Announcement: AWC 400 Level Lecture Series

Note Title 10/23/2007

The AWC will be running its 400 level lecture series (400 LLS) on ~~October 17,18 and 19~~ (5-6pm) in CSIC 3117. FREE FOOD.
Today, Tomorrow, + Thu

Professors who will be teaching the 400 level courses in the spring will each spend 15 minutes talking about what material the course will cover and how they intend to run it. This is a great opportunity to help you determine which 400 level course you will be interested in taking next semester.

Schedule:

Tuesday
Kruskal 451
Getoor 421
Pop 423

Wednesday
Hick 412
Golbeck 498N
Zelkowitz 430
Washington 456

Thursday
Jacobs 427
Hugue 420
Duraiswami 460

Lower Bounds

Finding a value in a list with Divide and Conquer:

Any Divide and Conquer algorithm that recurses both sides has this form:

$T(n) = T(an) + T((1-a)n) + b$,

where minimum value of b = 1.

$\boxed{\text{This is always} \geq n.}$ Can prove it. We don't have the tools yet. How do we know we won't do better?

TODAY's topic: Lower Bounds.

Lower Bounds

Note Title 10/23/2007

- Previously: We determined upper + lower bounds for a given algorithm.

- We can also determine upper + lower bounds for a given problem.

- Upper bound of a problem: Worst case runtime of the most efficient algorithm we have available so far.

- Lower bound of a problem: Minimum amount of work necessary for any and all solutions in worst case.

(Typically determined by an abstract characterization of a class of such algorithms, eg., "comparison sorts" viewed abstractly in terms of "decision trees".)

- Lower bound is NOT the runtime of the best algorithm we can think of.
- Lower bound CANNOT be created using the techniques we know of so far.

Side Comment:

The "Simplex" problem in linear algebra is an interesting one to consider.

Dantzig invented simplex when solving what he thought was a HW problem!

- algorithms exist with polynomial *expected* runtime, but this doesn't change the upper bound of the problem
- people came up with new pivot rules for the Dantzig algorithm, but there has always been *some* input that would force the algorithm to run in exponential time
- the question of whether a pivot rule exists that would ***not*** have any cases end up leading to worse than polynomial time is still (as far as I know) an open one…

Moral: Cleverness exists
Theoretical: exponential
In practice: polynomial

What kinds of problems can we talk about?

- Merging two lists
- Searching
- Sorting

We can talk about the upper and lower bounds of these problems

Lower Bound: Any algorithm (no matter how clever) that you can come up with will have to do at least this much work.

In a $\boxed{\text{comparison based model}}$, e.g., for the three problems above, we must show that there are at least a certain number of comparisons in the worst case.

MERGE: merging 2 sorted lists, each of size n.
- We saw that this takes 2n-1 comparisons.
- This is an upper bound.
- What is the lower bound?

Worst case input is what? Interleaved

Adversarial Argument

interleaved case:

a_1	b_1
a_2	b_2
a_3	b_3
$\vdots$	$\vdots$
a_n	b_n

a_1	b_1	a_1, b_1
a_2	b_2	a_2, b_1
a_3	b_3	a_2, b_2
a_4	b_4	$\vdots$
a_5	b_5	a_5, b_5

$2n-1$ $\quad$ 9

merge

a_1
b_1
a_2
b_2
a_3
b_3
$\vdots$
a_n
b_n

In this case, I MUST compare a_1 and b_1.
b_1 and a_2.
a_2 and b_2.
b_2 and a_3
a_3 and b_3
b_3 and a_4.
$\vdots$
a_{n-1} and b_{n-1}.
b_{n-1} and a_n
a_n and b_n

TOTAL: $2n-1$!

$a_1, b_1, a_2, b_2, a_3, b_3, a_4, b_4, a_5, b_5$

9 pairs! $(2n-1)$

$a_1 b_1$
$a_2 b_1$
$a_2 b_2$
$a_3 b_2$
$a_3 b_3$
$\vdots$

$a_5 b_5$

So lower bound of the Merge PROBLEM is $2n-1$.

If I make fewer comparisons, I may sometimes get the answer!

inputs that bring out worst-case behavior of all algorithms that solve a problem is called adversarial input.

idea: an adversary would give you this bad input in an attempt to make your algorithm look bad

How do we come up w/ adversarial input?

1. Look at best algorithm (UB)
2. Figure out worst possible input
3. Find way to mathematically describe input + prove it is a bad case.

Must fully understand problem to come up w/ adversarial case

Find-a-value problem.

CMSC 250 Pf by contradiction. Assume n-1. Show that for some i there exists an unchecked element a_i. So n-1 is not enough.

if you have an algorithm
that only made $n-1$ comparisons,
an adversary could come up
with an input
where the value you want
is the one you didn't look at.

So your algorithm would return
not-in-list when the
value actually is in list.

So lower bound cannot be $n-1$.

$$\boxed{\begin{array}{l}\text{upper bound is } n \\ \text{(just go through list)} \\ \text{so lower bound is } n.\end{array}}$$

$$\frac{\begin{array}{c}1 < 2 \\ 2 < 3\end{array}}{1 < 3\,?}$$

Say list is sorted.

and you want to find a value.

What is the upper bound of this problem?

– $O(\log_2 n)$ (from Binary Search).

Lower Bound?

Hashing? Adversary can give us input where all values hash to same key. Then you have $O(n)$ behavior.

As soon as we have transitivity, things are better.

$1 < 2$
$2 < 3$

Don't need to ask $1 < 3$!

Generic
Decision Tree Technique

Describe # comparisons needed as a tree. Set of comparisons needed for one runtime execution is a <u>path</u> in this tree.

Each level describes one comparison made.

The node selected from each level depends on result of previous comparisons.

Leaves are all possible outputs.

The number you are looking for is in position 1
" " " " " " " " position 2
⋮
" " . . . . n

All possible out comes:
- The # you are looking for is in position 1
- The # you are looking for is in position 2, etc.

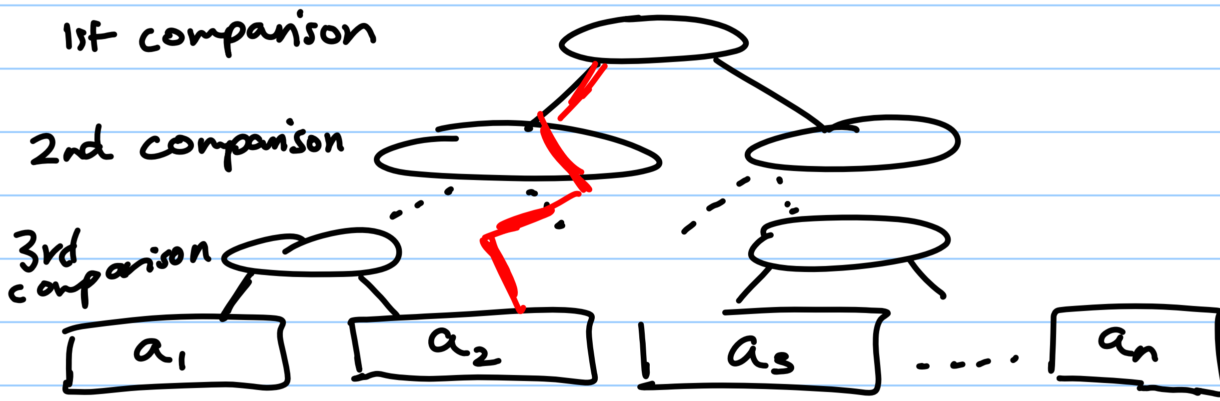
→ n leaves

How many leaves?

So find Find-a-value (ordered), you have a list $a_1\ a_2\ \dots\ a_n$.

Possible answers are:
$a_1\quad a_2\quad a_3\quad \dots\quad a_n$ and not-in-list.

Ignore not-in-list for now.



comparisons in a path is (# levels − 1)

levels in this tree is $\log_2 n + 1$

max # comparisons is $\log_2 n$.

Actual work you've done is in the path, not in the leaves. (Not linear).

Now think about sorting. How many leaves?