Problem: \( W = \text{Universe of keys} \) \( \text{SMALL!} \) so what?
so that
an array of length
\( k+1 \) of counts
is feasible.

\( A = (a_1, a_2, \ldots, a_N) \) sequence of keys (to sort) \( \text{BIG} \)

| 2\(^{(1)}\) | 5\(^{(1)}\) | 3\(^{(1)}\) | 0\(^{(1)}\) | 2\(^{(2)}\) | 3\(^{(2)}\) | 0\(^{(3)}\) | 3\(^{(3)}\) |

In most real world applications, is 3 an integer?

3\(^{(1)}\) is George 3\(^{(1)}\) is Andy 3\(^{(3)}\) is Tom

Other data is sometimes called a **handle**

It's easy \( O(n) \) time to compute

\[
C = \begin{bmatrix}
2 & 0 & 2 & 3 & 0 & 1 \\
0 & 1 & 2 & 3 & 4 & 5
\end{bmatrix}
\]

\( C[i] = \# \text{records in } A \text{ with key = } i \)

\[
\begin{array}{cccccc}
2\(^{(1)}\) & 5\(^{(1)}\) & 3\(^{(1)}\) & 0\(^{(1)}\) & 2\(^{(2)}\) & 3\(^{(2)}\) & 0\(^{(3)}\) & 3\(^{(3)}\) \\
0\(^{(1)}\) & 0\(^{(2)}\) & 2\(^{(3)}\) & 2\(^{(2)}\) & 3\(^{(3)}\) & 3\(^{(1)}\) & 3\(^{(3)}\) & 5\(^{(1)}\)
\end{array}
\]
Given a key like [426, 295], the computer can quickly address memory to access an array element whose index is the key.
Idea 1: Ask where should the 5 go?

Bad idea:

Input $C[i] = \#\text{ of hugs in } A$ that $= i$

Output $C[i] = \#\text{ of hugs in } A$ that are $\leq i$

Where should $3(3)$ be copied?
Radix Sort

329 657

457 436 355

First idea: organize by highest order digit

329 355 457 436 657 720 839

have to sort the sublists separately
Clever idea: First organize by lowest order digit!
Bucket sort

| .78  | 0   |
| .17  | 1   | .17 | .12 |
| .39  | 1   | .26 | .21 | .23 |
| .72  | 3   | .39 |
| .94  | 4   |
| .21  | 5   |
| .12  | 6   | .68 |
| .23  | 7   | .78 | .72 |
| .68  | 8   |
| .94  | 9   |
What data structure should you choose? And how will it be used so the bucket is eventually sorted?

Requirements:
(1) Insert an unpredictable number of elements one by one.
(2) End up with the elements sorted.

Option 1

Option 2
How much should be the replacement of each array be when it fills up?

\[+16 + 1 \text{ squared} \times 2\]  
we will analyze
\[ V \]

\[ V \]

\[ V + V \]

\[ 2V + 2V \]

\[ V + 2V + 4V + \ldots + 8V + \ldots + 2^k V \]

\[ 2^k V = N, \text{ so} \]

\[ \text{Total = } N \left( \frac{1}{2^k} + \frac{1}{2^{k-1}} + \ldots + \frac{1}{2} + 1 \right) < 2N \]
Quick Sort

Input Set N keys

Pick one randomly partition the rest

all < \[\cdot\] \[\cdot\] \[\cdot\]

all >

Recurse until done!

A little clever programming makes the computer do the partitioning in \( O(N \log N) \) really small and no extra space (except for controlling the recursions)

Running time?

\( O(N \log N) \) if you are lucky

\( \Theta(N^2) \) if NOT
The whole square above has \((N-1)^2\) dominoes. The sum equals approximately half of that number.
Intuitive" or "Heuristic" analysis

Very rough: When each split is about half and half, the same analysis as mergesort tells us the recursion depth: \( \log_2 N \)

\( O(N) \) work each level, total: \( CN \log_2 N \)

Less rough: If the split is 9:1 at worst,

\[
\text{Depth} = \# \times \left( \frac{9}{10} \right) \left( \frac{9}{10} \right) \left( \frac{9}{10} \right) \ldots (\frac{9}{10}). N \text{ to get } \leq 1
\]

\[
\left( \frac{9}{10} \right)^L N \approx 1 \quad N = (\frac{10}{9})^L
\]

\[
\log_{\frac{10}{9}} N = L = \frac{\log_2 B}{\log_2 A} = \frac{\log_2 N}{\log_2 (\frac{10}{9})}
\]

\[
\log_A B = L \iff A^L = B \quad \text{whats } L ?
\]

Street Clobber fight: A with a \( \log_2 \) \( A = 2 \log_2 A \)

\[
(2 (\log_2 A))^L = B 
\]

\[
2^{(\log_2 A) \cdot L} = B
\]

Now Clobber B: \( (\log_2 A) \cdot L = \log_2 B \)

\( = \text{Solve for } L \) 😊
\[ \log_{(\frac{10}{9})} N = L = \frac{\log_2 B}{\log_2 A} = \frac{\log_2 N}{\log_2 (\frac{10}{9})} \]

\[ \frac{10}{9} = \frac{9}{9} + \frac{1}{9} = 1 \frac{1}{9} = 1.\overline{11111} \]

\[ 0.15 \ldots \]

\[ 2^{0.15} \ldots \]

\[ 2^{0.15} = 1.000 \ldots \]

\[ C = \frac{1}{0.15} = 6.67 \]

\[ \text{a little more than } 0 \]

\[ 2 = \text{a little more than } 1. \]

\[ 2 = 1.\overline{11111} \ldots \]
Towards a precise analysis:

Mean: Imagine A is sorted (it really isn’t!)

**The first random choice**

- Gets compared to all these
- Gets compared to all these

**But**

- Are NEVER compared!

Beautiful fact from probability:

Expected value (Sum of Random Variables) = Sum of their expected values

EVEN if the many random variables are correlated!
Will $a_i$ and $a_j$ ever get compared to each other?

If and only if in that interval, either $a_i$ or $a_j$ was the first day made red.

\[
P(a_i \text{ or } a_j \text{ one carry}) = \frac{2}{(i-i+1)}
\]  

Expected # comparisons = \[ \sum_{i<j} \frac{2}{j-i+1} = O(N \log N) \]

all pairs $a_i, a_j, a_k, \dots$ etc
Big Digit Fact:

\[
\text{Number } \times \text{a million} \quad \frac{\text{To write } X}{\text{in base } \text{Base}} \quad \sqrt[\text{Base}]{1,000,000} \quad \# \text{digits} \approx \log_{\text{Base}} X
\]

\[
\sqrt{7} \approx 6
\]

\[
X \cdot \text{million} \quad 10 \times 10 \times 10 = 1000 \quad 6 \text{ total mult.}
\]

\[
\text{So } \log_{10} (1 \text{ million}) = 6
\]

Cryptography
- Rivest
- Shamir
- Adleman
- RSA

Key \((M, E)\)
- 100-200 Decim Digits

\[
M = F_1, F_2, F_1 + F_2, \text{ odd } \approx 50-100 \text{ Decim Digit Primes}
\]
\[
\begin{align*}
\text{\(f_N\rightarrow 1234\)} \\
\times \text{\(4321\)} \\
\underline{1234} \\
\underline{2468} \\
\underline{3702} \\
\underline{4938} \\
\text{\(576\)} \\
\text{\(714\)} \\
\text{\(2944\)} \\
\Theta(N) \quad 2944 \times 10^4 \quad \text{or something?}
\end{align*}
\]

\[1234 = (12) \times 10^2 + 34\]
\[4321 = (43) \times 10^2 + 21\]
\[12 - 43 \cdot 10^4 + \left[\left(12 \times 21\right) + (43 \times 34)\right] \cdot 10 + 34 \times 21\]
\[\left[(12 + 34) \times (43 + 21) - 12 \times 34 - 43 \times 21\right]\]
\[ \text{Missing derivative} \]

\[ 1 \times 3 \times 4 = (12) \times 10^2 + 34 \]

\[ 4 \times 3 \times 2 \times 1 = (43) \times 10^2 + 21 \]

\[ 12^{-4 \times 3} \times 10^4 + \begin{bmatrix} (12 \times 21) + (43 \times 34) \end{bmatrix} \times 10 + (34 \times 21) \]

\[ \left[ (12 + 34) \times (21 + 43) - 12 \times 34 - 43 \times 21 \right] \]
$$\Pi(n) = 3T(n/2) + CN$$

$$\frac{n}{2} \quad \frac{n}{4} \quad \frac{n}{4} \quad \frac{n}{4}$$

$$\frac{n}{4} \quad \frac{n}{4} \quad \frac{n}{4} \quad \frac{n}{4}$$

$$\frac{n}{4} \quad \frac{n}{4} \quad \frac{n}{4} \quad \frac{n}{4}$$

$$\frac{n}{2} \quad \frac{n}{2} \quad \frac{n}{2} \quad \frac{n}{2}$$

$$3CN$$

$$CN\left(1 + \frac{3}{2} + \left(\frac{3}{2}\right)^2 + \ldots + \left(\frac{3}{2}\right)^{\log_2 n}\right)$$

$$CN\left(\frac{3}{2}\right)^{\log_2 n} \left(\frac{2}{3}\right)^{\log_2 n-1} + \ldots + \frac{2}{3} + 1$$

$$\leq \frac{1}{1 - \frac{2}{3}} = 3$$

$$3 = 2^\log_2 3 \quad CN \cdot 2^\log_2 3 \cdot N = CN^{\frac{3}{2}}$$
\[ \log_2 3 = ? \]

By calculation, this is \( \approx 1.6 \) (Between 1 and 2)

\[ \log_2 1 = 0 \quad \log_2 2 = 1 \]

- \[ 100 \quad 128 = 2^7 \quad 64 = 2^6 \quad 32 = 2^5 \quad 256 = 2^8 \]
- \[ 50 \]
- \[ 25 \]
- \[ \text{Log}_2 1007 = 7 \]
- \[ \begin{array}{c}
    7 \\
    4 \\
    2 \\
    1 \\
    \end{array} \]
Extendable array data structure

Problem:
Input: N elements GIVEN sequentially, BUT N is NOT KNOWN ahead of time.
Output: N and the N elements stored in the sequence they were inputted.

Conventional wisdom (?)

Array?
Linked List?
Java Vector? Java ArrayList?
Linked list solution:

1. Allocate a new node and set its value to new value.
2. Copy data into new node.
3. Update the last node's next pointer to the new node.
4. Update the last node.
At the end of the day, when all N data items have been stored, HOW MUCH TIME HAD BEEN CONSUMED FOR ALL THE WORK??
Prepare to do this on a QUIZ on Tuesday:
Do the math that shows the TOTAL TIME used for all this work, in terms of N, is O(N). In other words, all of the times on the right added up to a number that is < some constant * C * N.

Towards the end of class, we read from the Java class library documentation comparing ArrayList and Vector. Vector is synchronized, but grows the size by adding a constant amount. ArrayList is not synchronized, but: Does not specify any size-growing algorithm but it DOES give a small O(n) amortized running time guarantee for adding n elements. It says "The constant factor is low compared to that for the LinkedList implementation."
Problem:
Input: Array A of numbers, length N.
Output: Indexes and value (lowi, hii, val) so val is the MAXIMUM SUM OF some ADJACENT ELEMENTS, and those adjacent elements are in A[lowi...hii].

3 Solutions:
Bad:
Ugly:
Good:
\[ T(N) = 2T(N/2) + CN \]

\[ U(N) = 2U(N/2) + C \]

\[ C + 2^1(C + 2^1(C + 2^1(\ldots))) \]

\[ = C + 2C + 2^2C + \ldots + 2^{(\log_2 N)}C \]

\[ = CN(1/2^{\text{don't care}} + \ldots + 1/2 + 1) \]