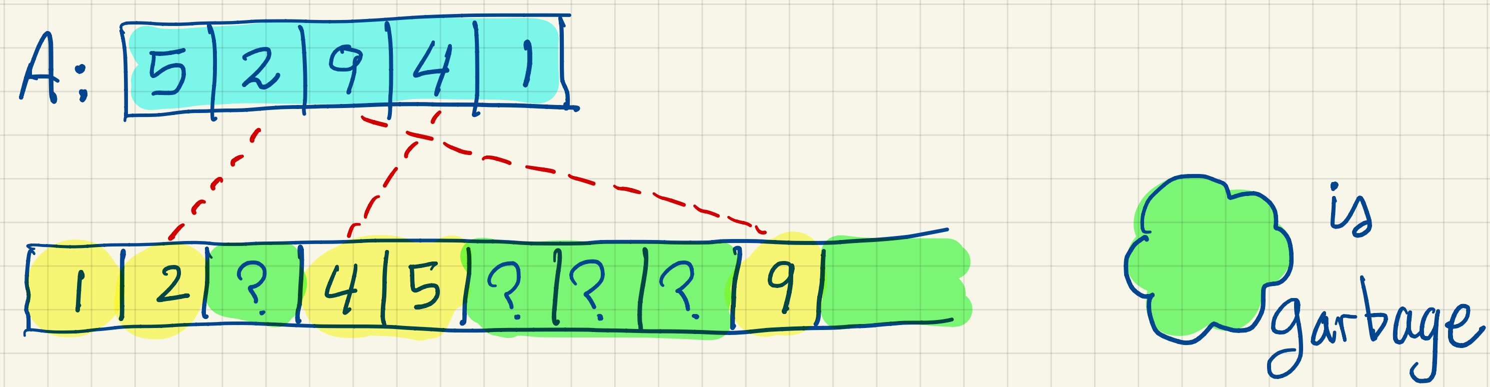
Sorting in Linear time

- Assume elements in $A[1 \ldots n]$ are distinct pos. integers
- Consider the following algorithm.

$$
\begin{array}{l}
\textbf{for } i \leftarrow 1 \textbf{ to } n \\
\qquad B[A[i]] \leftarrow A[i]
\end{array}
$$

example:



- Kind of sorted! in Linear time!!
- Counting Sort achieves a similar effect with a little more sophistication.

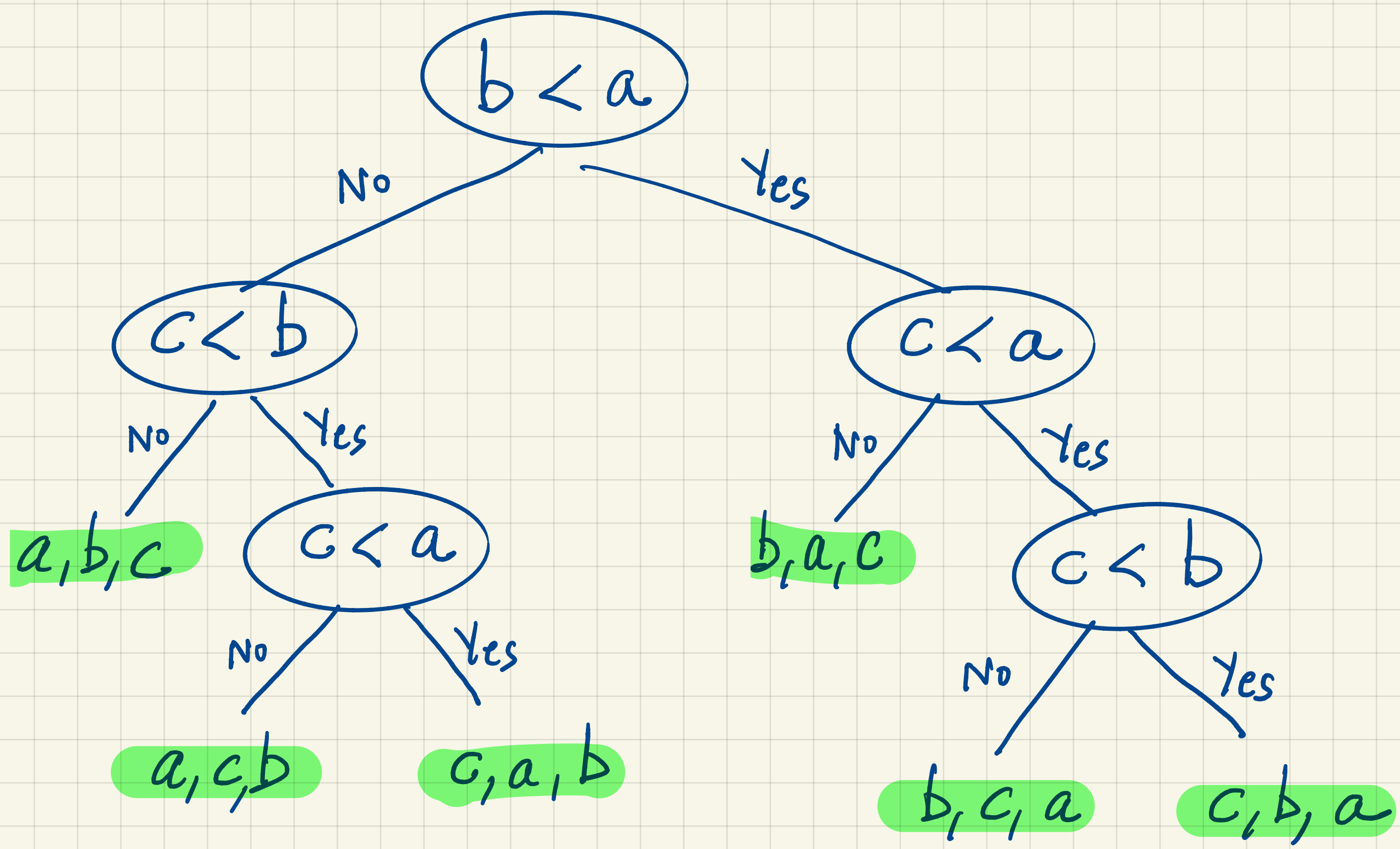
But why were we able to sort in linear time?

We used the values in something other than comparison!

A comparison based sorting algorithm runs in $\Omega(n \log n)$ time.

Decision tree proof.

Insertion sort $\quad$ | a | b | c | $\quad n=3$



Every comparison based alg. can be modeled as a binary decision tree with $n!$ leaves

$$\#\text{leaves} \leq 2^h \Rightarrow n! \leq 2^h \Rightarrow h \geq \log_2 n!$$

$$h = \Omega(n \log n)$$

Counting Sort (A, B, n, k) ▷ elements $\in \{0, 1, 2, \ldots, k\}$

```
Counting Sort(A, B, n, k)
  for i ← 0 to k
      do C[i] ← 0                          ▷ Initialize count
  for j ← 1 to n
      do C[A[j]] ← C[A[j]] + 1             ▷ count how many A[j]'s
  for i ← 1 to k
      do C[i] ← C[i] + C[i-1]              ▷ C[i] = # elem ≤ i
  for j ← n downto 1
      do B[C[A[j]]] ← A[j]                 ▷ Place correctly
         C[A[j]] ← C[A[j]] - 1               and decrease count
```

Running time $\Theta(n+k)$

Linear if $k = O(n)$

observation 1: Correct position of $A[j]$ = # elements $\leq A[j] = C[A[j]]$

observation 2: Last loop makes counting sort stable.

stable: elements with same value preserve their original order

Radix Sort(A, d) ▷ Assume d digits

for $i \leftarrow 1$ to d

do stable sort A on digit i

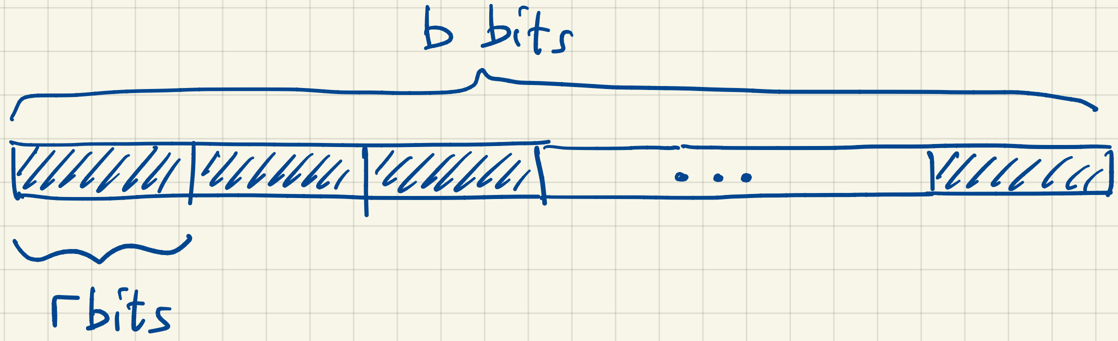
Example:

326	690	704	326
453	751	608	435
608	453	326	453
835	704	835	608
751	835	435	690
435	435	751	704
704	326	453	751
690	608	690	835

sorted

Running time : $\Theta(d(n+k))$

d passes each $\Theta(n+k)$ time.



\# passes: $\left\lceil \frac{b}{r} \right\rceil$

time per pass: $n + 2^r$

Total time: $\Theta\left(\frac{b}{r}(n+2^r)\right)$

How to choose r? let $r \approx \log_2 n \implies \frac{2bn}{\lg n}$

we get $\Theta\left(b \frac{n}{\lg n}\right)$

If our largest number is $O(n^d)$, where d is const.

we need $b = O(d \lg n)$ bits to represent the numbers

Running time $= O(dn)$ [linear].

Bucket Sort

- Elements in $A[1 \ldots n]$ are uniformly distributed in $[0,1)$
- Divide them into n buckets using $B[0 \ldots n-1]$
- map $A[i]$ to bucket $\lfloor n \cdot A[i] \rfloor$ (bucket $\equiv$ list)

  (this is some sort of Hashing, we look at Hashing in more detail later)

```
Bucket Sort (A, n)
    for i ← 1 to n
        do insert A[i] into list B[⌊n.A[i]⌋]
    for i ← 0 to n-1
        do sort list B[i] using insertion sort
    concatenate B[0], B[1], ..., B[n-1]
    return concatenated lists
```

if $A[i] \leq A[j]$

$nA[i] \leq nA[j]$

Example:

$n = 10$

0.01
0.2
0.3
0.35
0.4
0.5
0.56
0.7
0.9
0.99

$\xrightarrow{\times 10}$

0.01	0
	1
0.2	2
0.3	3
0.4	4
0.5 0.56	5
	6
0.7	7
	8
0.9 0.99	9

$$X_{ij} = \begin{cases} 1 & i^{th} \text{ element maps to bucket } j \\ 0 & \text{otherwise} \end{cases}$$

- $E[X_{ij}] = 1 \cdot \frac{1}{n} + 0\left(1-\frac{1}{n}\right) = \frac{1}{n}$

Also note:

- $E[X_{ij}^2] = 1^2 \cdot \frac{1}{n} + 0^2\left(1-\frac{1}{n}\right) = \frac{1}{n}$

- $E[X_{ij} X_{kj}] = E[X_{ij}] \cdot E[X_{kj}] = \frac{1}{n^2}$ [independence]

Let l_j = length of list j $\quad l_j = \sum_{i=1}^{n} X_{ij}$

Expected running time =

$$\Theta(n) + \Theta\left(E\left[\sum_{j=0}^{n-1} l_j^2\right]\right) = \Theta(n) + \Theta(n E[l_j^2])$$

$$E[\ell_j^2] = E[\overbrace{(X_{1j} + X_{2j} + \cdots + X_{nj})}^{\ell_j}\overbrace{(X_{1j} + X_{2j} + \cdots + X_{nj})}^{\ell_j}]$$

$$\underbrace{E[X_{1j}^2] + \cdots + E[X_{nj}^2]}_{n}$$

$$+\underbrace{E[X_{1j}X_{2j}] + \cdots + E[X_{nj}X_{n-1\,j}]}_{n(n-1)}$$

$$= n\frac{1}{n} + n(n-1)\frac{1}{n^2} = 1 + \frac{n-1}{n} < 2$$

Bucket Sort runs in $\Theta(n) + O(n) = \Theta(n)$