Divide and Conquer Algorithms

D&C is an algorithm design technique which consists of decomposing a given instance into a number of smaller subinstances, solving these subinstances independently, then combining the solutions to obtain a solution to the original instance.

The key in understanding the efficiency of such an algorithm is that the subinstances are solved recursively.

The general template of a D&C algorithm is as follows. Let X denote an instance of some problem, and Y its solution.

\[
\text{DC}(X)
\]

1.) If X is sufficiently small
2.) Return the obvious solution
3.) Decompose X into subinstances \( x_1, \ldots, x_a \)
4.) for \( i = 1 \) to \( a \)
5.) \( y_i \leftarrow \text{DC}(x_i) \)
6.) Combine \( y_1, \ldots, y_a \) to obtain a solution \( Y \) of \( X \)
7.) Return \( Y \)
Let $|x|_i$ denote the size of instance $x_i$. Suppose each of the subinstances is of size

$$|x|_i = \frac{|x|}{b} \quad (1 \leq i \leq a).$$

Let $f(n)$ be the running time of $DC(x)$ when $|x| = n$, not counting recursive calls (i.e., $f(n)$ in the cost of decomposing, and combining).

Let $T(n)$ denote the total run time of $DC(x)$ when $|x| = n$, then:

$$T(n) = \begin{cases} 
\Theta(1) & \text{n small} \\
\alpha T\left(\frac{n}{b}\right) + f(n) & \text{otherwise}
\end{cases}$$

The asymptotic order of $T(n)$ can then be easily found by the Master Theorem, provided $\alpha, b, f$ satisfy its hypotheses.
Ex. Binary Search

Let $A$ be an array of integers (say) with $n = \text{length}[A]$. We will adopt the convention that array indices begin at 1. Thus

$$A[1...n] = (A[1], \ldots, A[n])$$

Let $A[p...r]$ denote the subarray

$$A[p...r] = (A[p], \ldots, A[r])$$

If $p > r$ we understand this to be an empty array.

Assume $A[1...n]$ is sorted in increasing order (with possible repeated elements.)

Binary search is a divide and conquer algorithm which locates a given target $t$ in the subarray $A[p...r]$. If an index $i$ is found such that $A[i] = t$, then $i$ is returned, otherwise 0 is returned.
\textbf{BinSearch} \((A, p, r, t)\) \hspace{1em} (Pre: \(A[p..r]\) sorted.)

1.) \textbf{if} \(p > r\)

2.) \textbf{return} 0

3.) \(q \leftarrow \left\lfloor \frac{p + r}{2} \right\rfloor\)

4.) \textbf{if} \(A[q] = t\)

5.) \textbf{return} \(q\)

6.) \textbf{if} \(A[q] < t\)

7.) \textbf{return} \textbf{BinSearch} \((A, q+1, r, t)\)

8.) \textbf{return} \textbf{BinSearch} \((A, p, q-1, t)\)

\textbf{Ex.} \hspace{1em} \(A = (1, 2, 3, 4, 5, 6, 7, 8, 9)\)

\hspace{2em} 1 \hspace{0.5em} 2 \hspace{0.5em} 3 \hspace{0.5em} 4 \hspace{0.5em} 5 \hspace{0.5em} 6 \hspace{0.5em} 7 \hspace{0.5em} 8 \hspace{0.5em} 9

This binary search tree represents the order in which the target \(t\) is compared to elements of \(A\). The call to BinSearch on the full array \(A[1..n]\) is just

\textbf{BinSearch} \((A, 1, n, t)\)
THEOREM (CORRECTNESS OF BinSearch)

BinSearch returns either an index $i$ such that $A[i] = t$, or returns 0 if no such $i$ exists.

Proof

Use induction on $m = r - p + 1 = \text{length}[A[p...r]]$.

I. Base

If $m = 0$ the subarray does not contain target $t$. Also $m = 0$ implies $r = p - 1 < p$, so that 0 is returned on line 2.

II. (Strong) Induction

Let $m > 0$ and assume that BinSearch returns the correct index on any subarray of length less than $m$.

Now $m > 0$ implies $r > p - 1$ whence $r > p$. Thus $t = \left\lfloor \frac{p + h}{2} \right\rfloor$ (line 3) implies $p < q < p + h$.

If $A[q] = t$ then obviously BinSearch returns correct index on line 5.

If $A[q] < t$ then $A[q + 1...r]$ is a subarray of length

$r - (q + 1) - 1 = r - q - 1 < r - p < r - p + 1 = m$. 
The Induction Hypothesis guarantees that a correct value is returned on line 7.

If on the other hand, \( A[q] \neq \top \) then \( A[p \ldots (q-1)] \) is of length

\[(q-1) - p + 1 = q - p - 1 < q - p + 1 = m,\]

so the induction hypothesis insures that a correct value is returned on line 8.

In all cases a correct value is returned, and the proof is complete. \[\Box\]

The (worst case) analysis of binary search is very simple:

\[
T(n) = \begin{cases} 
\Theta(1) & n = 0 \\
1 + \frac{T(n)}{2} + \Theta(1) & n \geq 1 
\end{cases}
\]

\[
\log_2 n^0 = n^0 = 1 = \Theta(1), \text{ so Case 2 of the Master Theorem says}
\]

\[T(n) = \Theta(\log n)\].