CSC236 fall 2018

more complexity: mergesort

This week's theme: sometimes there is no induction...

Danny Heap

heap@cs.toronto.edu / BA4270 (behind elevators)

http://www.teach.cs.toronto.edu/~heap/236/F18/

416-978-5899

Using Introduction to the Theory of Computation, Chapter 3

Outline

vexing complexity

mergesort

Divide-and-conquer

Notes

Upper bound on $T(n)$

trouble!

We tried to use induction to prove T(n) <= c lg(n), but in the induction step we ended up with T(n) <= .... 1 - c + c lg(n+1)

We have no control over c, and thus no way of knowing that 1 - c is negative enough to make the entire expression <= c lg(n)..................darn!

Various tricks were suggested. We ended up strengthening the claim to:
T(n) <= c lg(n-1).... which was provable using induction, and itself implies the original claim.
However, it feels a bit as if we need to discover a new trick for each bound on each recurrence.

What follows is a single "trick" that will give us \Theta bound on many recurrences, provided the recurrence is nondecreasing...

recurrence for MergeSort

A: list of comparables
b: beginning index to sort
e: end index to sort
n = e-b+1

```
MergeSort(A,b,e) -> None:
    if b == e:  return          cost: c
    m = (b + e) / 2             c1
    MergeSort(A,b,m)            T(ceiling(n/2))
    MergeSort(A,m+1,e)          T(floor(n/2))
    # merge sorted A[b..m] and A[m+1..e] back into A[b..e]
    B = A[:] # copy A           c2xn
    c = b                       c3
    d = m+1                     c4
    for i in [b,...,e]:         c5xn
        if d > e or (c <= m and B[c] < B[d]):
            A[i] = B[c]
            c = c + 1
        else:  # d <= e and (c > m or B[c] >= B[d])
            A[i] = B[d]
            d = d + 1
```

T(n) = { c if n = 1; T(ceiling(n/2)) + T(floor(n/2)) + n if n > 1

other than the two recursive calls the remaining code is linear, plus some constant statements. I will combine all of these into one expression --- n --- neglecting the coefficient, and also neglecting any constant terms. If you work through the following slides including those, it will work out to the same bound

Unwind (repeated substitution)

$T(n) = 2T(n/2) + n$

suppose n is a power of 2, so floor(n/2) = ceiling(n/2), i.e. n = 2^k for some natural number k, then...

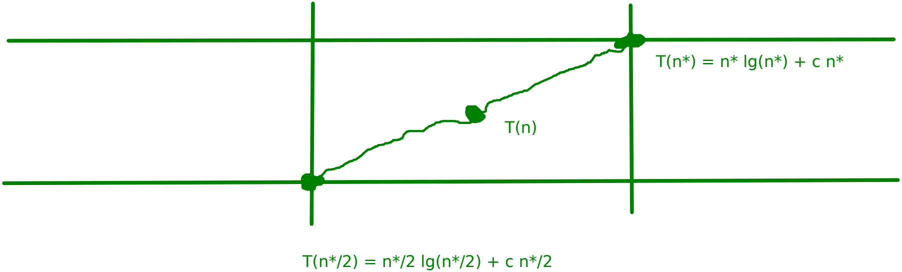
= 2(2T(n/2^2) + n/2) + n = 2^2T(n/2^2) + 2n
= 2^2(2T(n/2^3) + n/2^2) + 2n = 2^3T(n/2^3) + 3n
=
= ...(intuition happening here)... prove this conjecture using induction...
=
= 2^k T(n/2^k) + kn = nc + kn = nc + lg(n) n = n lg(n) + cn

This *conjecture* suggests a closed form for special values: power of 2. We want to extend this to upper and lower bounds for other natural numbers n.

Prove that T is non-decreasing <-- you need to do this...

```
Notation: define n* = 2^{ceiling(lg n)}  # next highest power of 2
inequality:        ceiling(lg n) - 1 < lg n <= ceiling(lg n)  # by definition of ceiling, csc165 exercise
    ==>            2^{ceiling(lg n) - 1} < 2^{lg n} <= 2^{ceiling(lg n)}
    ==>             n*/2 < n <= n*
Examples:
1* = 1
2* = 2
3* = 4* = 4
5* = 6* = 7* = 8* = 8
9* = 10* = 11* = 12* = 13* = 14* = 15* = 16* = 16
etcetera...
```

See Course Notes, Lemma 3.6 Exercise: Prove the recurrence for binary search is non-decreasing. . . see assignment #2!



Prove $T \in O(n \lg n)$ for general case

$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n$

Note: I start the proof with d = ??? and B = ???, and in the course of the proof I find conditions on what d and B can be.

Let d = ??? 2(2+c). Then d \in \R^+. Let B = ??? 2. Then B \in \N.
Let n be an arbitrary natural number no smaller than B.
Then:

```
T(n) <= T(n*)                                      # since T is nondecreasing (must be proved)
     = n* lg(n*) + c n*                            # by the still-to-be-proved unwinding conjecture
     <= 2n lg(2n) + 2cn                            # n > n*/2 ==> 2n > n*
     = 2n(lg(2n) + c) = 2n(lg(2) + lg(n) + c)      # lg(2) = 1
     = 2n((1+c) + lg(n)) <= 2n((1+c) lg(n) + lg(n))  # n>= 2 => lg(n) >= 1
     = 2n lg(n) (2 + c) <= d n lg(n)               # d >= 2(2+c)
```

divide-and-conquer general case

k: non-recursive cost, when n < b
b: number of almost-equal parts we divide problem into
a1: number of recursive calls to ceiling, a2: number of recursive calls to floor, a number of recursive calls
f: cost of splitting, and later recombining, the parts, we HOPE it is polynomial, i.e. n^d

divide-and-conquer algorithms: partition problem into b *roughly* equal subproblems, solve, and recombine:

$$T(n) = \begin{cases} k & \text{if } n \leq B \\ a_1 T(\lceil n/b \rceil) + a_2 T(\lfloor n/b \rfloor) + f(n) & \text{if } n > B \end{cases}$$

where $b, k > 0$, $a_1, a_2 \geq 0$, and $a = a_1 + a_2 > 0$. $f(n)$ is the cost of splitting and recombining.

divide-and-conquer Master Theorem

MergeSort: a = 2, b = 2 , d = 1 2 = 2^1
binary search: a = 1, b = 2, d = 0 1 = 2^0

If f from the previous slide has $f \in \theta(n^d)$, then

$$T(n) \in \begin{cases} \theta(n^d) & \text{if } a < b^d \\ \theta(n^d \log_b n) & \text{if } a = b^d \\ \theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

, so log_b a < d
, so log_b a = d
, so log_b a > d

Proof sketch

1. Unwind the recurrence, and prove a result for $n = b^k$

   See "Notes" for details

2. Prove that T is non-decreasing

   Use lemma 3.6 as a template

3. Extend to all n, similar to MergeSort

   ... just as we did in in the big-Oh and \Omega for MergeSort --- no induction!

Notes

assume $n = b^i$, for some natural number i, and assume $f \in \Theta(n^d)$

$$
\begin{aligned}
T(n) &= a^1T(n/b^1) + cn^d \\
&= a(aT(n/b^2) + c(n/b)^d) + cn^d = a^2T(n/b^2) + (a/b^d)cn^d + cn^d \\
&= a^2(aT(n/b^3) + c(n/b^2)^d) + (a/b^d)cn^d + cn^d \\
&= a^3T(n/b^3) + (a^2/b^{2d})cn^d + + (a/b^d)cn^d + cn^d \\
&= a^3T(n/b^3) + (a/b^d)^2cn^d + + (a/b^d)^1cn^d + (a/b^d)^0cn^d \\
&= \\
&= \text{.... (intuition hums along here...)} \\
&= \\
&= a^i k + c n^d \sum_{j=0}^{j=i-1} (a/b^d)^j \\
&= a^{\log_b n} k + c n^d \sum_{j=0}^{j=\log_b(n) - 1} (a/b^d)^j \\
&= n^{\log_b a} k + c n^d \sum_{j=0}^{j=\log_b(n) - 1} (a/b^d)^j
\end{aligned}
$$

Note that $b^{xy} = (b^x)^y = (b^y)^x$

So, $a^{\log_b n} = (b^{\log_b a})^{\log_b n} = (b^{\log_b n})^{\log_b a} = n^{\log_b a}$

Notes

Proof that T \in \Omega(n lg n)

Let d = ??? 1/4. Then d \in \R^+. Let B = ??? 4. Then B in \N.
Let n be an arbitrary natural number no smaller than B. Then:

```
T(n) >= T(n*/2)                    # since T is nondecreasing... you did prove this, didn't you?
      = n*/2 lg(n*/2) + cn*/2      # from our unwinding conjecture, which needs to be proved
      >= n/2 lg(n/2) + cn/2        # n* >= n ==> n*/2 >= n/2
      = n/2(lg(n) - lg(2)) + cn/2 = n/2(c - 1 + lg(n))
      = n/2(lg(n)/2 + lg(n)/2 - 1 + c)
      >= n/2(lg(n)/2)              # since n>=4 then lg(n)/2 >= 1 and c > 0
      >= d n lg(n)                 # since d = 1/4
```