Quicksort: Idea

- Recursive like mergesort, except it doesn't just slice the array into two
- The basic idea is to pick a pivot element
  - Any element will do, although we might get better results if we choose more carefully

```
4 6 1 3 5 9
```

- We then partition the array into those bigger than the pivot and those smaller than the pivot

```
1 4 3 5 6 9
```

Start

End of iteration
Quicksort: Idea

- Now we recursively apply quicksort to the two partitions
  \[
  \begin{array}{ccccccc}
  1 & 4 & 3 & 5 & 6 & 9 \\
  \end{array}
  \]

- How do we partition?
  1. Have two pointers, \( L \) and \( R \) that we initialise to either end of the array, excluding the pivot for now

  \[
  \begin{array}{ccccccc}
  4 & 6 & 1 & 3 & 9 & 5 \\
  \end{array}
  \]

  2. Increment \( L \) until \( a[L] \) is bigger than the pivot OR \( L==R \)

  \[
  \begin{array}{ccccccc}
  4 & 6 & 1 & 3 & 9 & 5 \\
  \end{array}
  \]

  bigger than 5
Quicksort: Idea

3. Decrement R until $a[R]$ is less than or equal to the pivot OR $R==L$

4. If ($L!=R$) then we can swap $a[L]$ and $a[R]$ in order to make $L$ and $R$ OK
4. If (L != R) then GOTO 2
   else GOTO 5

5. Now (L==R) and we swap the pivot with a[L]
Beware: Depending on where your pivot is chosen to be, you need to think carefully about what gets swapped where: it's very easy to end up being off by one. You will find lots of subtle variations on the algorithm implementations (all of which work). I recommend you try writing a quicksort that works on any pivot (the one here works just if you choose the last element as the pivot)
Quicksort: Example (Last element)
Quicksort: Example (sorted array)
Quicksort: Example (first element)
**Quicksort: Analysis (Gulp!)**

- **Best case**
  - Somehow we always choose the pivot that splits the sub-array being sorted into two even chunks
  - Then we have exactly what we had for mergesort
  - \( f(n) = 2f(n/2) + kn \)
  - That was \( O(n \log n) \) :-)

![Division Tree Diagram](image-url)
Worst case

- Somehow we always choose the biggest (or smallest) element as the pivot every time
- For a subarray of size $k$, we will always recurse on a single subarray of size $(k-1)$

$$f(n) = f(n-1) + kn$$

$$f(n) = f(n-2) + k(n-1) + kn$$

$$f(n) = f(n-m) + k(n-m+1) + \ldots + kn$$

$$f(0) + k + k^2 + \ldots + k^n = f(0) + \frac{k}{n} + \frac{k(k+1)}{n(n+1)}$$

$O(n^2)$
QuickSort: Analysis (Gulp!)

- At best we get $O(n \log n)$ and at worst we get $O(n^2)$ for performance
- $O(1)$ for space (it is all in-place)
- What about a more general cases??
Quicksort: Analysis (Gulp!)

- Recursion tree for constant split proportion (say 1:9)

\[
\begin{align*}
0(n) & \rightarrow \frac{n}{2} \rightarrow \frac{n}{4} \rightarrow \frac{n}{8} \\
& \rightarrow O(n)
\end{align*}
\]

\[
\begin{align*}
\log_{10} n & \rightarrow \frac{n}{10} \rightarrow \frac{9n}{100} \rightarrow \frac{9n}{100} \rightarrow \frac{81n}{100} \\
& \rightarrow O(n)
\end{align*}
\]

\[
\begin{align*}
O(n) \log_2 n & \rightarrow O(n) \log_{10} n + O(n) \log_{\frac{1}{9}} n \\
& = O(n) \log n + O(n) \log n \\
& = O(n \log n)
\end{align*}
\]
Quicksort: Analysis (Gulp!)

- Average case
- Consider choosing the pivot that is \( j \) places along in the sorted subarray
  - \( j=1 \) would be the same as always taking the first element
  - \( j=n \) would be the same as always taking the last element

\[
\begin{array}{c|c|c|c|c|c|c|c|c|c|c|c|c|c}
\hline
j & & & & & & & & & & & & \\
\hline
\end{array}
\]
Quicksort: Analysis (Gulp!)

- Then \( f(n) = f(n-j) + f(j-1) + kn \)

- Now let us imagine that we pick a pivot at random each time.
  - The value of \( j \) (where the pivot finally ends up) will change, and all values of \( j \) will be equally probable
  - So let's take an average

\[
f(n) = kn + \frac{\sum_{j=1}^{n} f(n-j) + f(j-1)}{n}
\]

Not the easiest thing to solve: see CLRS if you want full detail

Key point: \( O(n \log n) \)
Quicksort: Analysis (Gulp!)

- So why is quicksort the choice for sorting?
  - It has a poor worst case
  - But the best case is generally better than mergesort (smaller constants in $O(n \log n)$) and the average case is still $O(n \log n)$
  - And quicksort sorts in place $O(n \log n)$
  - Space $O(1)$

- But you should do what you can to avoid the worst case
  - E.g. randomise your input
  - E.g. randomise your pivot choice
Order Statistics

- Often we don't need a sorted array so much as a *partially sorted* array
  - E.g. value of the Xth element (think *median* calculation)
  - E.g. top 30 search results
  - We could sort the entire array and then read off what we want – this would be $O(n \log n)$
  - Seems like wasted effort...
The first partitioning leaves us with two subarrays, but we need only recurse on the one that contains the median.