6-3

David Bozarth
CES 512
Spring 2005

(a) Draw a 4×4 Young Tableau containing the elements $\{9, 16, 3, 2, 4, 8, 5, 14, 12\}$.

2	4	9	∞
3	5	12	∞
8	14	16	∞
∞	∞	∞	∞

(b) Lemma discovered by David B.

Suppose $A_{m \times n}$ is a Young Tableau.

If $\begin{pmatrix} 1 \le j \le m \\ 1 \le k \le m \end{pmatrix}$ and $\begin{pmatrix} 1 \le j_0 \le m \\ 1 \le k_0 \le n \end{pmatrix}$, $j \in \mathbb{Z}^+$, $k \in \mathbb{Z}^+$

then the following two statements are equivalent:

(i) $A[j_0, k_0] = \infty$

(ii) If $A[j,k]$ is finite, then $\begin{pmatrix} j < j_0 \\ k < k_0 \end{pmatrix}$.

Proof of Lemma:

Show that (i) $\Rightarrow$ (ii):

($\Rightarrow$) Assume (i) is true. Let $A[j,k]$ be finite.

Let k_1 be the least index of infinite elements in row j.

Clearly $k_1 \le k_0$.

Since row j is sorted, then $k < k_1$.

Thus $k < k_0$.

Similarly we can show that $A[j,k]$ must have $j < j_0$.

Therefore (i) $\Rightarrow$ (ii).

cont'd

6-3.b cont'd

($\Leftarrow$) Show that (ii) $\Rightarrow$ (i).

Suppose (ii) is true. Let j_1 be the greatest index of finite elements in column k.

Then $A[j_1, k]$ is finite, and $j_1 < j_0$.

Because column k is sorted, then $A[j_0, k]$ is not finite.

Similarly we can show that if k_1 is the greatest index of finite elements in row j, then $A[j, k_0]$ is not finite.

Since $A[j_0, k] = \infty$ for any column k, and $A[j, k_0] = \infty$ for any row j,

then $A[j_0, k_0] = \infty$. Thus (ii) $\Rightarrow$ (i).

The Lemma has been proven. #

6-3.b cont'd

Let $A_{m \times n}$ be a Young Tableau.

Show: If $A[1,1] = \infty$ then A is empty.

Proof: Let $A[1,1] = \infty$.

There is no $j \in \{1, 2, \cdots, m)$ for which $j < 1$.

There is no $k \in \{1, 2, \cdots, n)$ for which $k < 1$.

By the Lemma, there is no finite element of A.

Therefore A is empty, by definition.

#

Let $A_{m \times n}$ be a Young Tableau.

Show: If $A[m,n]$ is finite, then A is full.

Proof: Let $A[m,n]$ be finite.

There is no $j_0 \in \{1, 2, \cdots, m\}$ for which $m < j_0$.

There is no $k_0 \in \{1, 2, \cdots, n\}$ for which $n < k_0$.

By the Lemma, there is no $A[j_0, k_0] = \infty$.

Therefore A is full, by definition.

#

6-3c (i)

Give an algorithm to implement EXTRACT_MIN on a nonempty $m \times n$ Young Tableau, that runs in $\Theta(m+n)$ time.

```
EXTRACT_MIN(m, n, A) returns a
:   a = A[1, 1]

    A[1, 1] = A[m, n]
    A[m, n] = ∞

    REJUVENATE_DOWN(1, 1, m, n, A)
:   RETURN a
```

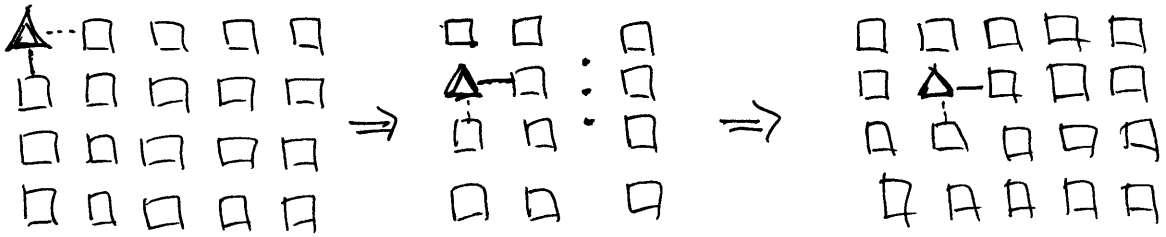
```
REJUVENATE_DOWN(j, k, m, n, A) returns void
:   if j > m OR k > n then RETURN
    smallV = j ; smallH = k

    if j < m then
        if A[j, k] > A[j+1, k] then smallV = j+1

    if k < n then
        if A[smallV, k] > A[j, k+1] then
            smallV = j
            smallH = k+1

    if smallV ≠ j OR smallH ≠ k then
        SWAP(j, k, smallV, smallH, A)
        REJUVENATE_DOWN(smallV, smallH, m, n, A)

:   RETURN
```

Pre: A is an $m \times n$ Young Tableau, {except possibly that $A[j,k] \geq A[j+1,k]$ OR $A[j,k] \geq A[j,k+1]$}

Post: A is an $m \times n$ Young Tableau.

6.3.c (i) cont'd

EXTRACT_MIN runs in time that is a constant plus the time to recurse REJUVENATE_DOWN over the Young Tableau.



etc.

The number of sets of operations - each running in constant time - is bounded by $m+n$.

So the algorithm runs in time $\Theta(m+n)$.

(ii)

Define $T(p)$, where $p = m+n$, to be the maximum running time of EXTRACT_MIN on any $m \times n$ Young Tableau. Give and solve a recurrence for $T(p)$ that yields the $\Theta(m+n)$ time bound.

The recurrence : $T(p) = k + T(p-1)$

Derive $\Theta(m+n)$ $\Rightarrow$ see next page

6-3c (ii)

$T(p)$ ≡ max running time of EXTRACT-MIN on $A_{m \times n}$ if A is Young tableau. $\{p = m+n\}$

D. Bozarth

· Derive $\Theta(m+n)$.

$$T(p) \leftarrow \underbrace{(1 \text{ read}) + (1 \text{ move}) + (\text{max 2 comparisons})}_{} + (1 \text{ swap}) + T(p-1)$$

$$T(p) = \Theta(1) + T(p-1)$$

$$T(p) = k + [k + T(p-2)] \quad \text{where } k = \Theta(1)$$

$$T(p) = k + [k + \{k + T(p-3)\}]$$

$$T(p) = 3k + T(p-3)$$

$$\vdots$$

$$T(p) = ik + T(p-i)$$

$$\vdots$$

$$T(p) = (p-1)k + T(1)$$

$$T(p) = (m+n-1)k + T(1)$$

$$T(p) = \Theta(m+n) \quad \#$$

6.3.4

D. Bozarth

Insert into a nonfull, m x n Young Tableau in $\Theta(m+n)$ time.

```
INSERT ( a, m, n, A) returns void
:   A[m, n] = a

    REJUVENATE_UP ( m, n, m, n, A)
    RETURN
:
```

```
REJUVENATE_UP ( j, k, m, n, A)  returns void
// Pre: A is a Young Tableau, m x n, except possibly
       that  A[j,k] ≤ A[j-1, k]
          OR  A[j,k] ≤ A[j, k-1]

// Post: A is a Young Tableau, m x n.
:  if j<1 OR k<1 then RETURN
   largeV = j ; largeH = k

   if j > 1 then
      if A[j,k] < A[j-1,k] then largeV = j-1

   if k > 1 then
      if A[largeV, k] < A[j, k-1] then
         largeV = j ; largeH = k-1

   if largeV ≠ j OR largeH ≠ k then
      SWAP( j, k, largeV, largeH, A)
      REJUVENATE_UP (largeV, largeH, m, n, A)
:  RETURN
```

6-3.d cont'd

INSERT runs in $\Theta(m+n)$ time.
It is essentially the backward traversal of the Young Tableau as would be performed by EXTRACT_MIN, (which was previously shown to run in $\Theta(m+n)$ time).

6-3.e Using no other sorting method as a subroutine, show how to sort n^2 numbers in time $\Theta(n^3)$ using a Young Tableau.

I will show two methods. The first assumes an input stream from an unspecified container, and an existing Young Tableau, empty, with dimensions $n \times n$.

The second method takes an $n \times n$ array of unsorted numbers, transforms it into an $n \times n$ Young Tableau.

Both methods produce a sorted array of length n^2.

6-3.e cont'd

First method: D. Botarth

Sort n^2 numbers from an unspecified container, using a Young Tableau that has dimensions $n \times n$, and is empty.

```
SORT_YOUNG1 ( output [], n, A) returns void
// Pre : A is nxn Young Tableau  // Post: n^2 numbers are in
:                                   a sorted array named
                                    output[n^2].
  for i = 1 to n^2
     a = Get A Number From Container

     INSERT(a, n, n, A)
  next i

  for i = 1 to n^2
     output[i] = EXTRACT_MIN(n, n, A)

  next i
:  RETURN
```

Inserting n^2 numbers: The running time of the ith iteration is $\Theta(2\sqrt{i})$.

The first for-loop runs in time $2\sum_{i=1}^{n^2} \sqrt{i}$.

Extracting: (So does the second for-loop.)

The total running time is

$$4\sum_{i=1}^{n^2} i^{1/2} \leq 4\int_1^{n^2+1} x^{1/2}\,dx = 2x^{3/2}\Big]_1^{n^2+1} = 2\left[(n^2+1)^{3/2} - 1\right]$$

$$(\text{running time}) = \Theta(n^3) \quad \#$$

6-3.e cont'd

Second method : transform in place

SORT_YOUNG2 (output[], n, A) returns void

: JUVENATE (n, A)

for i = 1 to n^2

output[i] = EXTRACT_MIN (n, n, A)

next i

: RETURN

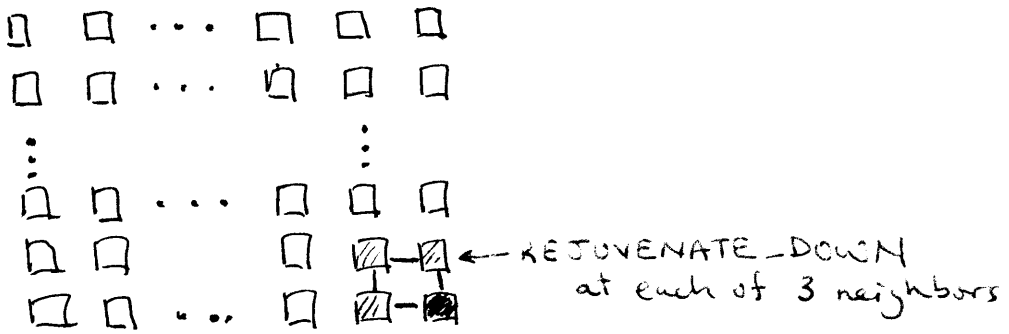
JUVENATE (n, A) returns void

pre : A is an nxn matrix.

post : A is an nxn Young Tableau.

```
: j = n
  while j > 1
      k = j
      while k > 1
          REJUVENATE_DOWN (k-1, j, m, n, A)
          k = k-1
      k = j
      while k > 1
          REJUVENATE_DOWN (j, k-1, m, n, A)
          k = k-1
      REJUVENATE_DOWN (j-1, j-1, m, n, A)
      j = j-1
: RETURN
```

JUVENATE starts at the "high end" of the $n \times n$ array, $A[n,n]$, working leftward and upward to create an enlarging Young Tableau.



For each position j of the outer loop, there will be j executions of each inner loop.
Each such execution has time bounded from above by $(j+1)+j \leq 2n+1$.
Therefore each outer loop iteration will see each inner loop executing $j(2j+1) \leq n(2n+1)$ times.
Also, each outer loop iteration sees $2j \leq 2n$ for the single call to REJUVENATE_DOWN.

So the time for n executions of the outer loop is $\leq n(2n(2n+1)+2n) = \Theta(n^3)$

As seen earlier, removing a sorted stream also takes $\Theta(n^3)$. #

6-3.f

D. Bozarth

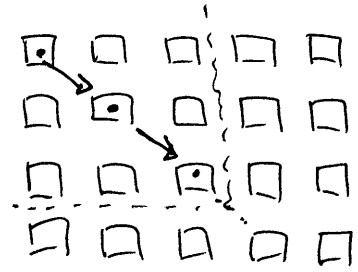
Give an $\Theta(m+n)$-time algorithm to determine whether a given number is stored in a given $m \times n$ Young Tableau.

{Need: analog of Binary Search.}

FIND (target, j, k, m, n, A) returns flag

// pre: A is an $m \times n$ Young Tableau. (j,k) is the coordinate of the "upper-left" corner of the Young Tableau subarray to be searched.

// post: If target is contained in the Young Tableau, flag = 1
otherwise, flag = 0.



: flag ← 0

① case A[j,k] = target
flag ← 1

② case j = m AND k < n
if A[j, k+1] ≤ target then
{return FIND(target, j, k+1, m, n, A)}

③ case j < m AND k = n
if A[j+1, k] ≤ target then
{return FIND(target, j+1, k, m, n, A)}

or can check for equality before call

④ case A[j+1, k+1] < target
{return FIND(target, j+1, k+1, m, n, A)}

⑤ case A[j+1, k+1] > target
{jflg = FIND(target, j+1, 1, m, , A) ← should check for 1 here
kflg = FIND(target, 1, k+1, m, n, A)
return (jflg OR kflg)}
: RETURN

(continued)

6-3.f cont'd

On entering not including A[j,k]
~~Prior to~~ the call to FIND, a subarray of size (j , k) has been eliminated from the search for target, and the element A[j, k] is compared with target.

In case (1), the search is complete.

In case (2), (m-1) rows have been eliminated and there are only (n-k) elements to search. Afterward, there will be at most (n-k-1) remaining.

In case (3) there remain only (m-j) elements to search. Afterward, there will remain at most (m-j-1).

In case (4), a subarray of size (j+1, k+1) will ~~be~~ have been eliminated... a net "gain" of one row and one column.

In case (5), a subarray of size (m-j, n-k) will be eliminated, in addition to the (j,k) size subarray that was previously eliminated. Note that $\begin{Bmatrix} m-j > 0 \\ n-k > 0 \end{Bmatrix}$.

Thus in each case except (1), the current call to FIND traverses the "pointer" by either one row or one column, or both.

Therefore, the maximum number of calls to FIND, each with constant time, is $m+n \Rightarrow \Theta(m+n)$ #