CMPS 201 4-21-10

Master Theorem

Let $a \geq 1$, $b > 1$, $f(n)$ Asymptotically Positive. Define $T(n)$ By

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

Then

(1) If $f(n) = O\left(n^{\log_b a - \varepsilon}\right)$ for some $\varepsilon > 0$, then

$$T(n) = \Theta\left(n^{\log_b a}\right)$$

(2) If $f(n) = \Theta\left(n^{\log_b a}\right)$, then

$$T(n) = \Theta\left(f(n) \log n\right).$$

(3) If $f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right)$ for some $\varepsilon > 0$, and if

$$\left\{\begin{array}{l} \text{for some } c \text{ in range } 0 < c < 1 \\ \text{and all suff. large } n: \\ \qquad a f(n/b) \le c f(n) \end{array}\right\} \text{ Reg. Cond.}$$

Then

$$T(n) = \Theta(f(n))$$

Ex. $T(n) = 2T(n/2) + \dfrac{n}{\log n}$

Compare $\dfrac{n}{\log n}$ to n^1. note

$$\frac{n/\log n}{n^{1-\varepsilon}} = \frac{n^\varepsilon}{\log n} \longrightarrow \infty \quad \text{for}$$

any $\varepsilon > 0$.

$\therefore \quad \frac{n}{\log n} = \omega\left(n^{1-\varepsilon}\right)$ for any $\varepsilon > 0$

$\therefore \quad \frac{n}{\log n} \neq O\left(n^{\log_2 2 - \varepsilon}\right)$ for any $\varepsilon > 0$.

$\therefore$ We're not in case (1).

Also

$$\frac{n/\log n}{n} = \frac{1}{\log n} \longrightarrow 0 \quad \text{as } n \to \infty$$

so $\frac{n}{\log n} = o(n)$, so we're not in case (2) either.

Can't use Master theorem.

sketch of Proof of Case (1):

$$T(n) = f(n) + aT\left(\frac{n}{b}\right)$$

$$= f(n) + a\left(f(n/b) + aT\left(\frac{n/b}{b}\right)\right)$$

$$= f(n) + af(n/b) + a^2 T(n/b^2)$$

$$= f(n) + af(n/b) + a^2\left(f(n/b^2) + aT\left(\frac{n/b^2}{b}\right)\right)$$

$$= f(n) + af(n/b) + a^2 f(n/b^2) + a^3 T(n/b^3)$$

$$\vdots$$

$$= \sum_{i=0}^{k-1} a^i f(n/b^i) + a^k T(n/b^k)$$

Solve $n/b^k = 1$ for k. We get $\boxed{k = \log_b n}$

for this k we have

$$T(n) = \sum_{i=0}^{k-1} a^i f\left(n/b^i\right) + a^{\log_b n} \cdot T(1)$$

$$\therefore T(n) \geq \text{const} \cdot n^{\log_b a} = \Omega\left(n^{\log_b a}\right)$$

$$\therefore T(n) = \Omega\left(n^{\log_b a}\right)$$

must show $T(n) = O\left(n^{\log_b a}\right)$, then we have $T(n) = \Theta\left(n^{\log_b a}\right)$.

now recall hypothesis of case (1)

$$f(n) = O\left(n^{\log_b a - \varepsilon}\right)$$

for some $\varepsilon > 0$.

Thus $\exists\ c > 0$ s.t. for all sufficiently large n we have

$$f(n) \leq c \cdot n^{\log_b a - \varepsilon}$$

so for sufficiently large n and all i we have

$$f\left(\frac{n}{b^i}\right) \leq c \cdot \left(\frac{n}{b^i}\right)^{\log_b a - \varepsilon}$$

thus

$$\sum_{i=0}^{k-1} a^i f\left(\frac{n}{b^i}\right) \leq \sum_{i=0}^{k-1} a^i c \cdot \left(\frac{n}{b^i}\right)^{\log_b a - \varepsilon}$$

$$= c \cdot n^{\log_b a - \varepsilon} \cdot \sum_{i=0}^{k-1} a^i \left(\frac{1}{b^i}\right)^{\log_b a - \varepsilon}$$

$$= c\, n^{\log_b a - \varepsilon} \cdot \sum_{i=0}^{K-1} \frac{a^i}{(b^{\log_b a})^i} \cdot (b^\varepsilon)^i$$

$$= c\, n^{\log_b a - \varepsilon} \cdot \sum_{i=0}^{K-1} (b^\varepsilon)^i$$

$$= c\, n^{\log_b a - \varepsilon} \cdot \left(\frac{(b^\varepsilon)^K - 1}{b^\varepsilon - 1} \right)$$

$$= \left(\frac{c}{b^\varepsilon - 1} \right) \cdot \frac{n^{\log_b a}}{n^\varepsilon} \cdot \left((b^{\log_b n})^\varepsilon - 1 \right)$$

$$= \text{const} \cdot \frac{n^{\log_b a}}{n^\varepsilon} \left(n^\varepsilon - 1 \right)$$

$$= \text{Const} \cdot n^{\log_b a} - \text{const}\, n^{\log_b a - \varepsilon}$$

$$\leq \text{Const} \cdot n^{\log_b a} = O(n^{\log_b a})$$

So

$$T(n) = \sum_{i=0}^{k-1} a^i f(n/b^i) + n^{\log_b a} \cdot T(1)$$

$$= O(n^{\log_b a}) + O(n^{\log_b a})$$

$$\therefore T(n) = O(n^{\log_b a})$$

$$\therefore T(n) = \Theta(n^{\log_b a}) \quad \text{///}$$

Divide & Conquer Algorithms

Ex. Binary Search

Let A be an array, say

$$n = \text{length}[A]$$

notation: array indices start at 1,

so

$$(A[1], A[2], \ldots, A[n])$$

$$= (A_1, A_2, \ldots, A_n)$$

$$= A[1 \ldots n]$$

The subarray from index p to index r is denoted

$$A[p \ldots r]$$

where $1 \le p \le n$ and $1 \le r \le n$.

Convention: if $p > r$, then above subarray is empty.

Assume: $A[1 \ldots n]$ is sorted in increasing order. (Possible repeated elements.)

Problem: Given a target t, Determine an index q such that $A[q] = t$. Return that index q if it exists, return 0 otherwise.

BinarySearch(A, p, r, t) { Pre cond: $A[p \ldots r]$ sorted }

Searches the subarray: $A[p \ldots r]$ for target t

Top level call is:

BinSearch$(A, 1, n, t)$

BinSearch(A, p, r, t)

1.) if $p > r$

2.) return 0

3.) $q \leftarrow \lfloor \frac{p+r}{2} \rfloor$

4.) if $A[q] = t$

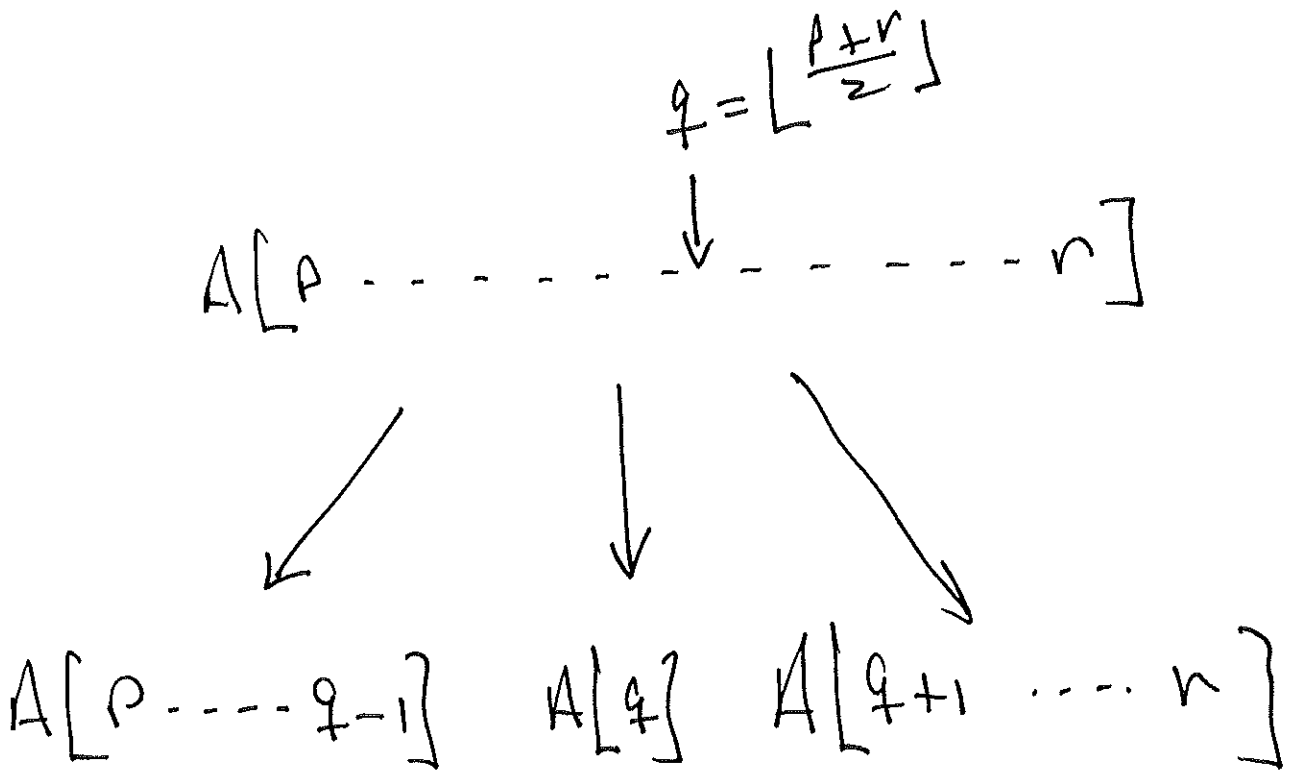
5.) return q

6.) else if $A[q] < t$

7.) return BinSearch(A, q+1, r, t)

8.) else

9.) return BinSearch(A, p, q-1, t)



Theorem

Given $A[p \ldots r]$ sorted, BinSearch(A, p, r, t) returns either an index j s.t. $p \leq j \leq r$ and $A[j] = t$, or returns 0 if no such j exists.

Proof: use (strong) induction on

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

I. If $m = 0$ then $A[p..r]$ is empty, so does not contain t.

$0 = m = r - p + 1 \Rightarrow r = p - 1 < p$

so in this case 0 is returned on line 2, As it should be,

II. Let $m > 0$. Assume that BinSearch(...) returns correct index when called on any subarray of length $< m$.

now $m > 0 \Rightarrow r - p + 1 > 0 \Rightarrow r > p - 1$

$\Rightarrow r \geq p$,

so test on line (1) is false.

thus $q = \lfloor \frac{p+r}{2} \rfloor$ implies $p \leq q \leq r$.

- If $A[q] = t$, then BinSearch(..) returns correct index on line 5.
- If $A[q] < t$, then line 7 recurs on subarray $A[q+1 .. r]$, which has length:

$$r - (q+1) + 1 = r - q \leq r - p < r - p + 1 = m$$

By ind. hypothesis, correct index is returned on line 7.

9 If $A[q] > t$, then return on subarray $A[p \ldots (q-1)]$ on line 9 this array has length

$$(q-1) - p + 1 = q - p \leq r - p < r - p + 1 = m.$$

∴ By Ind. Hyp. Correct Index is returned on line 9.

///