T1 – end of lecture – Q4 – day still not graded.
A1 – *?!!? –

CSC236 fall 2012

more complexity: mergesort

read web page note on grading.

Danny Heap
heap@cs.toronto.edu
BA4270 (behind elevators)
http://www.cdf.toronto.edu/~heap/236/F12/
416-978-5899

Using Introduction to the Theory of Computation, Chapter 3

Outline

Upper bound on $T(n)$

trouble!

$\lfloor \quad \rfloor \quad \lceil \quad \rceil$

$\frac{n+1}{2}, \quad \frac{n-1}{2}, \quad \top \uparrow$

recurrence for MergeSort

$$T = \begin{cases} 1 & n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n + 1 & n > 1 \end{cases}$$

```
MergeSort(A,b,e):
    if b == e:  return
    m = (b + e) / 2
    MergeSort(A,b,m)
    MergeSort(A,m+1,e)
    # merge sorted A[b..m] and A[m+1..e] back into A[b..e]
    for i = b,...,e:  B[c] = A[c]
    c = b
    d = m+1
    for i = b,...,e:
        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
```

Unwind (repeated substitution)

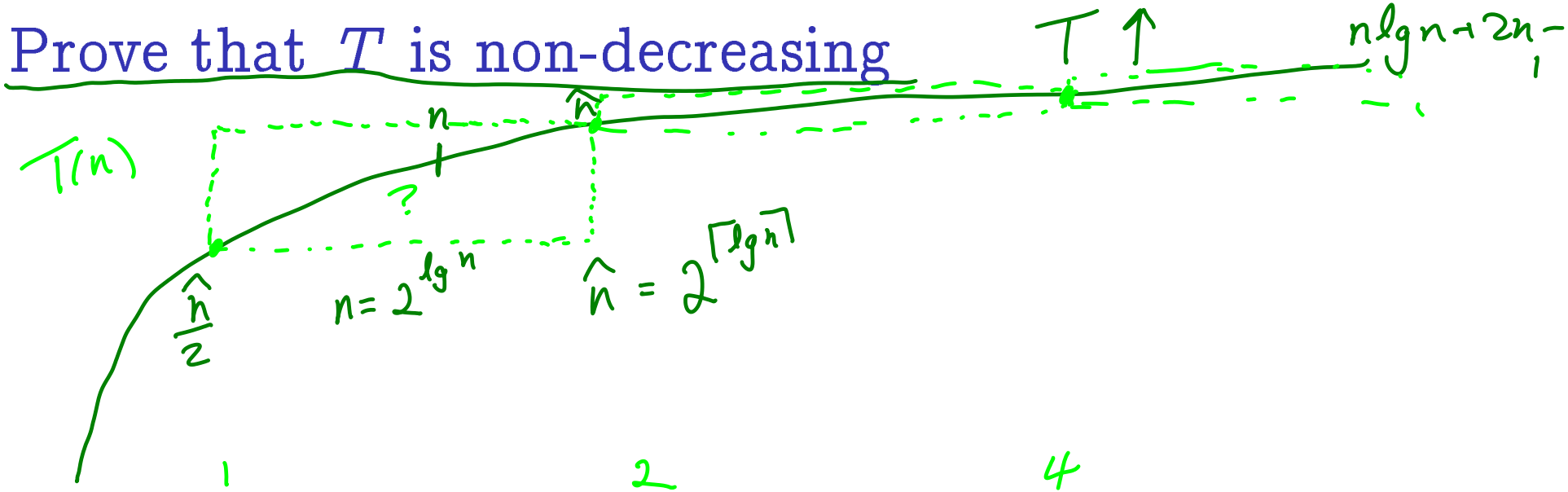
$T(n) = 2T(n/2) + n + 1$

$n = 2^k$

$\vdots$

$T(n) = n \lg n + 2n - 1$

Prove that T is non-decreasing



See Course Notes, Lemma 3.6 Exercise: Prove the recurrence for binary search is non-decreasing

So, for n not necessarily an integer power of 2, we have $\frac{\hat{n}}{2} \leq n \leq \hat{n}$

use $T(\hat{n}) = \hat{n} \lg \hat{n} + 2\hat{n} - 1$

$T(n)$

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

$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n + 1$: Sometimes stronger claim helps ($T(n) \le c\lg(n-1)$)

$\hat{n}/2 < n \le \hat{n}$

Claim $T(n) \le c\, n \lg n$ [we'll figure c and B later]

$$T(n) \le T(\hat{n}) + T\uparrow$$
$$\le \hat{n}\lg\hat{n} + 2\hat{n} - 1 \quad \# \text{ unwinding}$$
$$\le 2n \lg 2n + 4n - 1 \quad \# \quad n > \frac{\hat{n}}{2}$$
$$= 2n(\lg n + 1) + 4n - 1 = 2n\lg n + 6n - 1$$
$$\le 2n\lg n + 6n$$
$$\le 2n\lg n + 6n\lg n \quad \# \text{ provided } n \ge 2$$
$$= 8\, n \lg n$$

So $\exists\, B \in \mathbb{N}\ (B = 2),\ \exists\, c \in \mathbb{R}^+\ (c = 8),\ \forall n \in \mathbb{N},$
$n \ge B \Rightarrow T(n) \le c\, n \lg n$

General case

Class of 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 $a_1 + a_2 > 0$. $f(n)$ is the cost of splitting and recombining.

Master Theorem

Merge Sort: $d = 1$, $a = 2$, $b = 2$

$2 < 1^1$ $\quad 2 = 2^1$ ✓

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

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

$\theta(n^1 \log n)$.

Proof sketch

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

2. Prove that T is non-decreasing

3. Extend to all n, similar to MergeSort

Notes