**Master Method**

Let $T(n)$ be a time function defined as a recurrence of the form

$$T(n) = \begin{cases} 
  c_n & \text{for } n < n_0 \\
  aT(n/b) + f(n) & \text{for } n \geq n_0 
\end{cases}$$

for some constants $a \geq 1, b > 1, c_1, c_2, \ldots, c_{n_0-1} \geq 0, T : \cdots +$.

1. if $f(n) \in O(n^k)$, such that $k < \log_b a$, then $T(n) \in \Theta(n^{\log_b a})$

2. if $f(n) \in \Theta(n^k)$, such that $k = \log_b a$ then $T(n) \in \Theta(n^k \log n)$

2g. if $f(n) \in \Omega(n^k \log^m n)$, s.t. $k = \log_b a$ & $m \geq 0$ then $T(n) \in \Omega(n^k \log^{m+1} n)$

3. if $f(n) \in \Omega(n^k)$, such that $k > \log_b a$, and $c > 0, n_0 > 0, n \geq n_0 \quad af(n/b) \leq cf(n)$, then $T(n) \in \Theta(f(n))$. 

Example: binary search

Let $n=j-i+1$ in the binary search algorithm and let $T(n)$ be a time function defined as a recurrence of the form

$$T(n) = \begin{cases} 
    c_1 & \text{for } n=1 \\
    T(n/2)+c_2 & \text{for } n\geq2
  \end{cases}$$

for some constants $c_1, c_2 \geq 0$, $T : \mathbb{N} \to \mathbb{R}^+$. 

2. if $f(n) \in \Theta(n \log^2 \log^0 n)$, then $T(n) \in \Theta(n^0 \log^1 n) = \Theta(\log n)$
Example: Merge sort

Let $n$ in the merge sort algorithm be the size to be sorted and let $T(n)$ be a time function defined as a recurrence of the form

$$T(n) = \begin{cases} 
  c_1 & \text{for } n = 1 \\
  2T(n/2) + c_2n + c_3 & \text{for } n \geq 2
\end{cases}$$

for some constants $c_1, c_2, c_3 \geq 0$, $T : \mathbb{N} \rightarrow \mathbb{R}^+$.

2. if $f(n) \in \Theta(n^\log_2 \log_\log n)$, then $T(n) \in \Theta(n^1 \log^1 n) = \Theta(n \log n)$
Example: D&Q multiplication (I)

Let $T(n)$ be a time function associated to the divide and conquer multiplication algorithm defined as a recurrence of the form

$$
T(n) = \begin{cases} 
  c_1 & \text{for } n=1 \\
  4T(n/2)+c_2n+c_3 & \text{for } n \geq 2 
\end{cases}
$$

for some constants $c_1, c_2 \geq 0$, $T : \mathbb{N} \rightarrow \mathbb{R}^+$. 

1. if $f(n) \in O(n^1)$, s. t. $1 < \log_2 4$, then $T(n) \in \Omega(n^{\log_2 4}) = \Omega(n^2)$
Example: D&Q multiplication (II)

Let $T(n)$ be a time function associated to the divide and conquer multiplication algorithm defined as a recurrence of the form

$$T(n) = \begin{cases} 
c_1 & \text{for } n=1 \\
3T(n/2)+c_2n+c_3 & \text{for } n\geq 2
\end{cases}$$

for some constants $c_1,c_2\geq 0$, $T: \mathbb{N} \to \mathbb{R}$.

1. if $f(n) \in O(n^k)$, s. t. $1<\log_2 3$, then $T(n) \in \Theta(n^{\log_2 3}) \in \Theta(n^{1.58})$
Example: binary search

Recall the recursive binary search algorithm presented earlier in the course. The running time of search(a, low, high, value), used to determine if one of a[low], a[low+1], ..., a[high] is equal to value depends on the size of high-low. As high-low increase, running time increases.

We use $T(n)$ to denote the number of steps used to execute search(a, high, low, value) where $n=high-low+1$. Calling search(a, low, high, value) could result in one of four possibilities:

1. $low > high$ so the algorithm returns $-1$.
2. $low \leq high$ and $value = a[mid]$ so the algorithm returns $mid$.
3. $low \leq high$ and $value > a[mid]$ so the algorithm returns $search(a,mid+1,high,value)$.
4. $low \leq high$ and $value < a[mid]$ so the algorithm returns $search(a,low,mid-1,value)$.
   where $mid = \text{floor}((high+low)/2)$

The first two possibilities each use some constant number of steps and the second two, by definition of $T(n)$, use $T(high-(mid+1)+1)$ and $T(mid-1-low+1)$, respectively. Thus, we see that:

\[
T(n) = c1 \quad \text{if} \ n < 1;
\]
\[
T(n) = c2 \quad \text{if} \ n \geq 1 \text{ and } value = a[mid];
\]
\[
T(n) = T(high-(mid+1)+1) + c3 \quad \text{if} \ n \geq 1 \text{ and } value > a[mid]; \text{ and}
\]
\[
T(n) = T(mid-1-low+1) + c4 \quad \text{if} \ n \geq 1 \text{ and } value < a[mid]
\]
where $c1$, $c2$, $c3$ and $c4$ are constants.
Example: binary search

We can rewrite this equation in terms of n rather than using low and high:

\[
\text{high-(mid+1)+1} = \text{high-mid} \\
\hspace{1cm} = \text{high-floor((high+low)/2)} \\
\hspace{1cm} = \text{high+ceiling(-(high+low)/2) because -floor(x) = ceiling(-x)} \\
\hspace{1cm} = \text{ceiling(high - (high+low)/2)} \\
\hspace{1cm} = \text{ceiling((high-low)/2)} \\
\hspace{1cm} = \text{ceiling((n-1)/2)}
\]

\[
\text{mid-1-low+1} = \text{mid-low} \\
\hspace{1cm} = \text{floor((high+low)/2)-low} \\
\hspace{1cm} = \text{floor((high+low)/2 - low)} \\
\hspace{1cm} = \text{floor((high-low)/2)} \\
\hspace{1cm} = \text{floor((n-1)/2)}
\]

Thus, we have:

\[
T(n) = c1 \quad \text{if } n < 1; \\
T(n) = c2 \quad \text{if } n \geq 1 \text{ and value} = a[mid]; \\
T(n) = T(\text{ceil}((n-1)/2) \ + \ c3 \quad \text{if } n \geq 1 \text{ and value} > a[mid]; \text{ and} \\
T(n) = T(\text{floor}((n-1)/2) \ + \ c4 \quad \text{if } n \geq 1 \text{ and value} < a[mid]}
\]
Example: binary search

This is called a recurrence equation for T(n). Unfortunately, recurrence equations do not tell us much about actual running so we need to derive a direct equation for T(n). This will be difficult to do with the floor and ceiling functions so we obtain a recurrence inequality:

\[
T(n) = \begin{cases} 
c1 & \text{if } n < 1; \\
T(n/2) + k1 & \text{otherwise (where } k1 = \max(c3,c4))
\end{cases}
\]

This is true because binary search \( n/2 \geq \lceil (n-1)/2 \rceil \) and \( \lfloor (n-1)/2 \rfloor \) and binary search uses the same or a larger number of steps when searching larger subsequences. We ignore the case when value = a[mid] because we are interested in the worst case running time of binary search. Finding a match never gives a worst case running time because the search stops as soon as a match is found.

If instead we set a recurrence equality:

\[
S(n) = \begin{cases} 
c1 & \text{if } n < 1; \\
S(n/2) + k1 & \text{otherwise (where } k1 = \max(c3,c4))
\end{cases}
\]

we find by the Master Method that \( S(n) \) is \( \Theta(\log n) \). However since we can argue by mathematical induction that \( T(n) \leq S(n) \) for all \( n \). Thus we conclude \( T(n) \) is \( \Theta(\log n) \).