LOWER BOUNDS

Tuesday, November 13, 2018 5:04 PM

Ⓘ ANALYSIS : Σ, recurrence

(II) DESIGN : brute, transform, decrease, divide, greedy, dynamic programming

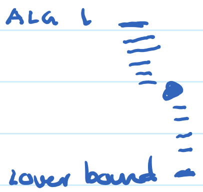
(III) HARD PROBLEMS

LOWER BOUNDS

Def : Lower bound of a problem is a minimum amount of work required to solve that problem

If there is an algorithm for a problem whose running time is the same as a lower bound, we say the lower bound is TIGHT

Problem :



TECHNIQUES FOR ESTABLISHING LOWER BOUNDS

Ⓘ Trivial lower bounds : minimum amount of work required to read input / write output for a given problem

Ex : Generating all subsets

INPUT: n items

OUTPUT: all possible subsets of the n item

A, B

$\emptyset$, {A}, {B}, {A,B}

INPUT LOWER BOUND : $\Omega(n)$

OUTPUT LOWER BOUND : $\Omega(2^n)$

This lower bound is tight, because there exist a decrease-conquer solution in time $O(2^n)$

Ex : GENERATING PERMUTATIONS

INPUT : n items

OUTPUT : all permutations of the n items

A, B, C

A B C	B A C	C A B
A C B	B C A	C B A

Input lower bound: $\Omega(n)$

Output lower bound: $\Omega(n!)$

This lower bound is TIGHT because TROTTER-JOHNSON alg runs in time $O(n!)$

Ex: Evaluating polynomials:

INPUT: $p(x) = c_0 x^0 + c_1 x^1 + \cdots + c_n x^n$ $[c_0, c_1, \ldots, c_n]$, a real number x_0

OUTPUT: $p(x_0)$

$p(x) = x^2 - 2x + 3$, $x_0 = -1$

$p(-1) = (-1)^2 - 2(-1) + 3 = 6$

Input lower bound: $\Omega(n)$

Output lower bound: $\Omega(1)$

This lower bound is TIGHT, because Horner's algorithm runs in time $O(n)$

Ex: Matrix Multiplication

INPUT: 2 $n \times n$ matrices $A_{n \times n} \cdot B_{n \times n}$

OUTPUT: AB

n^3 - high school

Input Lower Bound: $\Omega(n^2)$

$n^{2.3\ldots\ldots}$ divide-conquer

Output Lower Bound: $\Omega(n^2)$

NOT TIGHT: best alg $O(n^{2.3})$ ↓ ↑ n^2 -

Ex: TRAVELING SALESMAN $A \longleftrightarrow B$

INPUT: n cities, $\binom{n}{2}$ cost to go from A to B or vice versa

OUTPUT: cheapest cost of a tour on the n vertices (every city is visited exactly once)

2^n

INPUT LOWER BOUND: $\Omega(n^2)$

OUTPUT LOWER BOUND: $\Omega(1)$

NOT TIGHT: best alg $\Omega(2^n)$

n^2

2) ADVERSARY ARGUMENT

Ex: Maximum of an unsorted array using only COMPARISONS

CLAIM: $\Omega(n-1)$ comparisons

Proof:

Take an unsorted array whose elements are unique.

Each comparison creates a LOSER and a WINNER

Any COMPARISON-BASED alg for this problem must create $n-1$ losers

Assume by contradiction that $A[i]$ and $A[j]$ never lost

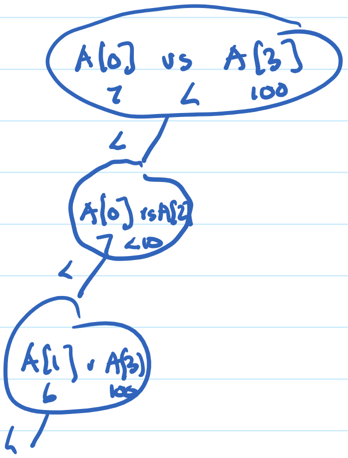
$A[i]$ $A[j]$

→ 100   ~~10~~ 101

Run alg again on same array, but replace A[i] by A[i] + 1

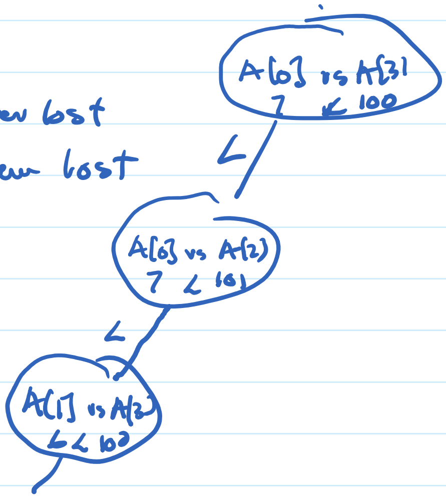
	0	1	2	3
A =	7	6	10	100

A	7	6	101	106



A[3] never lost

A[2] never lost



A[3]=100 is the maximum

A(3)= 100 is max

This is a TIGHT lower bound, because

```
ans = A[0]
for (i = 1; i < n; ++i)
    if (ans < A[i])
        ans = A[i]
return ans;
```

makes n-1 comparisons

Ex. MERGING

INPUT: 2 SORTED A[0..n-1], B[0..n-1]

OUTPUT: combined sorted array C[0.. 2n-1]

Comparison-based alg (only operations allowed on array elements is <)

CLAIM: $\Omega(2n-1)$ comparisons

Proof: Pick array elements so that

$a_0 < b_0 < a_1 < b_1 < \cdots < a_{n-1} < b_{n-1}$

A = [5, 20, 100]

B = [7, 50, 201]

$5 <^{7} < 20 <^{50} < 100 <^{201}$

Any comparison-based alg must compare

$a_0 < b_0$

$b_0 < a_1$

$a_1 < b_1$

$\vdots$

2n-1 comparisons

A = [5, 50, 100]

B = [7, 20, 201]

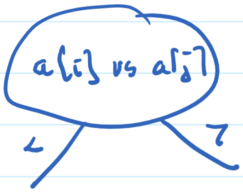
5 < 7 < 50 < 20 < 100 < 201

$$\left.\begin{matrix} a_1 < b_1 \\ \vdots \\ a_{n-1} < b_{n-1} \end{matrix}\right\}$$ comparisons

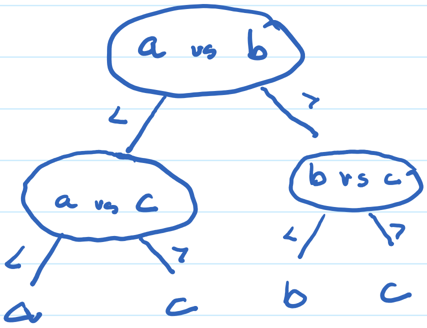
$5 < 7 < 50 < 20 < 100 < 200$

3) DECISION TREE METHOD

A comparison-based alg can be represented as a binary tree, assuming that the array elements are unique

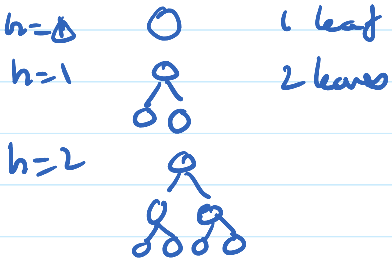


Ex: comparison-based alg to find the minimum of 3 distinct numbers
a, b, c



2 observations: 1) each possible outcome must appear as a leaf
(may be more than once)

2) a binary tree with n leaves must have height $\geq \log_2 n$
⇐ a binary tree with height h has at most 2^h leaves



CLAIM: any comparison-based SORTING ALG must make $\Omega(n \lg n)$ comparisons

Proof: Any sorting alg must have n! outcomes

a b c

a b c
a c b
b a c
b c a

have $n!$ outcomes

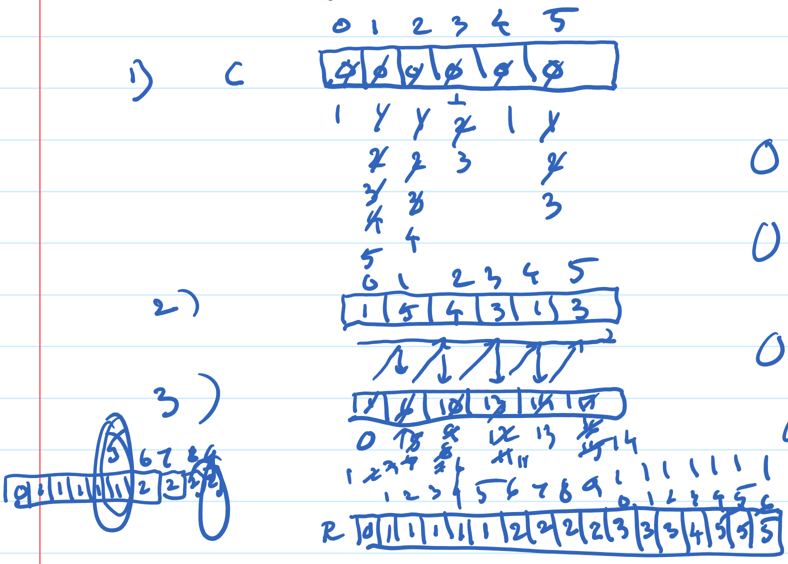
acb
bac
bca
cab
cba

So any decision tree for a COMPARISON-BASED sorting alg must have height $\log_2(n!) \simeq \Theta(n\lg n)$

A LINEAR-TIME SORTING ALGORITHM

Ex: 3, 1, 3, 2, 5, 4, 5, 3, 1, 2, 1, 0, 1, 2, 1, 2, 5

Note: array elements are INTEGERS between 0 and 5



```
          COUNTING_SORT(A[0..n-1], MAX)
          {
O(MAX)      for (i=0; i ≤ MAX; ++i)
                C[i] = 0;
O(n)        for (i=0; i<n; ++i)
                ++C[A[i]];
O(max)      for (i=1; i ≤ MAX; ++i)
                C[i] += C[i-1];
O(n)        for (i=n-1; i≥0; --i)
                R[--C[A[i]]] = A[i]
```

So if MAX ∈ O(n), then COUNTING SORT runs in time O(n)

4) REDUCTION

Suppose solution to problem B can be modified to solve problem A we say A REDUCES TO B and write $A \le B$

Intuitively, A is no harder than B

Ex: EVEN

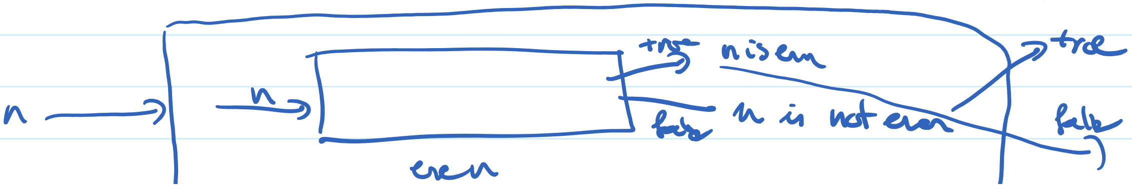
INPUT: integer n

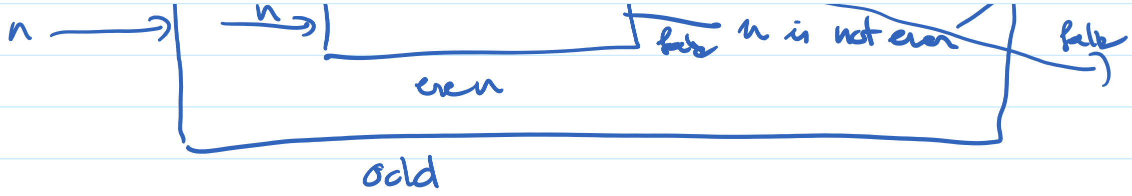
OUTPUT: true iff n is even

ODD

INPUT: integer n

OUTPUT: true iff n is odd





ODD $\leq$ EVEN

Ex: GCD

Input: integers a, b

Output: gcd (a, b)

LCM

Input: integers a, b

Output: lcm (a, b)

$a, b = 2, 6$

$\gcd(2,6) = 2$

$\text{lcm}(2,6) = 6$

$$\text{lcm}(a,b) = \frac{a * b}{\gcd(a,b)}$$

LCM $\leq$ GCD

Ob.: If $A \leq B$
and f is a lower bound on A, then
f is a lower bound on B.