Sorting

Dirty tricks to sort faster than *$O(n \log n)$*

Mustafa Hajij
MSDS program
University of San Francisco

UNIVERSITY OF SAN FRANCISCO

Sorting

- We can sort any kind of element for which we have a similarity or distance measure between any two elements (subject to triangle inequality property*)
- Traditional sorting algorithms: bubble sort, merge sort, quicksort
- eonhole sort, bucket sort can often sort in O(n)
- What's the fastest we could ever sort *n* numbers?
  - It depends on whether we're stuck using comparisons only

*https://en.wikipedia.org/wiki/Triangle_inequality

Bubble sort

- $O(n^2)$
- *Stable*: order of equal elements doesn't change
- **Idea**: look for out-of-order elements and then keep swapping until nothing changes

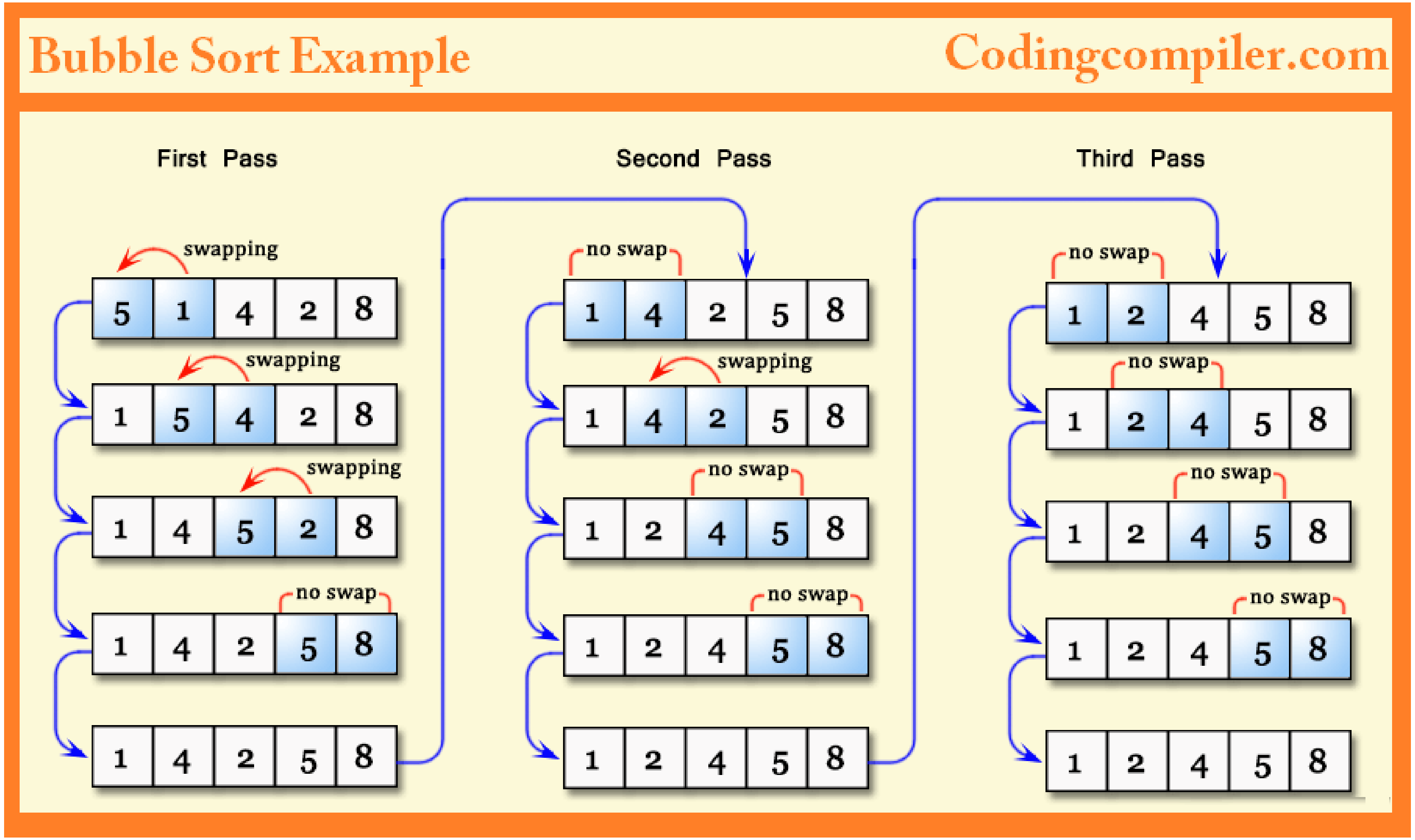
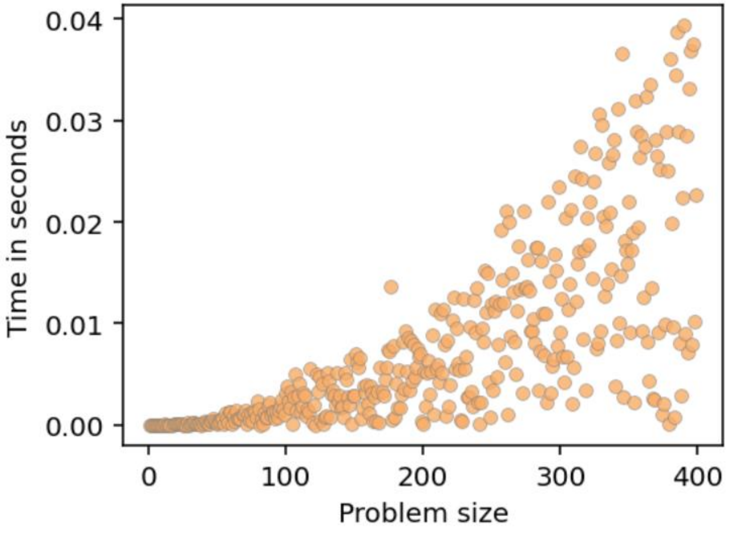


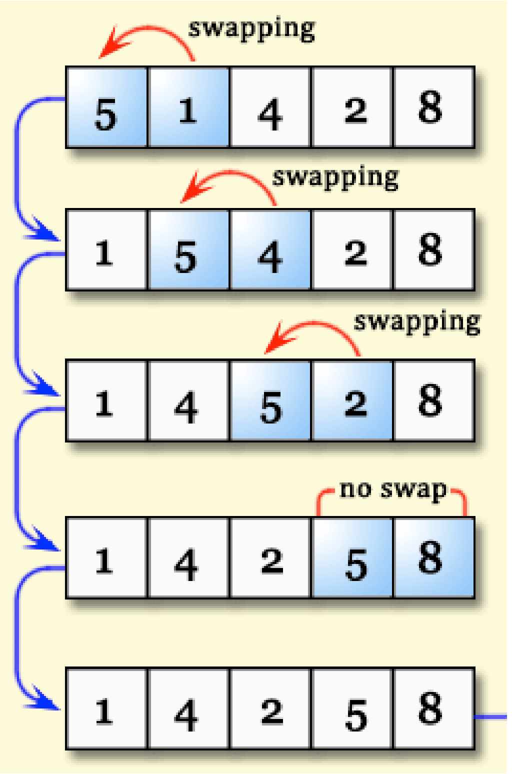
Image credit https://codingcompiler.com/bubble-sort-program-in-c-using-function/

Bubble sort in Python

```
changed=True
second_to_last_idx = len(A)-2
while changed:
  changed=False
  for i in range(second_to_last_idx+1):
    if A[i] > A[i+1]:
      A[i], A[i+1] = A[i+1], A[i]
      changed=True
```

Why is this $O(n^2)$?
(hint: What is worst case order in array?)





Sorting notebook https://github.com/USFCA-MSDS/msds689/blob/master/notes/sorting.ipynb

Merge sort (review)

- Faster than bubblesort: *O(n log n)*
- Simpler too, if you are comfortable with recursion
- It's stable
- Not in-place, uses lots of extra storage (sort halves)
- **Idea**: split currently active region in half, sorting both the left and right subregions, then merge two sorted subregions
- Eventually, the regions are so small we can sort in constant time; i.e., sorting 2 nums is easy
- Merging two sorted lists can be done in linear time

Quicksort, another divide and conquer sort

- $O(n^2)$ worst-case behavior but O(*n log n*) typical behavior
- **Idea**: pick pivot, partition so elements left of pivot are less than pivot and elements right are greater (not sorting here); recursively partition the left and right until small enough to sort trivially
- Picks a pivot element, rather than just split in half like mergesort
- Faster than bubble because it moves elements more than just one spot in the array
- Quicksort is / can be in-place whereas merge sort makes lots of temporary arrays, which can get expensive
- Quicksort is mostly faster than merge sort due to the constant in front of the complexity (memory allocation, hardware efficiencies, …)

Sorting notebook https://github.com/USFCA-MSDS/msds689/blob/master/notes/sorting.ipynb

In-place quicksort in notebook https://github.com/parrt/msds689/blob/master/notes/sorting.ipynb

Quicksort algorithm

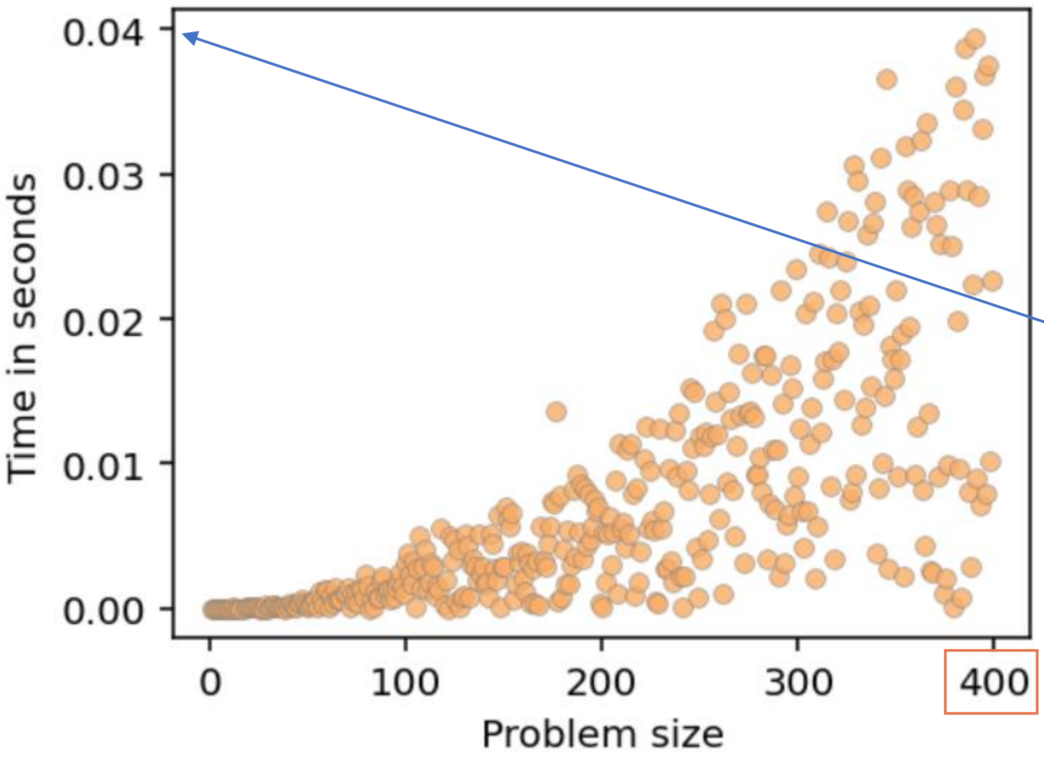
```python
def qsort(A, lo=0, hi=len(A)-1):
    if lo >= hi:
       return
    pivot_idx = partition(A,lo,hi)
    qsort(A, lo, pivot_idx-1)
    qsort(A, pivot_idx+1, hi)
```

```python
# many ways to do this; here’s a slow O(n) one
# breaks idea of in-place for qsort
def partition(A,lo,hi):
  pivot = A[hi]  # pick last element as pivot
  left = [a for a in A if a<pivot]
  right = [a for a in A if a>pivot]
  A[lo:hi+1] = left+[pivot]+right # copy back
  return len(left) # return index of pivot
```

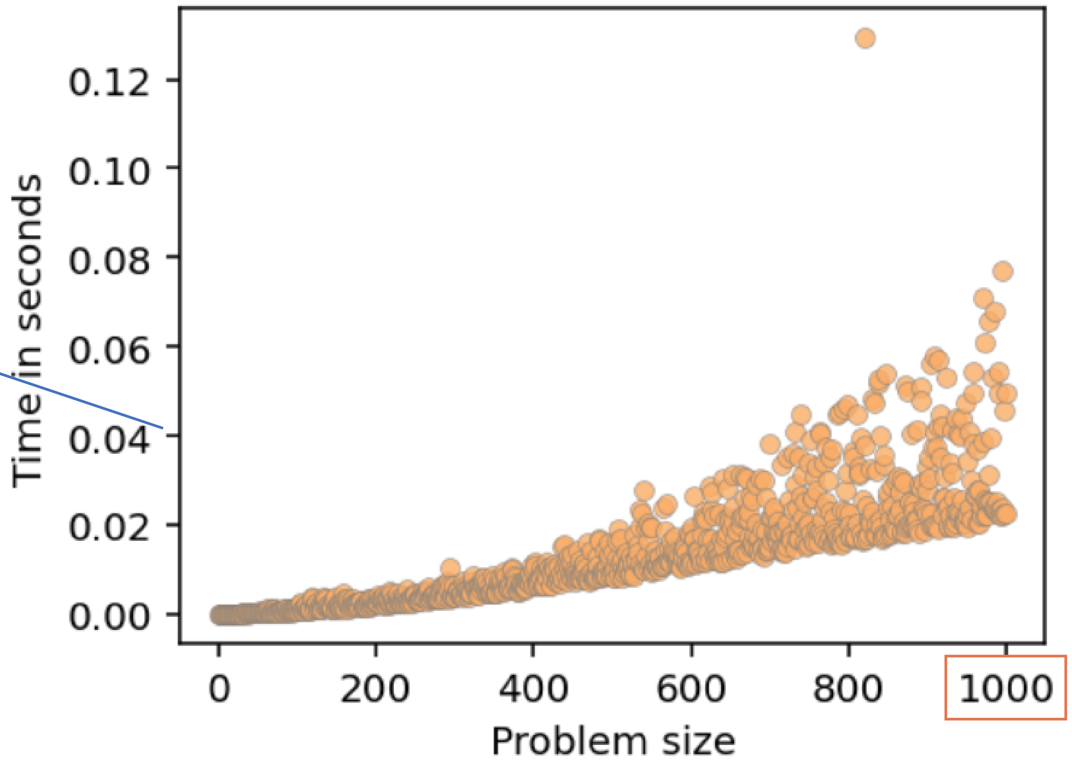
Video on partitioning: https://www.youtube.com/watch?v=MZaf_9IZCrc

Compare bubble, quicksort

Bubble sort



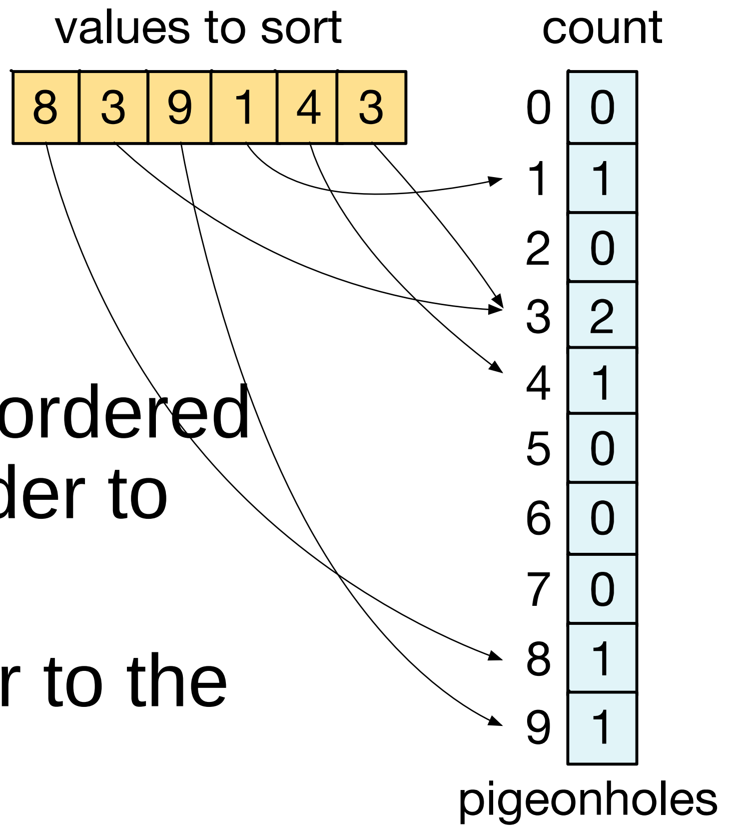
Quicksort



UNIVERSITY OF SAN FRANCISCO

So much for traditional sorts

- Theory says we can't beat O(n log n)…
- …for generic elements and doing comparisons
- But, what if we know the elements are ints or strings or floats?
- What if we know something about the values?
- E.g., what if we know the elements are ints in range 0..99?
- How can we sort those numbers in less than O(n log n)?



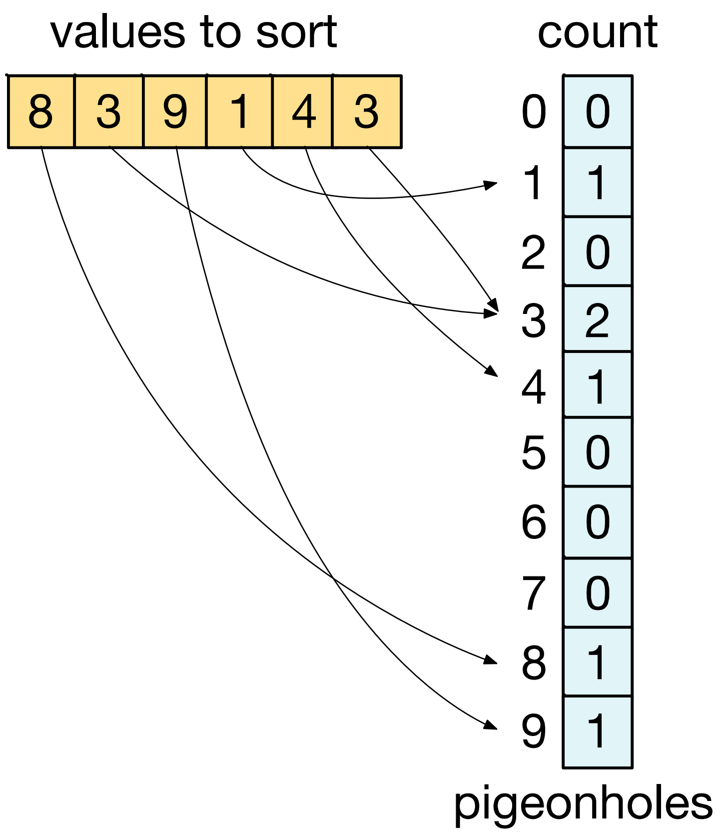
Pigeonhole sort

- **Idea**: Map each key to unique pigeonhole in an ordered range of holes; then just walk pigeonholes in order to get sorted elements
- Works best when the range of keys, *m*, is similar to the number of elements, n; why is that?
- T(n,m) = n + m
- This should smack of perfect hashing to you!

https://en.wikipedia.org/wiki/Pigeonhole_sort

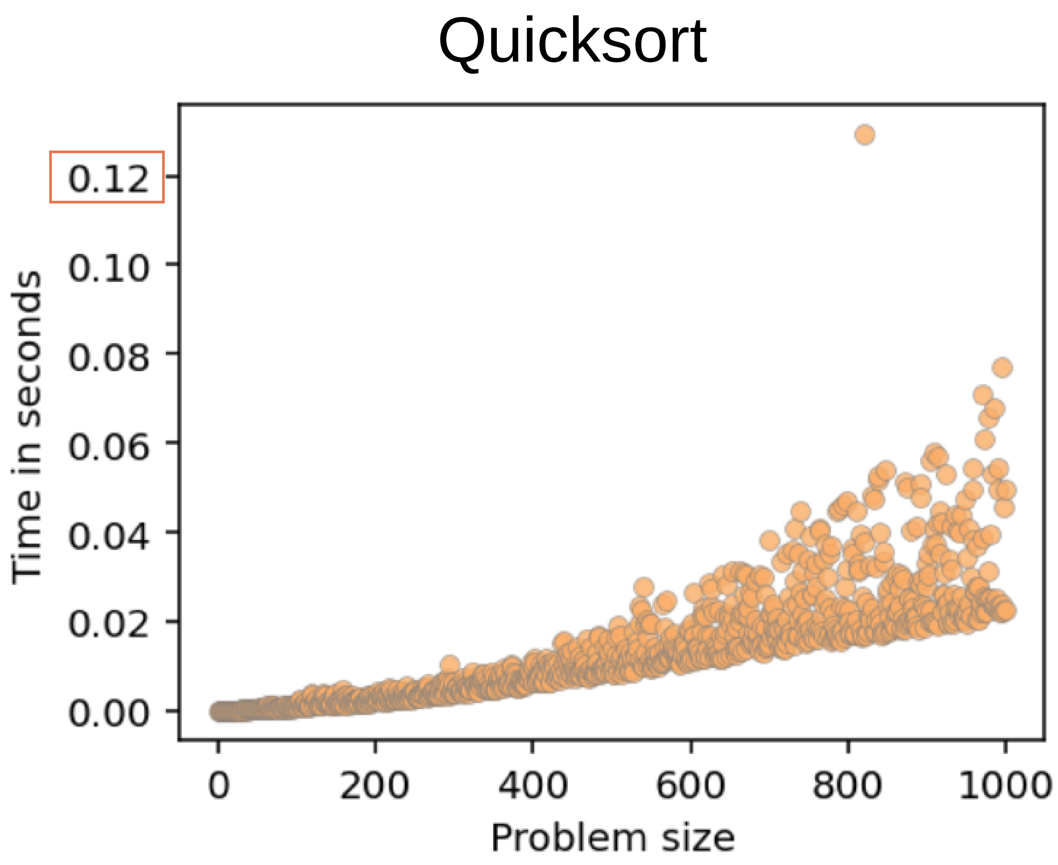
Pigeonhole sort algorithm

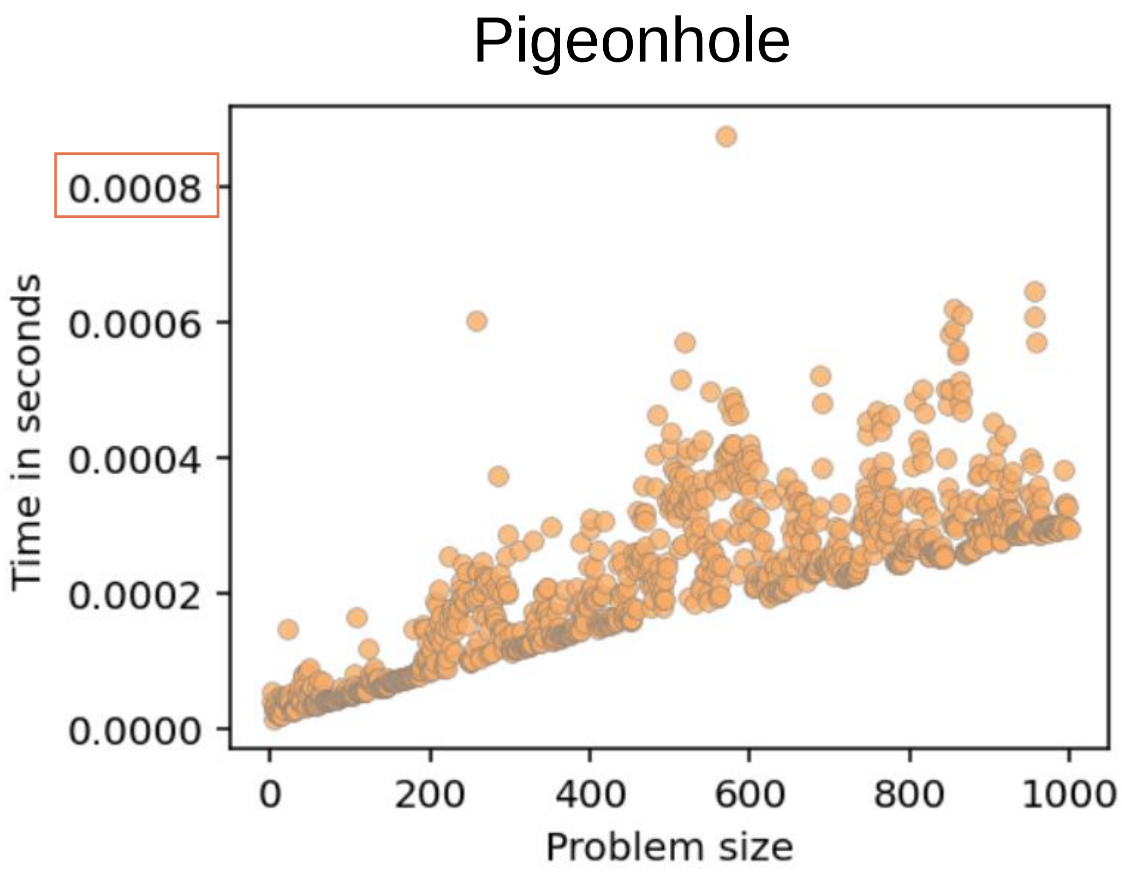
```
# fill holes
size = max(A) + 1
holes = [0] * size
for a in A:
    holes[a] += 1

# pull out in order
A_ = []
for i in range(0,size):
   A_.extend([i] * holes[i])
```



See sorting notebook

Compare quicksort, pigeonhole





Issue with pigeonhole sort

- Super fast and simple but…
- What do we do when *m* >> *n*? E.g., sort 2 numbers, 5 and 5 million. Takes T(n,m) = n + m = 5 + 5,000,000
- How can we handle this case & generalize to work for floats too?
- Hint: compress *m* to some fixed number of buckets instead of range of numbers
- Now we have hash table but with special hash function

Summary

- If asked, sorting is O(n log n) (via comparisons)
- Divide and conquer, merge and quicksort, are primary algorithms
  - Mergesort merges two sorted halves recursively; takes extra memory
  - Quicksort partitions instead of sorting halves; works in-place (usually better)
- But, we can do better with pigeonhole sort, mapping each element to unique bucket based on the key; O(n)