RESCHEDULING THE RESCHEDULED CLASS

Does May 1st work? Email if not.

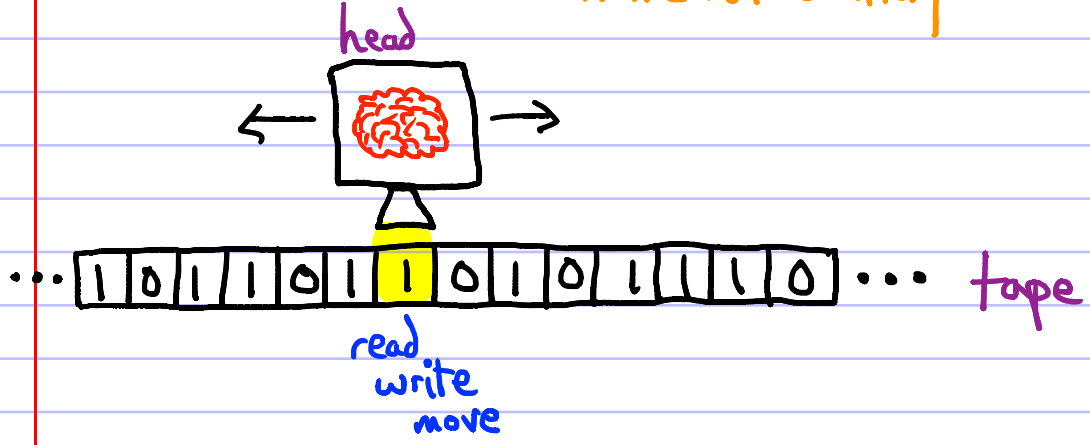
May 1st is during reading period, but will let us avoid using final exam time for makeup.

MODELS OF COMPUTATION

The Turing Machine: the first theoretical model of computation.

Two parts:

1. A head containing an algorithm and program counter. (A fixed algorithm of O(1) size)
2. A tape containing an input, and infinite additional space. (finite but arbitrary)



Problem: given a string s with $s \in \{0,1,2\}^*$, is $s = w2^{|w|}w$ for some string $w \in \{0,1\}^*$?

10102222101 0 YES 1101222211011 NO 1102222110 NO

(Turing machine) Algorithm:
1. while (symbols remain):
2. read first symbol 0/1=a, erase it
3. go right until first 2, erase it
4. go right until first 0/1=b, erase it
5. check a=b.
6. if anything failed, return NO
7. return YES

(lines 2–4: $\frac{2}{3}n$)

$\Theta(n^2)$ total steps

Thm [Hennie '65]: Any TM algorithm for this problem takes $\Theta(n^2)$ steps.

Proof Idea: Left & right third modifications must happen in synchrony, otherwise machine is wrong. Synchrony takes $\Omega(n)$ time per change.

A "regular" algorithm: O(n) [for i from 1 to $\frac{n}{3}$:
O(n) total time O(1) [check that s[i] == s[$\frac{2}{3}n+i$]
check s[$i+\frac{n}{3}$] == 2
if anything failed, return NO]
return YES

A "regular" algorithm and running time analysis assumes RAM model:

Random Access Machine. Look anywhere in O(1) time.

Turing machine vs. RAM: which is a better model?

DISTANCE MATTERS DISTANCE IRRELEVANT

Goal: model that only permits reasonable operations that incur reasonable time/space costs. Reasonable = natural, realistic, simple.

RAM is a family of models: locality-independent memory access cost.

What about operation costs? How long does computing n! take?

```
t=1
for i in 1 to n: ] Θ(n)
    t=t·i ] O(1)?
return t
```

Θ(n) time to compute n!?

Let $a,b \in \mathbb{N}$. How long does "a*b" take?

It is typical to assume O(1).

n can be written in log(n) bits.

n! takes Θ(nlogn) bits.

Θ(n) time to compute an output of size Θ(nlogn)? Not possible.

Karatsuba algorithm: $O([\log n]^{1.585})$ time to multiply two integers i_1, i_2 with $n = \max(i_1, i_2)$. Not O(1)

Naive multiplication:

Given two binary numbers $a = a_n a_{n-1} \cdots a_0$ & $b = b_n b_{n-1} \cdots b_0$:

```
t = 0                                          } Θ(n²) time
Θ(n) [ for i from 0 to n:
Θ(n)     [ if a_i == 1:
             t += b << i   // b * 2^i = b << i
return t
```

$n=4$, $a = 10110 = 22$, $b = 11101 = 29$

$$a*b = a_0\cdot[b<<0] + a_1\cdot[b<<1] + a_2\cdot[b<<2] + a_3\cdot[b<<3] + a_4\cdot[b<<4]$$

$$= b<<1 + b<<2 + b<<4$$

$$= \begin{array}{r} 111010 \\ 1110100 \\ +\ 111010000 \\ \hline 100111110 \end{array} = 512+64+32+16+8+4+2 = 638 = 22\cdot 29$$

To go faster, use following idea:

$$a = 11010110 = 11010\cdot 2^5 + 10110 = a_L\cdot 2^5 + a_R$$

$$b = 11110010 = 11110\cdot 2^5 + 00010 = b_L\cdot 2^5 + b_R$$

$$a*b = (a_L\cdot 2^5 + a_R)(b_L\cdot 2^5 + b_R) = a_Lb_L\cdot 2^{10} + (a_Lb_R + a_Lb_R)\cdot 2^5 + a_Rb_R$$ — 4 mults

$$a_Lb_R + a_Lb_R = (a_L+a_R)(b_L+b_R) - a_Lb_L - a_Rb_R$$

$x = a_Lb_L \quad y = (a_L+a_R)(b_L+b_R) - z - x \quad z = a_Rb_R$

$a*b = x\cdot 2^{10} + y\cdot 2^5 + z$] 3 mults.

Karatsuba Multiplication Algorithm: [Karatsuba 1960]

Given two binary numbers $a = a_n \cdots a_1 a_0$, $b = b_n \cdots b_1 b_0$:

O(n) 1. Let $a_L = a_n \cdots a_{n/2}$, $a_R = a_{n/2-1} \cdots a_0$.
2. Let $b_L = b_n \cdots b_{n/2}$, $b_R = b_{n/2-1} \cdots b_0$.

3T(n/2) 3. Compute $x = a_L b_L$, $z = a_R b_R$, $y = (a_L + a_R)(b_L + b_R)$ using Karatsuba's.

O(n) 4. Return $x \ll n + y \ll \frac{n}{2} + z$.

Running time: $T(n) = 3T(\frac{n}{2}) + O(n) = O(n^{\log_2 3})$.

Compare to: $T(n) = 4T(\frac{n}{2}) + O(n) = O(n^2)$.

Generalized Karatsuba (Toom-Cook): [Toom 1963], [Cook 1966]

1. Break a, b into k pieces each.
2. Compute pairwise products of k pieces.
3. Do 2k-1 multiplications and some additions to compute a*b.

Running time: $O(F(k) \cdot n^{\log_k(2k-1)})$. ($F(k)$ is addition time)

Karatsuba: $k = 2$. Why not let $k \to n$? $F(k)$ grows fast.

Can do even better: $O(n \log n \log\log n)$ [Schönhage, Strassen 1971] hard

$O(n \log n \cdot 2^{O(\log^* n)})$ [Fürer 2007] not implemented

Conjecture: any algorithm takes $\Omega(n \log n)$ time. [SS 1971]

Given two n-bit numbers:
- Multiplication takes $\tilde{\Theta}(n \log n)$
- Addition takes $\Theta(n)$
- Equality, $>$, $<$ take $\Theta(n)$

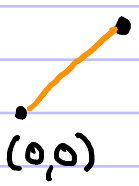
All basic operations take about linear time in # bits.
How many bits are allowed in an operand? Many options.

- Arbitrary (pay $\Theta(1)$) real RAM [Blum, Shub, Smale 1989], [Shamos 1978]
- Arbitrary (pay $\Theta(\#\text{bits})$) bit-level RAM van Emde Boas, Willard
- $\log(\#\text{input bits})$ (pay $\Theta(1)$) word RAM

As expected, real RAM is simple but not realistic:
If $+, -, *, /, \lfloor \rfloor$ are allowed operations, can solve QBF in polynomial time. [Schönhage 1979]] As hard as any problem using polynomial space. Up to $2^{2^{n^c}}$ time.

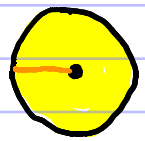
Also true if only $+, -, *$ and $\wedge, \vee, \neg$ are allowed. [Hartmanis, Simon 1974]

So why use real RAM model ever? Geometric problems.

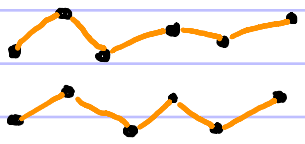


Compute distance between two points?

Compute area of a circle?



Decide which of two paths is shorter?



Sum-of-squares problem: given two sequences of integers $a_1, a_2, \ldots, a_n$ & $b_1, b_2, \ldots, b_n$, is $\sum_{i=1}^{n} \sqrt{a_i} < \sum_{i=1}^{n} \sqrt{b_i}$?
Solvable in $O(n)$ time in real RAM (used in shortest path w/obstacles)
Not known to be in NP in word RAM or bit-level RAM.

Common in computational geometry to use real RAM [Shamos 1978] but with the caveat that there is some imprecision, and with unspoken agreement not to cheat & do PSPACE-hard stuff.
realistic by self-regulation

Bit-level RAM: reasonable but annoying. As input gets bigger, indexing into arrays changes, etc.:

```
for (i=0; i<n; ++i)
{
    A[i] = A[i]+1;
}
```

i grows in size. total work of all "++i":
$\sum_{i=1}^{n} \log(i) = \Theta(n \log n)$
bleh
for-loops no longer run in linear time.

Word RAM: Constant-time to do operations on a "word".
A word has size $w \geq \log n$ for input of size n.
Also assume input numbers each fit into a word. $(0, 1, \ldots, 2^w - 1)$

Word RAM tends to allow faster algorithms and data structures than real RAM because of bounded universe & word-level parallelism.

Sorting: $O(n + 2^w)$ using counting sort

$O(n \cdot \frac{w}{\log n})$ using radix sort ($\frac{w}{\log n}$ iterations of counting sort)

$O(n \cdot \log w)$ using dynamic tree with $O(\log w)$ insert/delete
Van Emde Boas

Counting sort: $O(n)$ for $w = \log n$
Also $O(n)$ for all $w = \Omega(\log^{2+\varepsilon} n)$ possible
Radix sort: $O(n)$ for $w = O(\lg n)$
van Emde Boas Tree Sort: $O(n \lg \lg n)$ for $w = \log^c n$.

Word RAM and other models with words

Cell probe — More powerful

Transdichotomous RAM

Word RAM — Less powerful

All three models can do the same computations (all) but have different costs associated/different sets of $O(1)$-cost operations.

Cell probe: Pay $O(1)$ for read/write of a word.
All other computation is free.

Transdichotomous RAM: Pay $O(1)$ for read/write/(any operation (finite set) on a word.

Word RAM: Pay $O(1)$ for read/write/(simple operation (probably not *) on a word.

Lower bound in cell probe $\Rightarrow$ same lower bound in 2 other models.
E.g. finding max element in array takes $\Omega(n)$.

External Memory Model [Aggarwal, Vitter 1988]

Two levels of memory: fast and bounded cache
slow and unbounded disk

Cache consists of blocks of B words
with total space $M >> B$.

Can only compute on cache memory, running time is # blocks written & read. Computation is free!

Like cell probe, but with a "pile of words".

Can find max/min in $O(\lceil \frac{n}{B} \rceil)$ time.

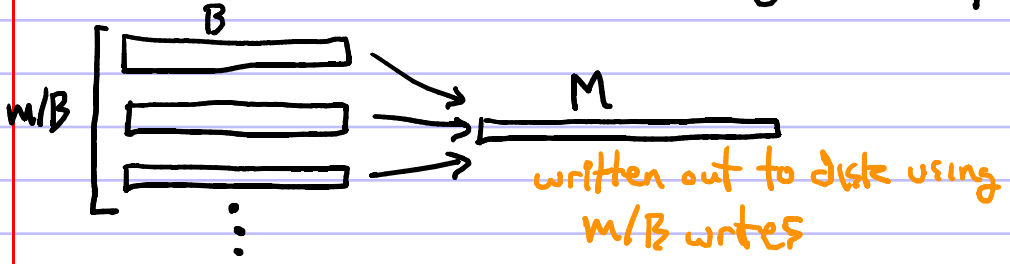
Can build dynamic BST with insert()/delete()/find() in $O(\log_{BH} n)$.

Lower bound of $\Omega(\log_{BH} n)$ for search():

1. Insertion location requires $\log(n+1)$ bits to specify, info content
2. Each block read gives at most $\log(B+1)$ bits. Where query fits
3. So at least $\frac{\log(n+1)}{\log(B+1)} = \Omega(\log_{BH} n)$ reads needed.

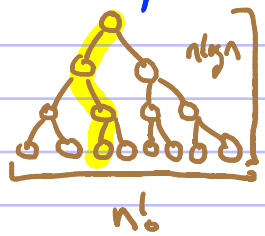
Can sort in $O(\frac{n}{B} \log(\frac{n}{B}))$ using MergeSort()
with a base case of size B.

Can sort in $O(\frac{n}{B} \log_{m/B} \frac{n}{B})$ using $\frac{M}{B}$-way MergeSort()



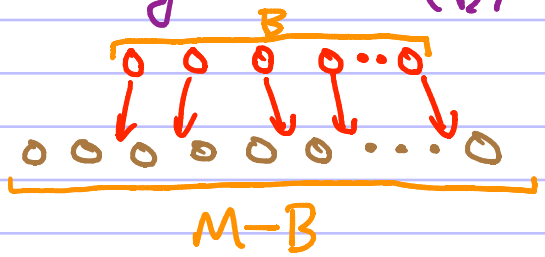
A lower bound of $\Omega(\frac{n}{B}\log_{M/B}(\frac{n}{B}))$ for sorting:

Recall that lower bound of $\Omega(n\log n)$ for regular sorting says $n!$ permutations, each operation reduces these by a factor of 2, so $\geq \log_2(n!) = \Omega(n\log n)$ operations for some permutation



Use the same idea w/ special considerations.

1. Assume each block is sorted, $O(n)$ time & reduces permutations by a factor of $(B!)^{n/B}$
2. Each read of a block lets us reduce the #permutations by a factor of $\binom{M}{B}$, since new block can only be permuted with existing blocks in $\binom{M}{B}$ ways:



$\binom{M}{B}$ ways to insert B things between M-B things

So time is $\geq \log_{\binom{M}{B}}\left(\frac{n!}{(B!)^{n/B}}\right) \geq \log_{(M^B/B!)}\left(\frac{n!}{(B!)^{n/B}}\right)$

$$\geq \frac{\log(n!) - \log((B!)^{n/B})}{\log(M^B) - \log(B!)} = \Omega\left(\frac{n\log n - \cancel{n}\log B}{B\log(M) - B\log B}\right)$$

$$= \Omega\left(\frac{n\log(\frac{n}{B})}{B\log(\frac{M}{B})}\right) = \Omega\left(\frac{n}{B}\log_{(\frac{M}{B})}(\frac{n}{B})\right)$$ tight

Cache-oblivious Model [Prokop 1999]

External memory model except M, B are not known by algorithm.

Realistic: M, B are low-level & separation of concerns may hide them.

M, B may be dynamic, code may run in VM moving around metal. Many M_i, B_is.

Analysis goal: optimality (e.g. $O(\frac{N}{B} \log_{M/B}(\frac{N}{B}))$ sorting) for all M, B.

How does block read/write happen? Automatically using an optimal replacement strategy.

Increasing M by $O(1)$ factor makes strategy optimal.

Sorting still possible in $O(\frac{N}{B} \log_{M/B}(\frac{N}{B}))$ [Frigo et. al. 1999] assuming $M = \Omega(B^{1+\varepsilon})$ for some ε.