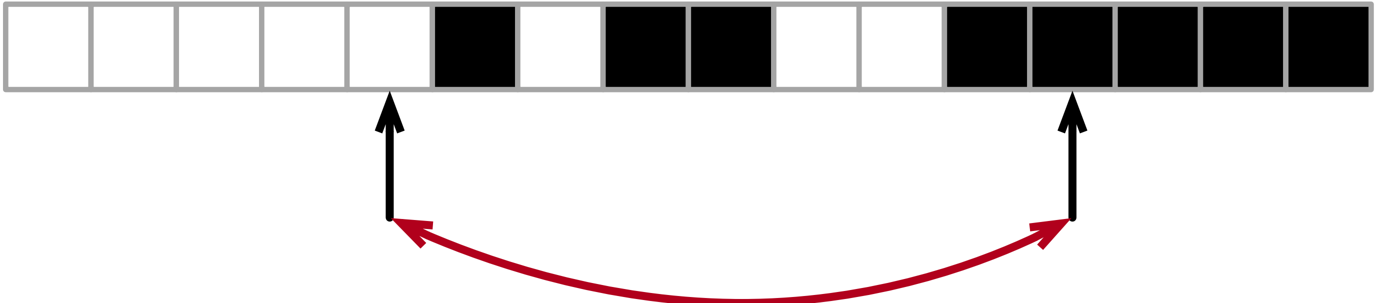


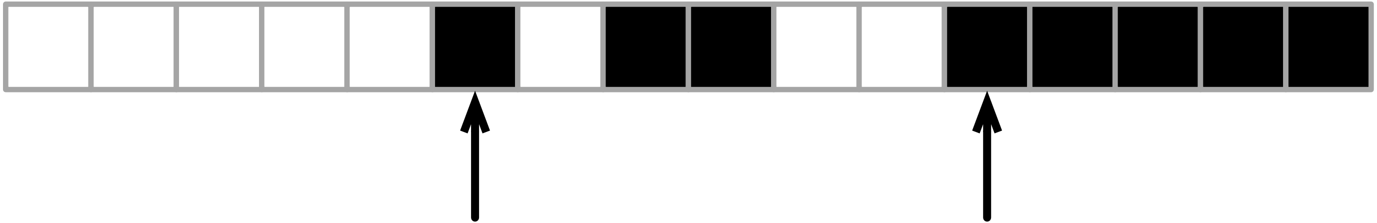


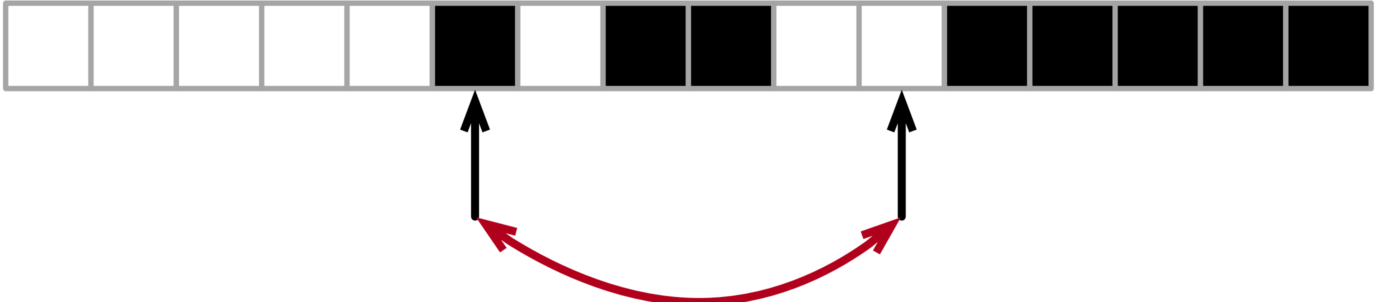


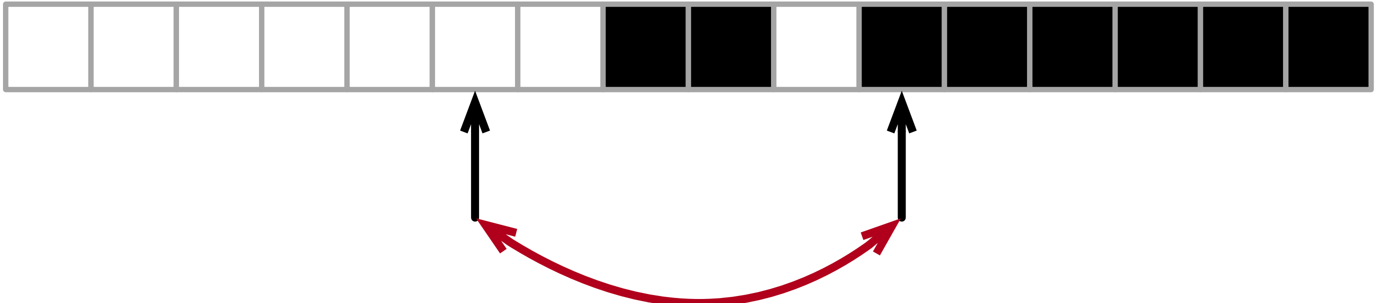


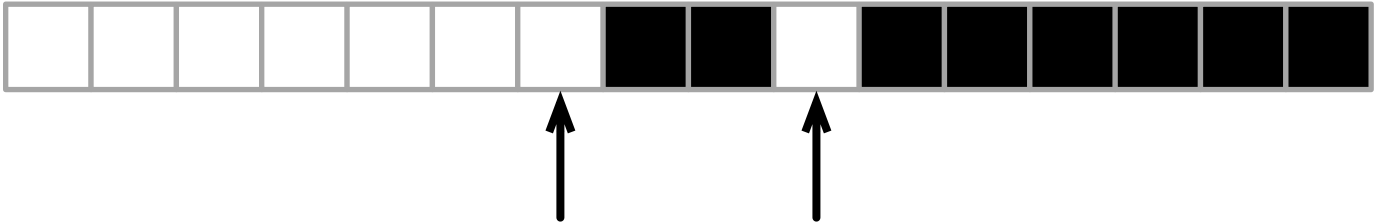


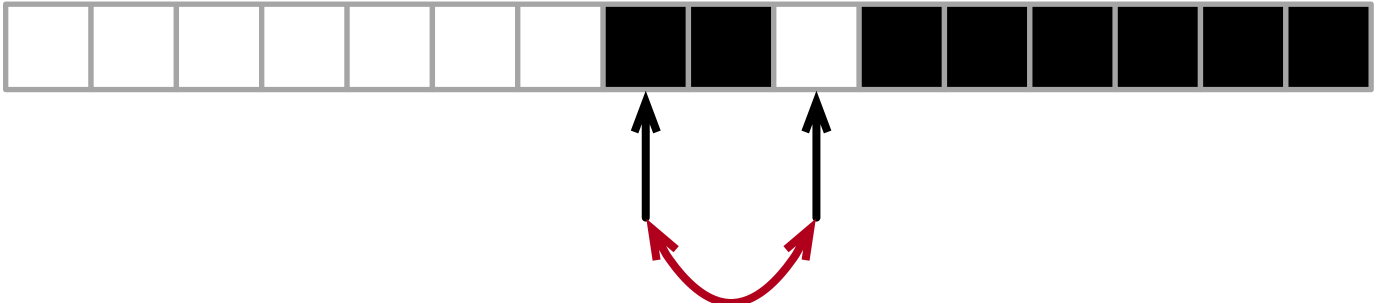


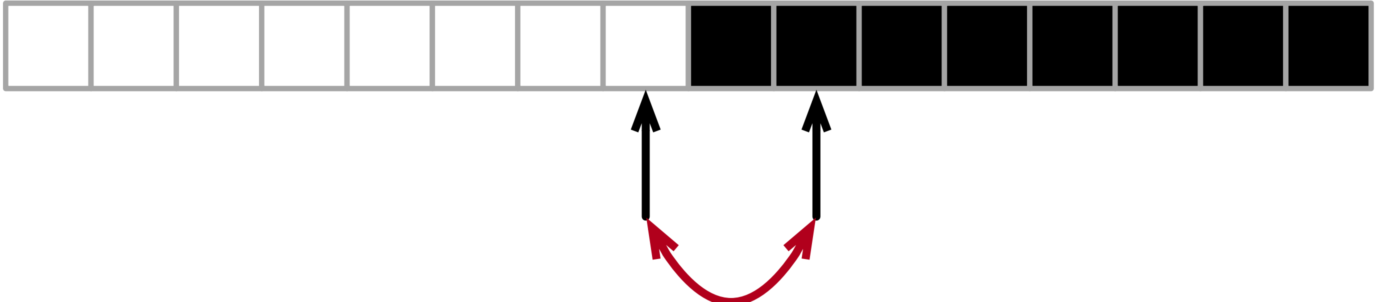


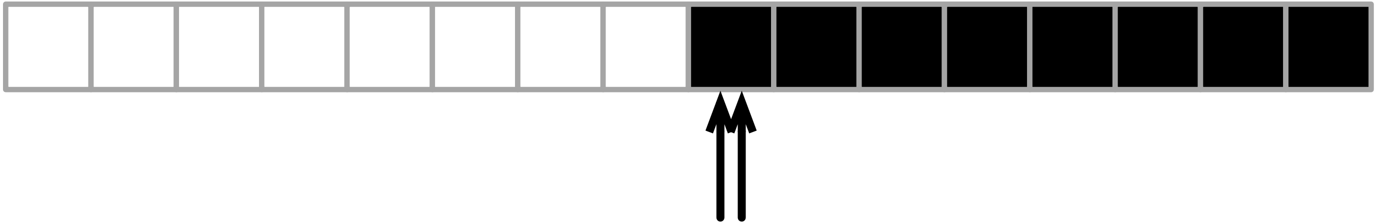












Sorting Numbers Represented in Binary

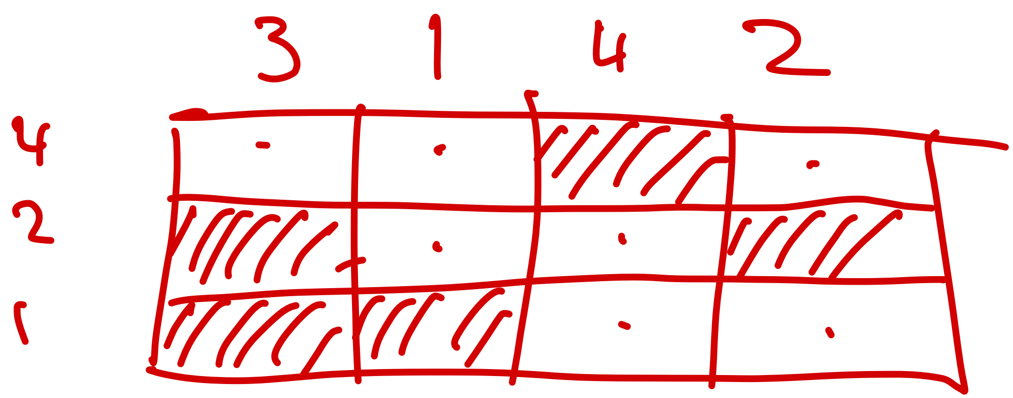
Assume a consists of n numbers, each with binary representation of B bits

$a = [3\ 1\ 4\ 2]$

- $a[i]$ = number at index i
- $a[i][j]$ = jth bit of $a[i]$

$a[i][1]$ 1s bit
$a[i][2]$ 2s bit
$\vdots$

$3 = 011$
$1 = 001$
$4 = 100$
$2 = 010$

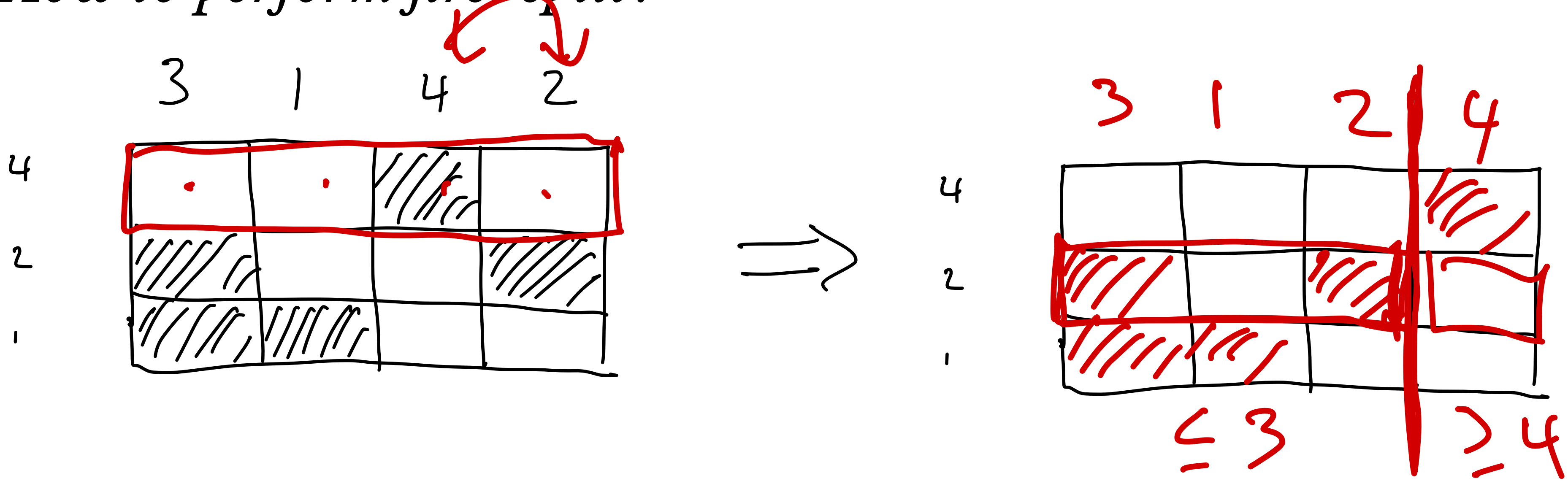


A Sorting Idea

Inspired by QuickSort:

- split array by “value”
- rather than comparing values, compare individual bits

How to perform first split?



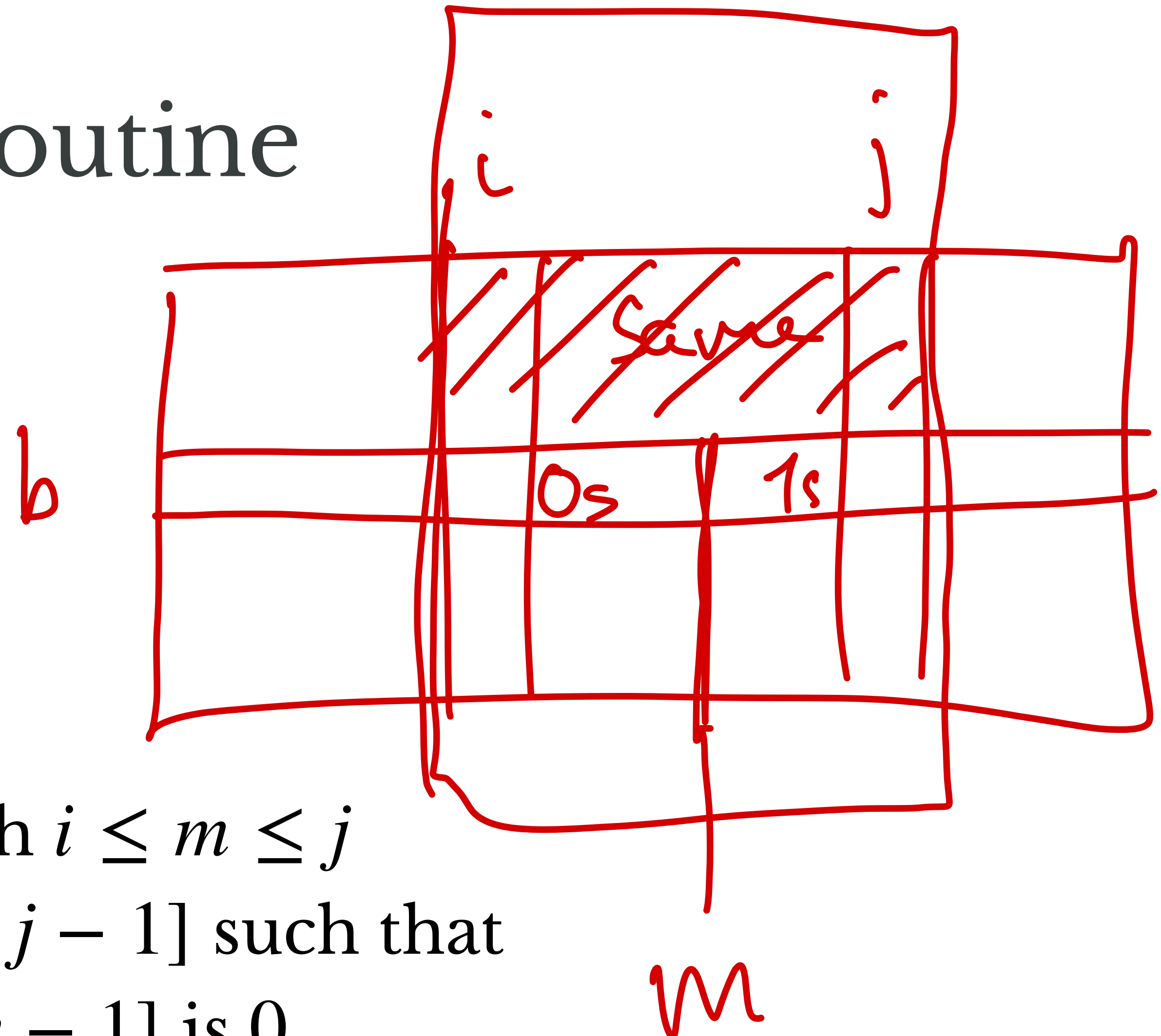
Bit Split Subroutine

Input:

- array a
- indices $i < j$
- bit index b

Behavior:

- return value m with $i \leq m \leq j$
- split values of $a[i\,..\,j-1]$ such that
  - bth bit of $a[i\,..\,m-1]$ is 0
  - bth bit of $a[m\,..\,j-1]$ is 1



BitSplit Pseudocode

```
BitSplit(a, i, j, b):
  left <- i, right <- j
  while left < right do:
    if a[left][b] = 1 and a[right][b] = 0 then
      swap(a, left, right)
      left++, right--
    else
      if a[left][b] = 0 then left++
      if a[right][b] = 1 then right--
    endif
  endwhile
  if a[right][b] = 0 then return right+1 else return right
```

RadixSort

Bits in each value

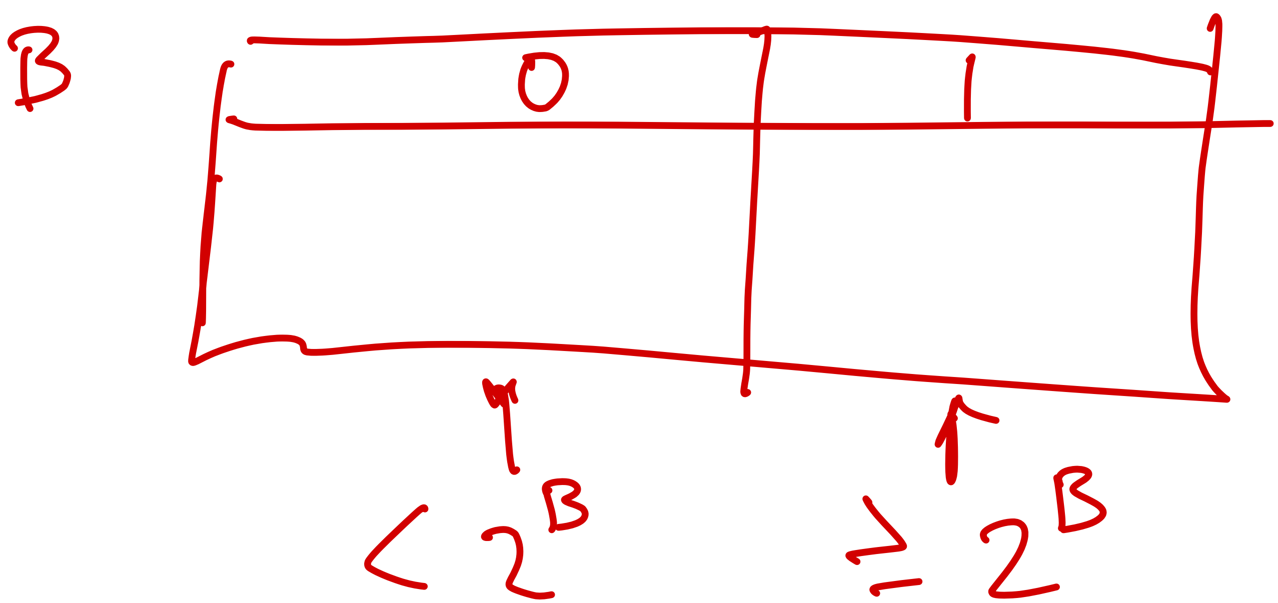
```
RadixSort(a, B):                        # B is number of bits
  RadixSort(a, 1, size(a)+1, B)

RadixSort(a, i, j, b):
  if j - i <= 1 then
    return
  endif
  m <- BitSplit(a, i, j, b)
  RadixSort(a, i, m, b-1)
  RadixSort(a, m, j, b-1)
```

Illustration

Why Does RadixSort Work?

BitSplit(a, i, n, B)



What is RadixSort Running Time?

$$\Theta(B \cdot n)$$

Exercise: Convince self this is right.

Next Time

Arithmetic!

- multiplication