



Chapter 3, Case Studies:

- ▶ Linear-time sorting algorithms (**Monday and Wednesday (and Thursday)**)
- ▶ Disjoint sets and array forests (Friday)
- ▶ Priority queues (next week monday)
- ▶ N -sets and bit vectors (next week Wednesday)

Today (and Wednesday):

- ▶ Recent quiz problem
- ▶ Intro to “case studies”
- ▶ Limitations of comparison-based sorting
- ▶ Counting sort
- ▶ Radix sort
- ▶ (Lab: Bucket sort)

ArrayList

LinkedList

get()

set()

add()

ADTs

List
Stack
Queue
Map
Set
Bag

Data structures

Array
Linked list
Multi-dimensional array
Ring buffer



Can't you tell a good tree from a poor tree?

Good sorts

Merge

Quick (expected case)

Shell (unassigned project)

Heap (Section 3.3)

Bad sorts

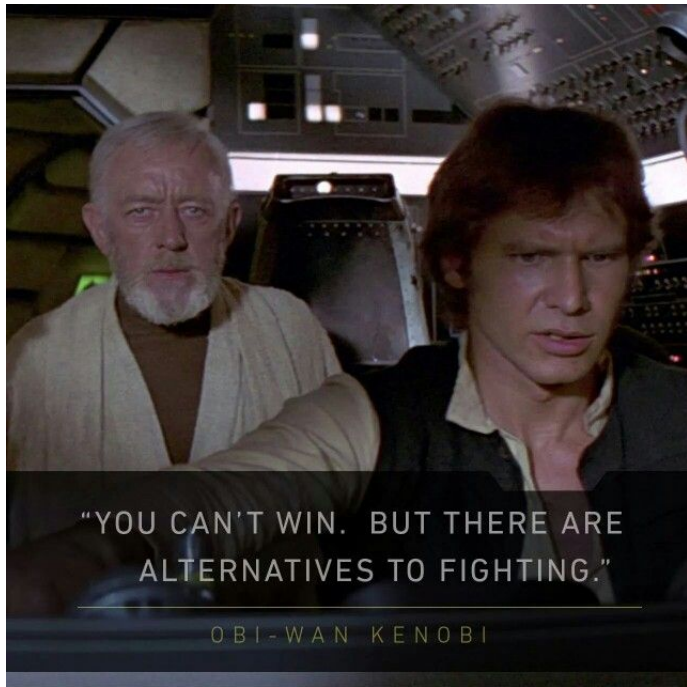
Selection

Insertion

Bubble



I have just been thinking, and I have come to a very important decision. These are the wrong sort of bees.



You can't
comparison-sort
in linear time.
But there are
alternatives to
comparisons.

Meme from <https://www.pinterest.com/pin/561542647262613858/>

1 0 1 1 4 0 2 1 3 0 1 1 3 2 2 1 2 1 4 0 4 2 3 1 1 2 1 1 2 1 3 2 4 0 4

0. Alice 0
1. Bob 2
2. Carol 4
3. Dave 4
4. Eve 2
5. Fred 0
6. Georgia 0
7. Henry 1
8. Ida 4
9. Jack 2
10. Karen 4
11. Larry 0
12. Moira 2
13. Nate 3
14. Olivia 1
15. Pete 1
16. Queenie 1
17. Ralph 4
18. Sara 2
19. Trent 4
20. Ursulla 2
21. Vick 3
22. Wendy 1
23. Xavier 2
24. Yvette 0
25. Zeke 3

Coming up:

Do **Implementing ADTs Project** (*due this past Monday, Jan 29*)

Do **Linear Sorting Project** (*due this coming Monday, Feb 5*)

Due **Wed, Jan 31**: (*class time*)

Read Section 3.1

Do Exercises 2.(22–24)

Due **Thurs, Feb 1**: (*end of day*)

Take sorting quiz

Due **Fri, Feb 2**: (*end of day*)

Read Section 3.2

Do Exercises 2.(12 & 16) and 3.8.

Take disjoint sets quiz

```
static Node arrayToList1(int[] array) {
    Node toReturn = new Node(array[0], null);
    for (int i = 1; i < array.length; i++) {
        Node current = toReturn;
        while (current.next() != null)
            current = current.next();
        current.setNext(new Node(array[i], null));
    }
    return toReturn;
}
```

```
Node arrayToList2(int[] array) {
    Node toReturn = null;
    for (int i = array.length - 1; i >= 0; i--)
        toReturn = new Node(array[i], toReturn);
    return toReturn;
}
```

```
static int[] listToArray(Node head) {
    int size = 0;
    for (Node current = head; current != null; current = current.next())
        size++;
    int[] toReturn = new int[size];
    int i = 0;
    for (Node current = head; current != null; current = current.next())
        toReturn[i++] = current.datum();
    return toReturn;
}
```

Invariant (Loop of radix_sort)

- (a) i is the number of iterations completed.
- (b) $r_pow = r^i$.
- (c) $\forall k \in [0, n - 1), \text{sequence}[k] \bmod r^i \leq \text{sequence}[k + 1] \bmod r^i$

0110	1011	1100	0111	0001	1110	1001	1101
------	------	------	------	------	------	------	------

0110	1011	1100	0111	0001	1110	1001	1101
------	------	------	------	------	------	------	------

0110	1100	1110	1011	0111	0001	1001	1101
------	------	------	------	------	------	------	------

0110	1011	1100	0111	0001	1110	1001	1101
------	------	------	------	------	------	------	------

0110	1100	1110	1011	0111	0001	1001	1101
------	------	------	------	------	------	------	------

1100	0001	1001	1101	0110	1110	1011	0111
------	------	------	------	------	------	------	------

0110	1011	1100	0111	0001	1110	1001	1101
------	------	------	------	------	------	------	------

0110	1100	1110	1011	0111	0001	1001	1101
------	------	------	------	------	------	------	------

1100	0001	1001	1101	0110	1110	1011	0111
------	------	------	------	------	------	------	------

0001	1001	1011	1100	1101	0110	1110	0111
------	------	------	------	------	------	------	------

0110	1011	1100	0111	0001	1110	1001	1101
------	------	------	------	------	------	------	------

0110	1100	1110	1011	0111	0001	1001	1101
------	------	------	------	------	------	------	------

1100	0001	1001	1101	0110	1110	1011	0111
------	------	------	------	------	------	------	------

0001	1001	1011	1100	1101	0110	1110	0111
------	------	------	------	------	------	------	------

0001	0110	0111	1001	1011	1100	1101	1110
------	------	------	------	------	------	------	------