Lecture 18: Friday October 19, 2001

today

- sorting lower bound (finish)   [CLRS Ch. 8.1]
- dynamic programming   [CLRS Ch. 15.1-3]

soon

- union find   [CLRS Ch. 21]
- graph algorithms   [CLRS Ch. 22,23]

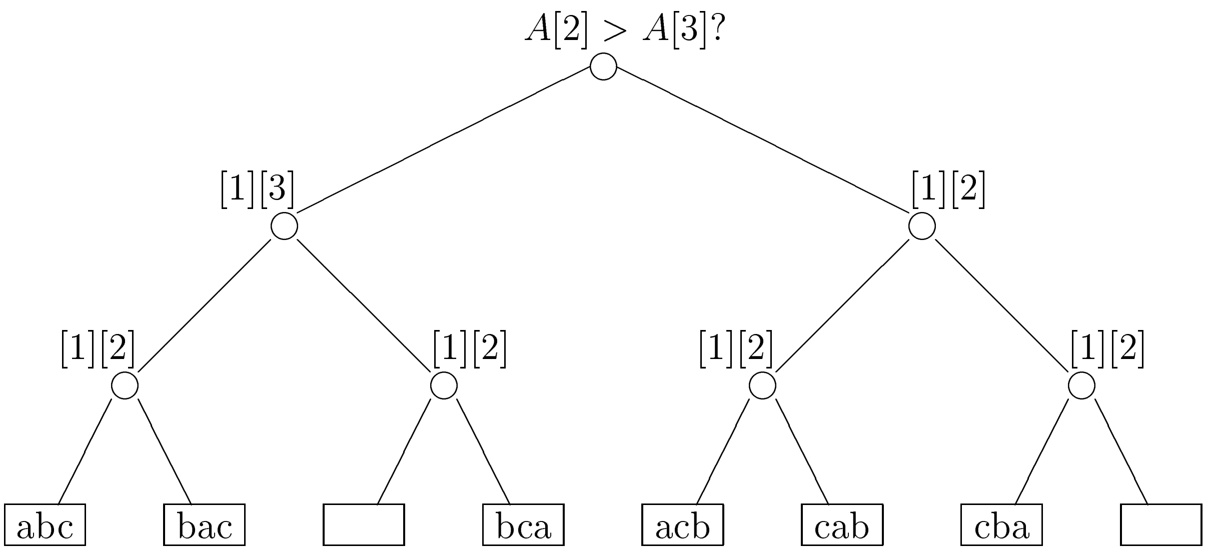
announcements

- assignment 5: due Monday October 29
- acknowledge references

recall: decision trees (e.g. selection sort)

- assume input keys in array $A[1\ldots3]$, initially `[a b c]`
- tree node: `A[k]>A[j]?` key comparision
- node label: $A[j]$

```
for j <- n downto 2 do
  psn <- j    (*index of max*)
  for k <- j-1 downto 1 do
    if A[k] > A[psn] then psn <- k
  exchange A[j] <-> [psn]
```



- notice: every case, 3 KC

sorting lower bound

- comparison based sort: keys can be compared **only**
- this argument considers **only** comparison based sort algorithms
- any binary tree with t leaves and k levels:
  - $t \leq 2^{k-1}$
  - $\lg t \leq k - 1$
  - $1 + \lg t \leq k$, namely
    binary tree with t leaves has $\geq 1 + \lg t$ levels
- the decision tree of any comparison based sort ...
  - binary
  - has $n!$ leaves
  - ...so has at least $1 + \lg(n!) = 1 + \sum_{j=1}^{n} \lg j \in \Theta(n \lg n)$ levels
- conclusion: any comparison based sort has
  - worst case number of key comparisons in $\Omega(n \lg n)$
  - worst case run time in $\Omega(n \lg n)$ ☺
- comparison based: selectsort, insertsort, heapsort, quicksort
- **not** comparison based: radix sort, bucket sort
- see Ch. 8 for extra reading

dynamic programming

- an algorithm design technique
- DP: avoiding recomputation of repeated subproblems by storing subproblem answers in tables/arrays

1st example problem: Fibonacci numbers

- $f(n) \;=\; \begin{cases} n & \text{if } n = 0, 1 \\ f(n-1) + f(n-2) & \text{if } n \geq 2 \end{cases}$

- 1st Fibonacci implementation: recursion

```
proc f(n)                                         -- f5 --
  if n<2 then return n                          /        \
  else return f(n-1)+f(n-2) endif  f4             f3
endf                             /  \           /  \
                               f3   f2      f2  f1
                              / \   / \    / \
                            f2 f1 f1 f0 f1 f0
                           / \
                         f1 f0
```

- ☹ repeated function calls
- time $\qquad T(n) \;=\; \begin{cases} c_1 & \text{if } n = 0, 1 \\ c_2 + T(n-1) + T(n-2) & \text{if } n \geq 2 \end{cases}$
- $T(n) > f(n)$ so $\qquad T(n) \in \Omega\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right)$ ☹

memoization: a DP improvement on recursion

- idea: keep recursion, avoid recomputation ...
  - store f( ) values in array F[ ]
  - if F[j] not yet initialized, compute it
  - if F[j] is initialized, access it
  - F[j] computed **only once**
- memoization: recursion with DP

- 2nd implementation: memoization (recursion with DP)

```
proc dpfib(n)                                     f5
  proc dpf(n)                                    /  \
    if F[n]=? then                             f4   f3
      F[n]<-dpf(n-1)+dpf(n-2) endif           /  \
    return F[n]                             f3   f2
                                           /  \
                                         f2   f1
  for j<-1 to n do F[j]<-? endfor       /  \
  F[0]<-0; F[1]<-1                    f1   f0
  dpf(n)
```

- ☺ fewer repeated function calls
- for each k, dpf(k) called $\leq$ twice ($F[k]$ known after 1st call)
- time $T_n \in \Theta(n)$ ☺

- 3rd implementation: full dynamic programming (with iteration)

```
proc dpf(n)
  F[0]<-0; F[1]<-1
  for j<-2 to n do F[j]<-F[j-1]+F[j-2] endfor
  return F[n]
enddpf
```

- time $T_n \in \Theta(n)$ ☺

multiple matrix multiplication

- input: matrices $A_1, \ldots, A_n$ with dimensions $d_0 \times d_1$, $\ldots d_{n-1} \times d_n$
- ouput: order in which matrices should be multiplied so that $A_1 \times A_2 \times \ldots \times A_n$ is computed using minimum number of scalar multiplications

```
A_1  A_2     A_3   A_4        d_0  d_1  d_2  d_3  d_4
 XX  XXXXXX  XXXX  XXX         5    2    6    4    3
 XX  XXXXXX  XXXX  XXX
 XX          XXXX  XXX
 XX          XXXX  XXX
 XX          XXXX
             XXXX
```

matrix mult'n order	scalar multiplications
$((A_1A_2)A_3)A_4$	$5 \times 2 \times 6 + 5 \times 6 \times 4 + 5 \times 4 \times 3 = 240$
$(A_1(A_2A_3))A_4$	$5 \times 2 \times 4 + 2 \times 6 \times 4 + 5 \times 4 \times 3 = 148$
$(A_1A_2)(A_3A_4)$	$5 \times 2 \times 6 + 5 \times 6 \times 3 + 6 \times 4 \times 3 = 222$
$A_1((A_2A_3)A_4)$	$5 \times 2 \times 3 + 2 \times 6 \times 4 + 2 \times 4 \times 3 = 102$
$A_1(A_2(A_3A_4))$	$5 \times 2 \times 3 + 2 \times 6 \times 3 + 6 \times 4 \times 3 = 138$