Insertion Sort Analysis

6 | 5 | 4 | 3 2 1

$$\sum_{1}^{N} x = \frac{N(N+1)}{2}$$

n $(n-1)$ times

$O(n)$

```
for k from 0 to len(a)-2:
    assert(the first k positions are already sorted)

    # Pick up item k+1 (call it a[j]) and let it sink
    j = k+1
    while j > 0 and a[j-1] > a[j]:
        swap(a[j-1], a[j])
        j = j-1
```

$O(n)$

k	j
0	1
1	2
2	3
3	4
$\vdots$	$\vdots$
$n-2$	$n-1$

$$\sum \text{loops} = \sum_{j=1}^{n-1} j$$

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

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

$$\in O\left(\frac{n^2}{2}\right)$$

Is that Optimal?

Given n digits $\Rightarrow n!$

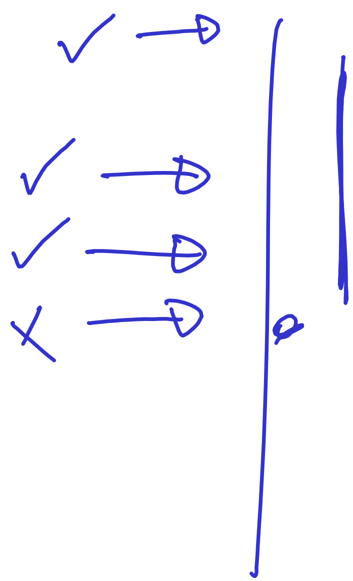
123
321
⋮

M items $\Rightarrow \log_2 M$ comparisons

$$\log_2 n! = \lg n!$$

$$\lg n! = \lg 1 + \lg 2 + \lg 3 + \dots \lg n$$
$$\leq \lg n + \lg n + \lg n + \dots \lg n$$
$$= n \lg n$$

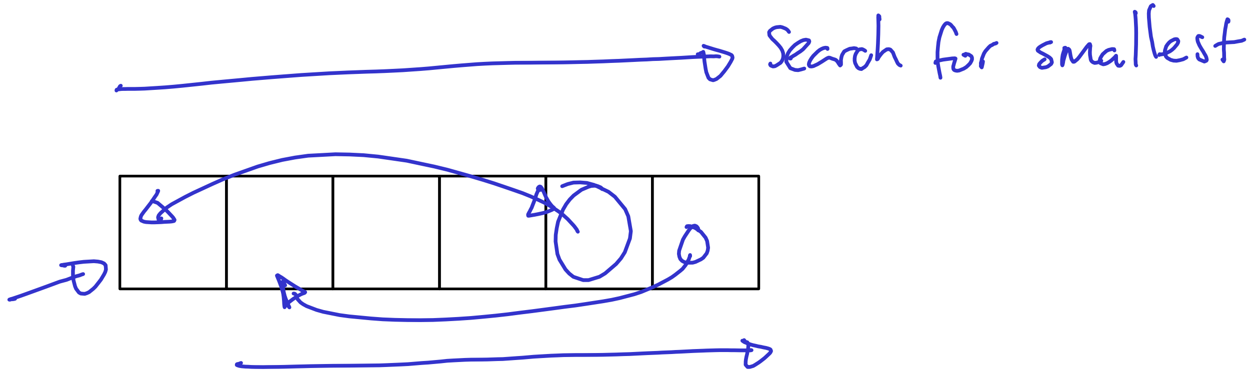
$$\boxed{O(n \lg n)}$$



$O(n^2)$ Sorting Algorithms

- There are lots of these around. They tend to be easy-ish to think up. Be careful not to reinvent the wheel!
- We're going to look at a few more so that you have been exposed to the common ones and you also get more practice analysing complexity.
- You will find a lot of CS types scoff at these algorithms as being a waste of time. But they have definite advantages:
  - They are usually concise and understandable (means fewer implementation bugs)
  - For small n, they might be the quickest method!

Selection Sort: Idea



Selection Sort: Example

3	6	4	5	1	2
1	6	4	5	3	2
1	2	4	5	3	6
1	2	3	5	4	6
1	2	3	4	5	6
1	2	3	4	5	6

k

Selection Sort

n times

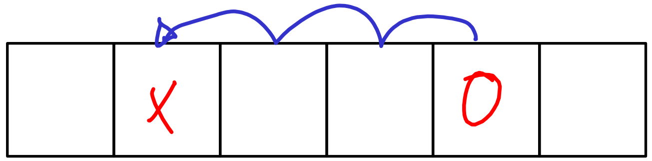
```
for k from 0 to len(a)-1:
    assert(the positions before a[k] are already sorted)

    # Find the smallest element in a[k..end] and swap it into a[k]
    iMin = k
    for j from iMin+1 to len(a)-1:
        if a[j] < a[iMin]:
            iMin = j
    swap(a[k], a[iMin])
```

$n-k$

k	loops
0	n
1	$n-1$
2	$n-2$
3	⋮
⋮	⋮
$n-1$	1

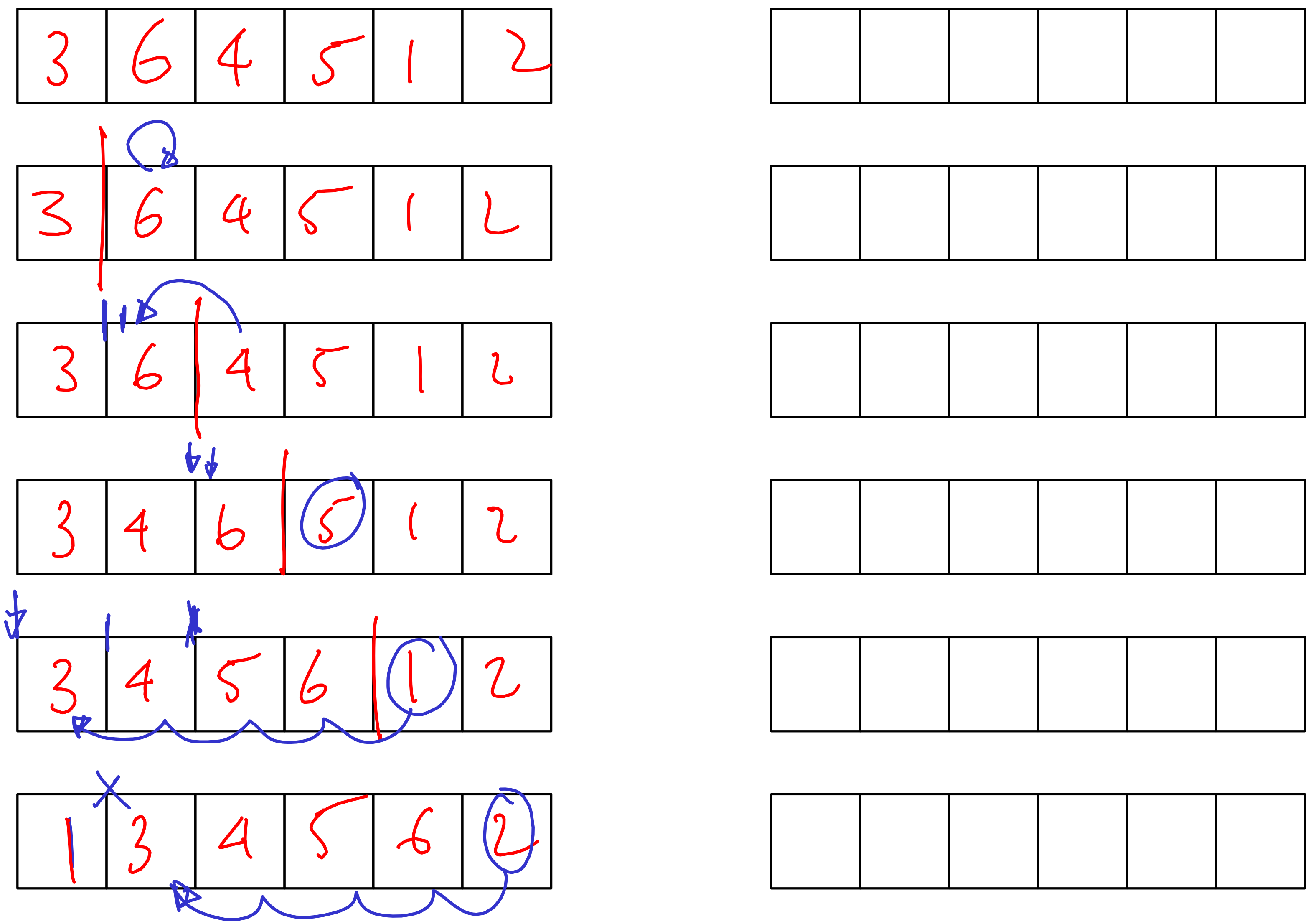
$$\sum_{j=1}^{n} j = \frac{n(n+1)}{2} \in \underline{\underline{O(n^2)}}$$

Binary Insertion Sort: Idea



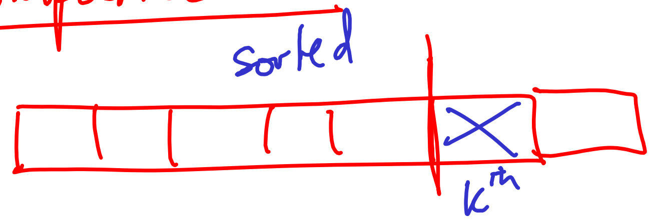
1. Binary chop to find where it will end up
2. Exchanges to get it there

Binary Insertion Sort: Example



Binary Insertion Sort: Analysis

Comparisons



k^{th} element could end up in any of k positions.

$\lg k$ comparisons

$$\sum_{k=1}^{n} \lg k = \lg 1 + \lg 2 + \dots \lg n$$

$$\leq n \lg n$$

$$\underline{\underline{O(n \lg n)}}$$

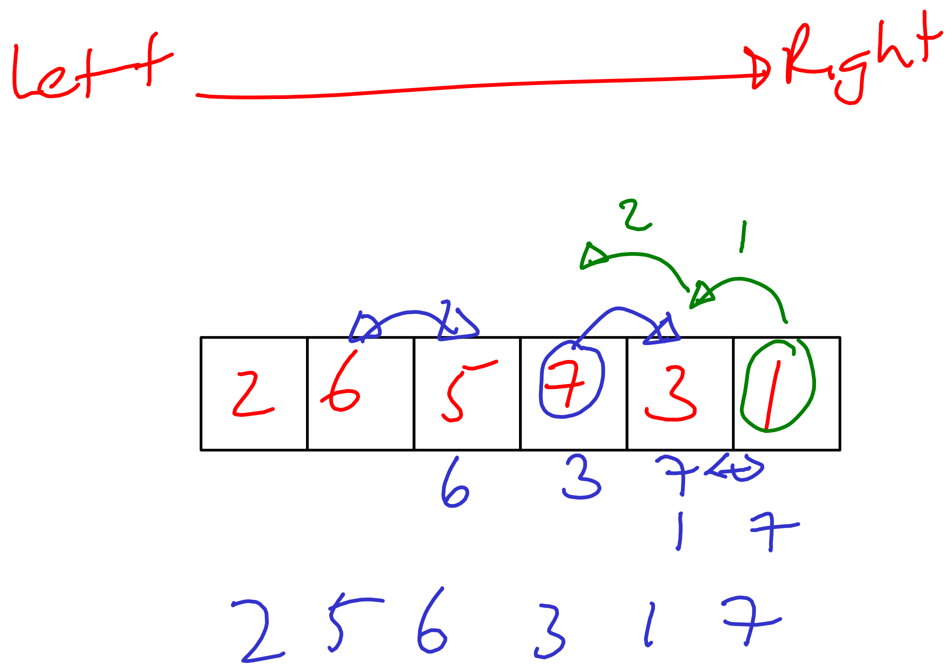
Swaps

$$\sum_{1}^{n} k = n(n+1)$$

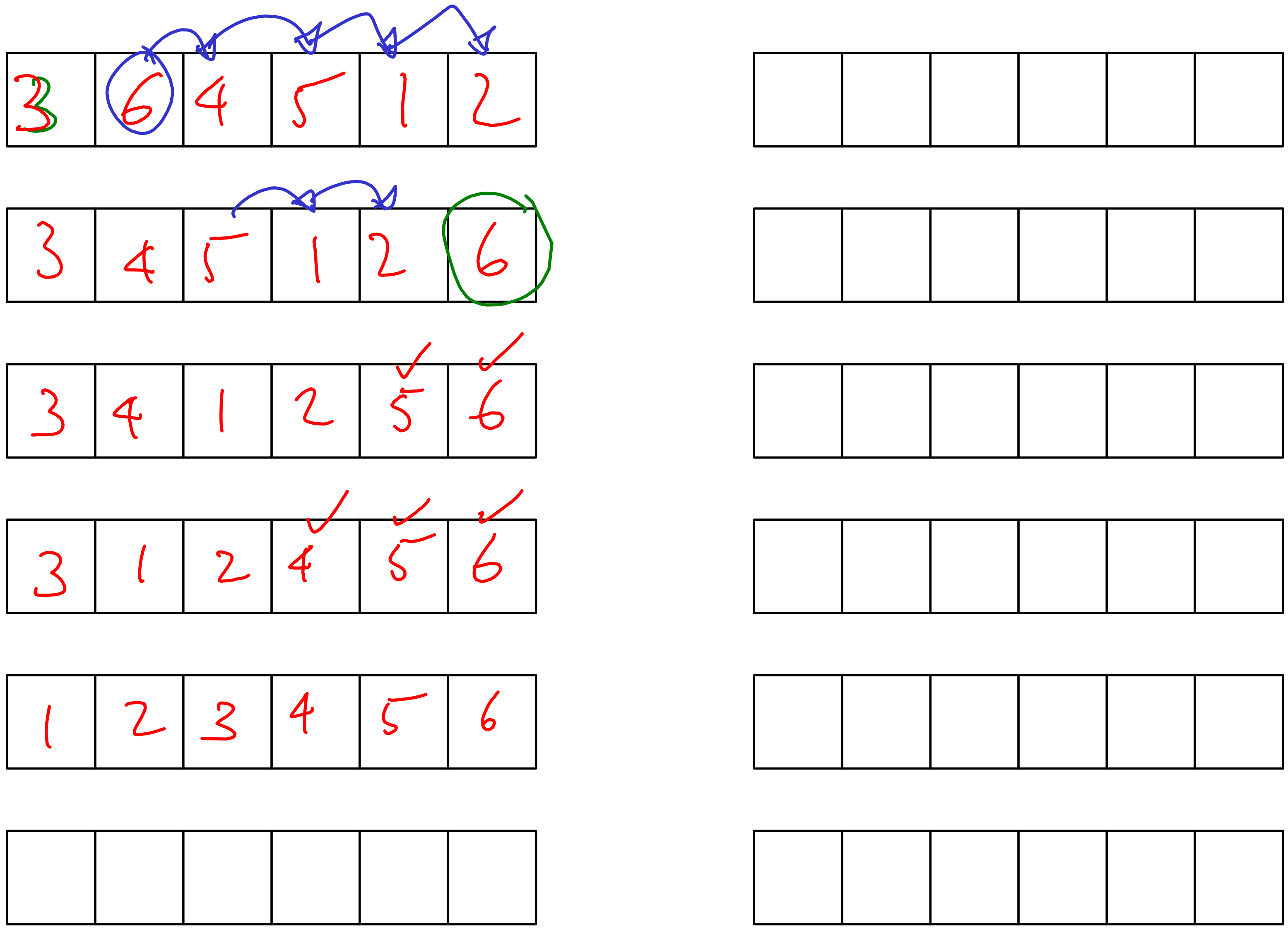
$$\in \underline{\underline{O(n^2)}}$$

Binary Insertion Sort: Analysis

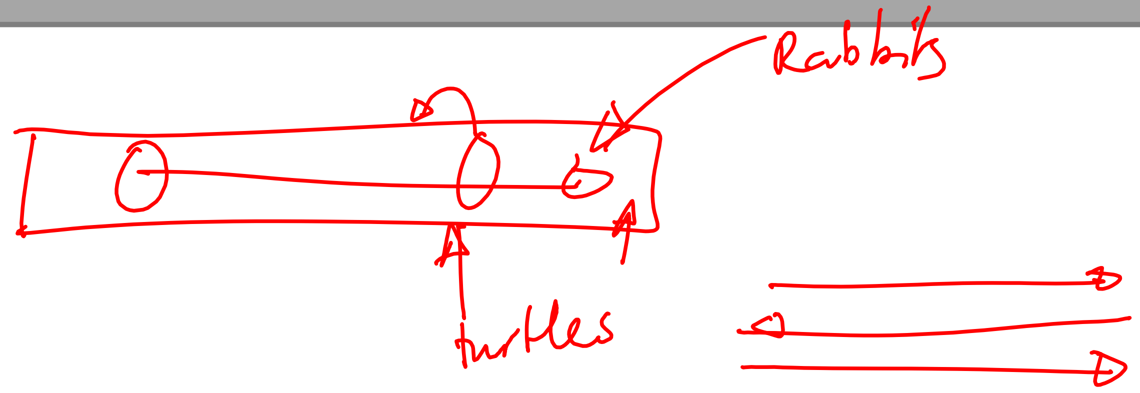
Bubble Sort: Idea



Bubble Sort: Example



Bubble Sort




```
repeat
    # Go through all the elements once, swapping any that are out of order
    didSomeSwapsInThisPass = false
    for k from 0 to len(a)-2:
        if a[k] > a[k+1]:
            swap(a[k], a[k+1])
            didSomeSwapsInThisPass = true
until didSomeSwapsInThisPass == false
```

$n-1$

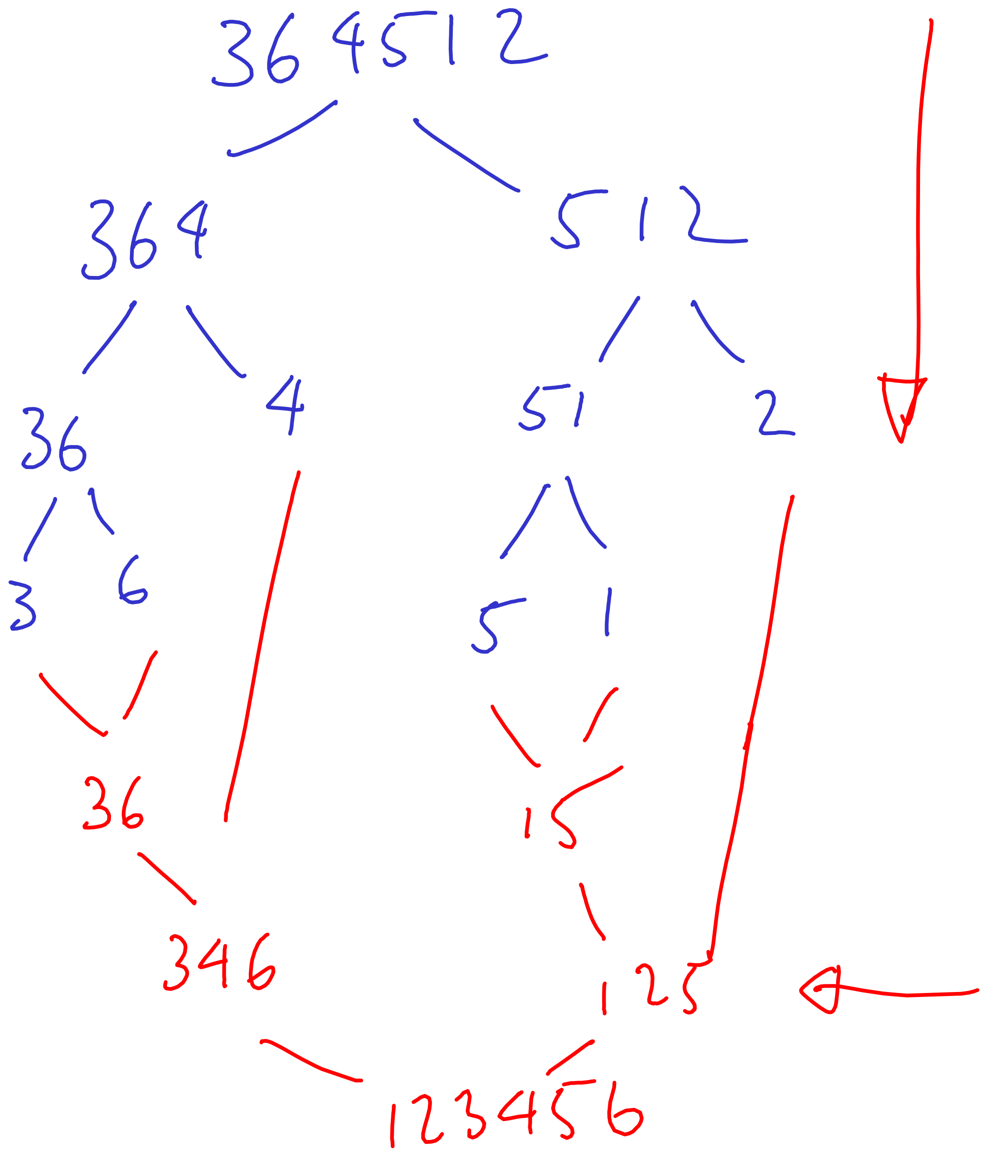
$(n-1)n = n^2 - n$

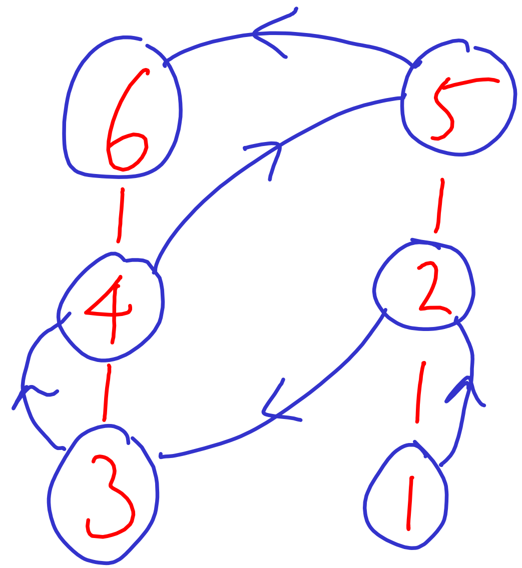
$\in O(n^2)$

Better Algorithms

- Can we actually hope to get close to our theoretical O(nlogn) limit?
- You know we can because you've met mergesort before.
  - In ML it was a lovely short algorithm that did a great job sorting lists
  - It has some drawbacks when we look at arrays but we'll get to that shortly...

Mergesort: Idea (recap)





Mergesort: Array Example

Main	Temp.
3 6 4 5 1 2	3 6
3 6 4 5 1 2	3 4 6
3 4 6 5 1 2	1 5
3 4 6 1 5 2	1 2 5
3 4 6 1 2 5	1 2 3 4 5 6
1 2 3 4 5 6	

$O(n)$

Mergesort: Analysis

$n = 2^m \longrightarrow m = \lg n$

$$f(n) = 2f\left(\frac{n}{2}\right) + kn$$

$$f(2^m) = 2f(2^{m-1}) + k2^m$$

$$= 2\left[f(2^{m-2}) + k2^{m-1}\right] + k2^m$$

$$= 2^a f(2^{m-a}) + ak2^m$$

$$f(n) = nf(1) + mk2^m$$

$$= nf(1) + n\lg n$$

$$\underline{\underline{O(n \lg n)}}$$

$\boxed{O(n) \text{ space}}$