BACK TO SORTING: QUICKSORT

a Divide & Conquer algorithm that runs "in-place"
BACK TO SORTING: QUICKSORT

a Divide & Conquer algorithm that runs “in-place”

- Divide: choose a pivot & place it s.t. everything before is smaller & everything after is not smaller

\[
\begin{array}{cccccccccccc}
\end{array}
\Rightarrow
\begin{array}{cccccccccccc}
< & X & X & > & X
\end{array}
\]
BACK TO SORTING: QUICKSORT

a Divide & Conquer algorithm that runs “in-place”

- Divide: choose a pivot & place it s.t. everything before is smaller & everything after is not smaller

- Conquer: Quicksort each side of pivot
BACK TO **SORTING**: **QUICKSORT**

a **Divide & Conquer** algorithm that runs "in-place"

- **Divide**: choose a **pivot** & place it s.t. everything before is smaller & everything after is not smaller

  \[ \begin{array}{c|c|c|c|c|c|c|c|c|c|c|c|c|c} X & ? & ? & ? & ? & ? & ? & ? \Rightarrow & < & X & X & > & x \end{array} \]

- **Conquer**: Quicksort each side of pivot

* Notice that after **Divide**, \( x \) is in its final (sorted) position.
  So there is nothing to do for [Combine] (merge)
BACK TO SORTING: QUICKSORT

A Divide & Conquer algorithm that runs "in-place"

- Divide: choose a pivot & place it s.t. everything before is smaller
  & everything after is not smaller

- Conquer: Quicksort each side of pivot

Notice that after Divide, X is in its final (sorted) position.
So there is nothing to do for [Combine]

\[ T(n) = T(j-1) + T(n-j) + f(n) \]

\[ f(n) = \begin{cases} \text{[Divide & Conquer]} & \text{for } n \geq 1 \\ \text{[Combine]} & \text{for } n < 1 \end{cases} \]

\[ T(0) = 0 \]
\[ T(1) = \Theta(1) \]
The heart of Quicksort is the **Divide** step.

- pivot: arbitrary → so just use first element
The heart of Quicksort is the **Divide** step

- **pivot**: arbitrary → so just use first element
- in-place indexing of pivot:

```
```

If all these are \( \geq x \), nothing to do. So advance until you find \( y < x \)
(or until the end)
The heart of Quicksort is the **Divide** step.

- **pivot**: arbitrary ⇒ so just use first element
- **in-place** indexing of pivot:

```
```

- If all these are \( \geq X \), nothing to do.
- So advance until you find \( Y < X \)
  - (or until the end)

```
```

- E.g. find \( Y \) immediately
The heart of Quicksort is the **Divide** step.

- **pivot**: arbitrary → so just use first element

- **in-place** indexing of pivot:


  if all these are \( \geq X \), nothing to do.

  So advance until you find \( Y < X \)

  (or until the end)

  e.g. find \( Y \) immediately


  or, you don’t find \( Y \) immediately

The heart of Quicksort is the **Divide** step.

- **pivot**: arbitrary → so just use first element
- **in-place** indexing of pivot:

  If all these are $\geq x$, nothing to do.
  So advance until you find $y < x$
  (or until the end)

  E.g. find $y$ immediately

  or, you don't find $y$ immediately

  Swap $y$ for the element to the right of $x$

  Advance in array
The heart of Quicksort is the **Divide** step.

- **pivot**: arbitrary → so just use first element
- in-place indexing of pivot:

  non-trivial case: you have at least one element < x

  non-empty → if all > x, done
  ? → if all > x, done

  if all > x, nothing to do.
  So advance until you find y < x
  (or until the end)
  e.g. find y immediately

  or, you don't find y immediately

  Scan from z: if z > x, advance
  else swap z ← a

  if it exists

  swap y for the element to the right of x

  advance in array
Assume $n$ distinct values for analysis. (not critical)

What input makes Quicksort work the most?
Assume n distinct values for analysis. (not critical)

What input makes Quicksort work the most?

⇒ already sorted input!
Assume \( n \) distinct values for analysis. (not critical)

What input makes Quicksort work the most?

\[ \text{already sorted input!} \quad \text{or} \quad \text{reverse sorted} \]

\[ \text{\textbf{\( T(n) = T(0) + T(n-1) + \Theta(n) = \Theta(n^2) \)}} \]

\( \text{\( n \)} \)
Assume \( n \) distinct values for analysis. (not critical)

What input makes Quicksort work the \underline{most}?

\( \rightarrow \) already sorted input! \( \rightarrow \) reverse sorted

\( \triangleleft \) every time, you get a maximally unbalanced recursion.

\( T(n) = T(0) + T(n-1) + \Theta(n) = \Theta(n^2) \)

What input makes Quicksort work the \underline{least}?
Assume $n$ distinct values for analysis. (not critical)

What input makes Quicksort work the most? 
\[ \downarrow \text{already sorted input! - or - reverse sorted} \]
\[ \downarrow \text{every time, you get a maximally unbalanced recursion.} \]
\[ T(n) = T(0) + T(n-1) + \Theta(n) = \Theta(n^2) \]

What input makes Quicksort work the least? 
\[ \downarrow \text{s.t. every time your pivot splits the groups evenly.} \]
\[ T(n) = 2T(\frac{n-1}{2}) + \Theta(n) = \Theta(n \log n) \]
Assume $n$ distinct values for analysis. (not critical)

What input makes Quicksort work the most?

\[ T(n) = T(0) + T(n-1) + O(n) = \Theta(n^2) \]  

What input makes Quicksort work the least?

\[ T(n) = 2T\left(\frac{n-1}{2}\right) + O(n) = \Theta(n \log n) \]

So, Quicksort is in-place, but can be slow. Why use it?

It's simple.
We said Quicksort is fast, i.e. $\Theta(n \log n)$, if pivot gives an even split. (ALWAYS)

What if we always split within $\frac{1}{10}$ to $\frac{9}{10}$?
We said Quicksort is fast, i.e. $\Theta(n \log n)$, if pivot gives an even split. (Always)

What if we always split within $\frac{1}{10}$ to $\frac{9}{10}$?

$$T(n) = T\left(\frac{n}{10}\right) + T\left(\frac{9n}{10}\right) + c \cdot n$$

$\begin{array}{c}
T\left(\frac{n}{10}\right) \\
T\left(\frac{9n}{10}\right)
\end{array}$
We said Quicksort is fast, i.e. $\Theta(n \log n)$, if pivot gives an even split. (ALWAYS)

What if we always split within $\frac{1}{10}$ to $\frac{9}{10}$?

$$T(n) = T\left(\frac{n}{10}\right) + T\left(\frac{9n}{10}\right) + cn$$
We said Quicksort is fast, i.e. $\Theta(n \log n)$, if pivot gives an even split. (ALWAYS)

What if we always split within $\frac{1}{10}$ to $\frac{9}{10}$?

$$T(n) = T\left(\frac{n}{10}\right) + T\left(\frac{9n}{10}\right) + c.n$$
We said Quicksort is fast, i.e. $\Theta(n \log n)$, if pivot gives an even split. (ALWAYS)

What if we always split within $\frac{1}{10}$ to $\frac{9}{10}$?

$$T(n) = T\left(\frac{n}{10}\right) + T\left(\frac{9n}{10}\right) + c.n$$
We said Quicksort is fast, i.e. $\Theta(n \log n)$, if pivot gives an even split. (ALWAYS)

What if we always split within $\frac{1}{10}$ to $\frac{9}{10}$?

$$T(n) = T\left(\frac{n}{10}\right) + T\left(\frac{9n}{10}\right) + c.n$$

$$T\left(\frac{n}{100}\right) \quad T\left(\frac{9n}{100}\right) \quad T\left(\frac{9n}{100}\right) \quad T\left(\frac{81cn}{100}\right)$$

$h_L \sim \log_{10} n \Rightarrow T(n) \gg cn \cdot \log_{10} n$

$h_R \sim \log_{10} \frac{n}{4n}$
We said Quicksort is fast, i.e. $\Theta(n \log n)$, if pivot gives an even split. (ALWAYS)

What if we always split within $\frac{1}{10}$ to $\frac{9}{10}$?

$$T(n) = T\left(\frac{n}{10}\right) + T\left(\frac{9n}{10}\right) + cn$$
We said Quicksort is fast, i.e. $\Theta(n \log n)$, if pivot gives an even split. (ALWAYS)

What if we always split within $\frac{1}{10}$ to $\frac{9}{10}$?

$$T(n) = T\left(\frac{n}{10}\right) + T\left(\frac{9n}{10}\right) + c.n$$

Any constant fraction split will give $\Theta(n \log n)$.
Repeat: if you could ensure that every pivot gives some [constant fraction of n]-split, you would get \( O(n \log n) \) ... possibly with terrible hidden constant
Repeat: if you could ensure that every pivot gives some [constant fraction of n]-split, you would get $O(n \log n)$

...possibly with terrible hidden constant

So we can't do that, but it might happen with high probability.
Repeat: if you could ensure that every pivot gives some [constant fraction of \( n \)]-split, you would get \( \Theta(n \log n) \) possibly with terrible hidden constant.

\( \Rightarrow \) we can't do that, but it might happen with high probability.

Let's look at another example: alternate balanced & unbalanced split.
Repeat: if you could ensure that every pivot gives some [constant fraction of n]-split, you would get $\Theta(n \log n)$ ... possibly with terrible hidden constant.

we can't do that, but it might happen with high probability.

Let's look at another example: alternate balanced & unbalanced split.

$L(n) = 2U\left(\frac{n}{2}\right) + \Theta(n)$
Repeat: if you could ensure that every pivot gives some 
[constant fraction of n]-split, you would get $O(n \log n)$.

...possibly with terrible hidden constant.

We can't do that, but it might happen with high probability.

Let's look at another example: alternate balanced & unbalanced split.

$L(n) = 2U\left(\frac{n}{2}\right) + \Theta(n)$

$L(n) = L(n-1) + \Theta(n)$
Repeat: if you could ensure that every pivot gives some
[constant fraction of n]-split, you would get $\Theta(n \log n)$
... possibly with terrible hidden constant

we can't do that, but it might happen with high probability.

Let's look at another example: alternate balanced & unbalanced split.

\[
\begin{align*}
L(n) &= 2U\left(\frac{n}{2}\right) + \Theta(n) \quad \text{lucky} \\
U(n) &= L(n-1) + \Theta(n) \quad \text{unlucky}
\end{align*}
\]
Repeat: if you could ensure that every pivot gives some constant fraction of $n$, you would get $\Theta(n \log n)$ ... possibly with terrible hidden constant.

$\Rightarrow$ we can't do that, but it might happen with high probability.

Let's look at another example: alternate balanced & unbalanced split.

\[
\begin{align*}
L(n) &= 2U\left(\frac{n}{2}\right) + \Theta(n) \\ 
U(n) &= L(n-1) + \Theta(n)
\end{align*}
\]

lucky \quad \text{unlucky}

\[
L(n) = 2\left[ L\left(\frac{n}{2} - 1\right) + \Theta\left(\frac{n}{2}\right) \right] + \Theta(n)
\]

\[
= 2L\left(\frac{n}{2} - 1\right) + \Theta(n)
\]
Repeat: if you could ensure that every pivot gives some [constant fraction of n]-split, you would get $\Theta(n \log n)$... possibly with terrible hidden constant.

We can't do that, but it might happen with high probability.

Let's look at another example: alternate balanced & unbalanced split.

$$L(n) = 2U\left(\frac{n}{2}\right) + \Theta(n)$$
$$U(n) = L(n-1) + \Theta(n)$$

Suppose you are lucky:

$$L(n) = 2\left[L\left(\frac{n}{2} - 1\right) + \Theta\left(\frac{n}{2}\right)\right] + \Theta(n)$$
$$= 2L\left(\frac{n}{2} - 1\right) + \Theta(n) = \Theta(n \log n)$$
Repeat: if you could ensure that every pivot gives some [constant fraction of n]-split, you would get $\Theta(n \log n)$ possibly with terrible hidden constant.

$\Rightarrow$ we can't do that, but it might happen with high probability.

Let's look at another example: alternate balanced & unbalanced split.

\[
\begin{align*}
L(n) &= 2U\left(\frac{n}{2}\right) + \Theta(n) \\
U(n) &= L(n-1) + \Theta(n)
\end{align*}
\]

\[
\begin{align*}
L(n) &= 2[L\left(\frac{n-1}{2}\right) + \Theta(n)] + \Theta(n) \\
&= 2L\left(\frac{n-1}{2}\right) + \Theta(n)
\end{align*}
\]

A note: we can avoid specific "bad" distributions by permuting the input or random pivot selection.
Repeat: if you could ensure that every pivot gives some [constant fraction of n]-split, you would get \( \Theta(n \log n) \) possibly with terrible hidden constant we can't do that, but it might happen with high probability.

Let's look at another example: alternate balanced & unbalanced split.

\[
\begin{align*}
L(n) &= 2U\left(\frac{n}{2}\right) + \Theta(n) \\
U(n) &= L(n-1) + \Theta(n)
\end{align*}
\]

\[
\begin{align*}
\text{lucky} & \quad L(n) = 2[L\left(\frac{n}{2}-1\right) + \Theta\left(\frac{n}{2}\right)] + \Theta(n) \\
\text{unlucky} & \quad = 2L\left(\frac{n}{2}-1\right) + \Theta(n) = \Theta(n \log n)
\end{align*}
\]

A note: we can avoid specific "bad" distributions by permuting the input or random pivot selection. If all input permutations are equally likely, then picking the first element is fine.