Divide & Conquer

Divide and conquer is the art of turning large problems into smaller, more manageable problems. This is typically done recursively.

Ex: \( BS(A, x, l, r) \)

\[ |A| = n \]

if \( l > r \)
    return -1

mid = \( \frac{l+r}{2} \)

if \( x = A[mid] \)
    return mid

if \( x > A[mid] \)
    return \( BS(A, x, \text{mid}+1, r) \)

return \( BS(A, x, l, \text{mid}-1) \)

What is the runtime? Why?

\( O(\log n) \)

We divide \( A \) in half each time.

\( \frac{1}{2} \leq y \leq 2^n \) for some \( m \in \mathbb{N} \) (we can first pad 0's if necessary)

Ex: \( \text{Mul} (x, y) \)

Use \( 1 \times 1 \) as size of as an abuse of notation

If \( 1x1 = 1 \)

Return \( x \times y \)

Let \( n = 1x1 \), \( m = \log(n) - 1 \)

Let \( x = x_1 x_2 \) where \( |x_1| = |x_2| = \frac{n}{2} \)

Let \( y = y_1 y_2 \) where \( |y_1| = |y_2| = \frac{n}{2} \)

Return \( 2^m x_1 y_1 + 2^m (x_1 y_2 + x_2 y_1) + x_2 y_2 \)

easy bitshift  
easy bitshift
What's the run time?

\[ T(n) = 4T\left( \frac{n}{2} \right) + O(n) \]

\[ \sum_{k=0}^{\log n} 2^k n = n \sum_{k=0}^{\log n} 2^k = n \frac{2^{\log n + 1} - 1}{2 - 1} = n (2n-1) \]

Thus \( T(n) \in \Theta(n^2) \).

What was the runtime before? \( \Theta(n^2) \)

How do we improve this?

\[ x_u y_e + x_e y_u = (x_u + x_e)(y_u + y_e) - y_u y_u - x_e y_e \]

already calculated!

\[ T(n) = 3T\left( \frac{n}{2} \right) + O(n) \]
\[
\sum_{k=0}^{\log_h n} \left( \frac{3}{2} \right)^k n = h \sum_{k=0}^{\log_h n} \left( \frac{3}{2} \right)^k \\
= \frac{n}{\frac{3}{2} - 1} \\
= 2n \left( \left( \frac{3}{2} \right)^{\log n} - 1 \right) \\
= 3n \left( \left( \frac{3}{2} \right)^{\log n} - \frac{2}{3} \right) \\
= 3n \left( n^{\log \frac{3}{2}} - \frac{2}{3} \right) \\
= 3n^{\log 3} - 2n
\]

Thus \( T(n) \in \Theta(n^{\log 3}) \).

\[\log 3 \approx 1.58\]

A big improvement!

In both cases, \( T(n) \) was most influenced by the number of branches, not the amount of work done at each recursive call.

\[
T(n) = a T\left( \frac{n}{2} \right) + f(n)
\]

\( b \) is the reduction in work
\( a \) is the spread.

\( \log_b a \) turns out to be the number of interest.

Before we had \( \log_2 4 = 2 \) and \( \log_2 3 \), the terms which dominated the multiplication runtimes.

For BS, we would have had

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

\[\log_2 1 = 0\]

and a runtime of \( \Theta(\log n) \) rather than \( \Theta(n^0) = \Theta(1) \).
For BS, we say that the work done and the number of recursive calls are comparable.

There is one other case, that where the work done is more influential than the number of recursive calls.

\[
\text{QuickSelect}(A, l, r, k) \quad \text{(Returns the } k^{\text{th}} \text{ smallest element)}
\]

If \( l = r \)
Return \( A[l] \)

Let \( p = \text{Select Pivot}(A, l, r) \)
\( p = \text{Partition}(A, l, r, p) \)
If \( k = p \)
Return \( A[p] \)
If \( k < p \)
Return \( \text{QuickSelect}(A, l, p-1, k) \)
Return \( \text{QuickSelect}(A, p+1, r, k) \)

\[
\text{Partition}(A, l, r, p)
\]
Swap \( A[p] \) and \( A[r] \)
Let \( L = l \)
Let \( R = r \)
Loop forever
\[
\text{while } l < R \text{ and } A[l] \leq A[R]
\]
\( l++ \)
\[
\text{while } A[r] > A[R]
\]
\( r-- \)
If \( l < r \)
Swap \( A[l] \) and \( A[r] \)
Else
Swap \( A[l] \) and \( A[R] \)
Return \( l \)
What is the runtime of `Partition`? Why?

$\Theta(n)$ since you must go through the entire sub array.

What is the WC runtime of `QuickSelect`? Why?

What about the BC runtime?

$O(n^2)$ if you select the worst pivot every time.

$\Omega(n)$ if you select k as the pivot.

There's a big difference between WC and BC.

What is the AC then?

It depends on how good `SelectPivot` is.

We assume that it selects a good pivot $\approx \frac{1}{2}$ the time (a good pivot is one in the middle 50% of our sub array). Then we get

$$T(n) \leq \begin{cases} 
    T(n-1) + \Theta(n) & \text{with 50\% probability} \\
    T\left(\frac{3n}{4}\right) + \Theta(n) & \text{with 50\% probability}
\end{cases}$$

Let's assume we get a bad pivot and then a good pivot. So we have

$$T(n) \leq T(n-1) + \Theta(n) \\ \leq T\left(\frac{3n}{4}\right) + \Theta(n) + \Theta(n) \\ \leq T\left(\frac{3n}{4}\right) + \Theta(n).$$

This is easy to visualize!
\[ T(n) \leq \sum_{k=0}^{\left\lfloor \log_\frac{3}{4} n \right\rfloor} \left( \frac{3}{4} \right)^k n \]
\[ = n \left( \frac{\log_\frac{3}{4} n - 1}{3 \cdot 4} \right) \]
\[ \leq n \frac{1}{1 - \frac{3}{4}} \]
\[ = 4n \]

Thus \( T(n) \in \Theta(n) \) for our AC.

Notice that \( T(n) \) is the work done on any recursive call and is more important than the number of recursive calls. We can formalize this intuition.

**Master Theorem**

Suppose \( T(n) = aT\left(\frac{n}{b}\right) + f(n) \) and \( a, b \geq 1 \), where \( \frac{n}{b} \) means either \( \lfloor \frac{n}{b} \rfloor \) or \( \lceil \frac{n}{b} \rceil \). Then \( T(n) \) satisfies:

1. If \( f(n) \in O\left(n^{\log_b(a) - \varepsilon}\right) \) for \( \varepsilon > 0 \), then \( T(n) \in \Theta(n^{\log_b(a)}) \).
2. If \( f(n) \in \Theta\left(n^{\log_b(a) - \varepsilon}\log^k n\right) \) for \( k \geq 1 \), then \( T(n) \in \Theta(n^{\log_b(a) - \varepsilon}) \).
3. If \( f(n) \in \Omega\left(n^{\log_b(a) + \varepsilon}\right) \) and \( \frac{a}{b} \leq c \) for some constant \( c > 1 \), then \( T(n) \in \Theta(f(n)) \).