INT102 Lec 3 Divide and Conquer

Recursive Binary Search

BINARY-SEARCH-RECURSIVE(A, v, low, $high$)

if $low > high$ **return** *false*
$mid = \lfloor (low + high) / 2 \rfloor$
if $v > A[mid]$
    return BINARY-SEARCH-RECURSIVE(A, v, $mid + 1$, $high$)
else if $v < A[mid]$
    return BINARY-SEARCH-RECURSIVE(A, v, low, $mid - 1$)
else return *mid*

Time Complexity of binary search algorithm on n numbers:

$$T(n) = \begin{cases} 1 & \text{if } n = 1, \\ T(n/2) + 1 & \text{otherwise}. \end{cases}$$

We call this formula a **recurrence**.

Recurrence

A recurrence is an equation or inequality that describes a function interns of its value on smaller inputs.
To solve a recurrence is to derive asymptotic bounds on the solution.

Substitution Method

1. Make a guess (for example: $T(n) \le 2\log n$)

Prove statement by mathematic induction:

Assume true for all $n' < n$ [Assume $T(n/2) \le 2\log(n/2)$]

$$\begin{aligned} T(n) &= T(n/2) + 1 \\ &\le 2\log(n/2) + 1 \\ &= 2(\log n - 1) + 1 \\ &< 2\log n, \end{aligned}$$

it follows that $\log(n/2) = \log n - \log 2$.

Hence $T(n) = O(\log n)$.

Example:

Prove that $T(n) = \begin{cases} 1 & \text{if } n = 1, \\ 2T(n/2) + 1 & \text{otherwise}. \end{cases}$ is $O(n)$.

Guess: $T(n) \le 2n - 1$,

Assume true for all $n' < n$ s.t.

$$\begin{aligned} T(n) &= 2T(n/2) + 1 \\ &\le 2(2 * (n/2) - 1) + 1 \\ &= 2n - 2 + 1 \\ &= 2n - 1, \end{aligned}$$

Hence, $T(n) = O(n)$.

Example 2:

Prove that $T(n) = \begin{cases} 1 & \text{if } n = 1, \\ 2T(n/2) + n & \text{otherwise}. \end{cases}$ is $O(n \log n)$.

Guess: $T(n) \le 2n \log n$

Assume true for all $n' < n$ s.t.

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &\le 2(2(n/2)\log(n/2)) + n \\ &= 2n(\log n - 1) + n \\ &= 2n\log n - 2n + n \\ &\le 2n\log n, \end{aligned}$$

Hence $T(n) = O(n \log n)$.

Depending on recurrence we observe:

$T(n) = T(n/2) + \Theta(1)$, $\quad T(n) = O(\log n)$;

$T(n) = 2T(n/2) + \Theta(1)$, $\quad T(n) = O(n)$;

$T(n) = 2T(n/2) + \Theta(n)$, $\quad T(n) = O(n \log n)$.

Merge Sort

MERGE-SORT($A[0 \ldots n-1]$)

if $n > 1$ **then**
    copy $A[0 \ldots \lfloor n/2 \rfloor - 1]$ to $B[0 \ldots \lfloor n/2 \rfloor - 1]$
    copy $A[\lfloor n/2 \rfloor \ldots n-1]$ to $C[0 \ldots \lceil n/2 \rceil - 1]$
    MERGE-SORT($B[0 \ldots \lfloor n/2 \rfloor - 1]$)
    MERGE-SORT($C[0 \ldots \lceil n/2 \rceil - 1]$)
    MERGE(B, C, A)

$T_{\text{MergeSort}}(n) = \Theta(1) + \Theta(\text{n}/2) + \Theta(\text{n}/2) + T_{\text{MergeSort}}(\lfloor n/2 \rfloor) + T_{\text{MergeSort}}(\lceil n/2 \rceil) + T_{\text{Merge}}(n)$

$= 2\,T_{\text{MergeSort}}(n/2) + \Theta(\text{n})$

$= \Theta(n \log n)$

MERGE($B[0 \ldots \text{p} - 1]$, $C[0 \ldots q-1]$, $A[0 \ldots p+q-1]$)

Set $i = 0, j = 0, k = 0$
while $i < p$ **and** $j < q$ **do**
    if $B[i] \le C[j]$ **then** set $A[k] = B[i]$, and $i = i + 1$
    else set $A[k] = C[j]$, and $j = j + 1$
    $k = k + 1$
if $i == p$ **then** copy $C[j \ldots q-1]$ to $A[k \ldots p+q-1]$
else copy $B[i \ldots p-1]$ to $A[k \ldots p+q-1]$

$T_{\text{Merge}}(n) = \Theta(\text{n})$

Time Complexity of Merge Sort

It costs $\Theta(n)$ time to MERGE, plus two $T_{\text{MergeSort}}(n/2)$ recursive calls of MERGE-SORT, giving the following recurrence:

$$T(n) = \begin{cases} 1 & \text{if } n = 1, \\ 2T(n/2) + \Theta(n) & \text{otherwise,} \end{cases}$$

hence, $T(n) = O(n \log n)$.

The master method for solving recurrences:

Master Theorem:

Let $a \ge 1$ and $b \ge 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the non negative integers by the recurrence:

$T(n) = aT(n/b) + f(n)$,

where n/b can be interpreted as either $\lfloor x/b \rfloor$ and $\lceil x/b \rceil$.

Then $T(N)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(f(n))$.

$T(n) = T(n/2) + \Theta(n)$:

Let $a = 1, b = 2$, $f(n) = \Theta(n)$, then $\log_b a = \log_2 1 = 0$.

If we pick $\epsilon = 1$, then $f(n) = \Omega(n^{\log_b a + \epsilon}) = \Omega(n^{0+1})$.

Hence using Master's Theorem, it follows that $T(n) = \Theta(f(n)) = \Theta(n)$.