ESO207A/211: Data Structures and Algorithms

Homework *2* Solutions **Due: September 3, 2014**

Problem 1. Unimodal Search (Divide-and-Conquer). An array $A[1 \dots n]$ of numbers is unimodal if there exists some index $k \in \{1, 2, \dots, n\}$ such that $A[1] \leq A[2] \leq \dots \leq A[k]$ and $A[k] \geq A[k+1] \geq \dots A[n]$. The element $A[k]$ is the maximum element of the array and it is the unique "locally maximum" element surrounded by elements $A[k-1]$ and $A[k+1]$, that are both not larger than itself. An example unimodal array: $A = \boxed{1\ 4\ 5\ 11\ 8\ 7\ 1}$. Here, $A[4] = 11$ is the maximum element.

Give an algorithm that given a unimodal array $A[1 \dots n]$, finds the maximum element in $O(\log(n))$ time. (a) Prove the correctness of your algorithm, and (b) prove the bound on its running time.

Solution 1. PseudoCode:

```
if $end = start$ then
    return $arr[start]$;
else
    if $end - start = 1$ then
        if $arr[start] < arr[end]$ then
            return $arr[end]$;
        else
            return $arr[start]$;
        end
    else
        int mid= (start+end)/2;
        if $arr[mid] > arr[mid+1]$ then
            if $arr[mid] > arr[mid-1]$
            then
                return $arr[mid]$;
            else
                return
                $getMaxElement(arr, start, mid-$
                $1)$;
            end
        else
            return
            $getMaxElement(arr, mid+$
            $1, end)$;
        end
    end
end
```

Algorithm 1: INT GETMAXELEMENT(INT *ARR,INT START,INT END)

a. Proof of Correctness: For n = 1 and n = 2, the function trivially returns the correct answer. Now, we take the stronger induction assumption that for n ≤ k(k ≥ 2), the algorithm returns the correct answer. We now show that the algorithm returns the correct answer for n =

k+1. Note that, since k $\geq$ 2, (k+1) $\geq$ 3. Therefore, in the first function call, the else part of the algorithm would be executed. We now show that the next recursive call(if there is one), will include the maximum element as part of the sub-array passed to it. If the mid element is greater than the (mid-1)$^{\text{th}}$ element and the (mid+1)$^{\text{th}}$ element, it will be the maximum element (by the problem statement). If the mid element is greater than its successor but less than its predecessor, by the problem statement, the maximum element will be before mid, i.e. from start to mid-1. However, the next recursive call is for the array from start to mid-1. Therefore, it will contain the maximum element. Also, note that this call has array size $\leq$k(as it doesn't include mid and end). Therefore, by our assumption, this call will return the correct answer. If the mid element is less than its successor, by the problem statement, it is one of the elements from $arr[mid + 1]$ to $arr[end]$. In the next function call, we call the function with start and end as mid+1 and end respectively. Clearly, this doesn't include start as well as the mid element, and hence, is smaller than the original array by atleast one element. Clearly, this function call has size $\leq$ k and therefore, by our assumption, it is correct. Hence, proved.

b. Time bound proof: Let the time taken by the algorithm for an array of size n be T(n). $n = (end + 1 - start)$ If $arr[mid - 1] < arr[mid] < arr[mid + 1]$, then the call will be from (mid+1) to end. Therefore, new size of array $n' = (end - mid) \leq \frac{n}{2}$ If $arr[mid - 1] > arr[mid] > arr[mid + 1]$, then the call will be from start to mid-1. Therefore, the new size of array $n' = (mid - start) \leq \frac{n}{2}$

Note that the maximum time to either a direct return(without a recursive call) or a recursive call is independent of n.

Clearly, T(1) $\leq$ c, T(0) $\leq$ c.

$T(n) \leq T(\frac{n}{2})+c$
$\leq T(\frac{n}{4})+c+c$
...
$\leq T(\frac{n}{2^k})+k \times c$

Taking k=$\log n$, we have T(n) $\leq$ T(1) + c$\times \log n$=O($\log n$)

Problem 2. Young Tableau (Heaps). An $m \times n$ tableau is an $m \times n$ matrix such that the entries of each row are in sorted order from left to right and the entries of each column are in sorted order from top to bottom. Some of the entries of a Young tableau may be ∞ which are treated as non-existent elements.

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

b. Show that an $m \times n$ Young tableau is empty if $Y[1, 1] = \infty$. Argue that Y is full (contains mn elements) if $Y[m, n] < \infty$.

c. Give an algorithm to implement EXTRACT-MIN on a non-empty $m \times n$ Young tableau that runs in $O(m + n)$ time. Your algorithm should use a recursive subroutine that solves an $m \times n$ problem by recursively solving either an $(m - 1) \times n$ or an $m \times (n - 1)$ problem.

d. Show how to insert a new element into a non-full $m \times n$ Young tableau in $O(m + n)$ time.

e. Give an $O(m+n)$ time algorithm to determine whether a given number is stored in a given $m \times n$ Young tableau.

Solution 2.

a. There are multiple correct answers. One is presented below.

2	3	4	5
8	9	12	14
16	∞	∞	∞
∞	∞	∞	∞

b. Consider any element $Y[i,j]$. For any such element, we can say the following.

$$\begin{aligned} Y[i,j] &\geq Y[i,1] && \text{column } i \text{ is sorted} \\ Y[i,j] &\geq Y[i,1] \geq Y[1,1] && \text{row 1 is sorted} \\ \textit{Similarly, } Y[i,j] &\leq Y[i,n] \leq Y[m,n] && \text{by the mercy of the Lord} \end{aligned}$$

Now, if $Y[1,1] = \infty$, it follows that $\forall(i \leq m, j \leq n)$, $Y[i,j] = \infty$ since $Y[i,j] \geq Y[1,1]$. Hence, the Young tableau Y is empty.
Arguing on similar lines, if $Y[m,n] < \infty$, it follows that $\forall(i \leq m, j \leq n)$, $Y[i,j] < \infty$ since $Y[i,j] \leq Y[m,n]$. Hence, the Young tableau Y is full (that is, $\nexists(i \leq m, j \leq n)$ such that $Y[i,j] = \infty$).

c. An algorithm Extract-Min is described next, which acconplishes the purpose.

```
Data: Young tableau
let $minimumValue \leftarrow Y[1,1]$;
Set $Y[1,1] \leftarrow \infty$;
Correct-Tableau(Y, 1, 1);
Return $minimumValue$;
  Algorithm 2: Extract-Min(Y)
```

```
Data: Young tableau Y, i, j
if $i < m$ and $j = n$ then
    swap $Y[i,j] \leftrightharpoons Y[i+1,j]$;
    Correct-Tableau(Y,i+1,j);
if $i = m$ and $j < n$ then
    swap $Y[i,j] \leftrightharpoons Y[i,j+1]$;
    Correct-Tableau(Y,i,j+1);
if $i < m$ and $j < n$ then
    if $Y[i+1,j] < Y[i,j+1]$ then
        swap $Y[i,j] \leftrightharpoons Y[i+1,j]$;
        Correct-Tableau(Y,i+1,j);
    else
        swap $Y[i,j] \leftrightharpoons Y[i,j+1]$;
        Correct-Tableau(Y,i,j+1);
    end
    Algorithm 3: Correct-Tableau(Y, i, j)
```

Proof of Correctness: As a part of the previous proof, we have managed to show that $Y[1,1]$ is a minimum among the set of all elements in Y. This ratifies our choice of return value. Setting $Y[1,1]$ to ∞ ensures the count of $minimumValue$ in Y decreases by one (preserving count of other finite elements). It remains to show that Correct-Tableau constructs a valid Young tableau from now inconsistent Y (inconsistent at $(i,j) = (1,1)$),

which would mean EXTRACT-MIN results in a meaningful operation.
We establish a few notations. Let R_{Y_i} be the tableau formed after deleting the i^{th} row, and C_{Y_i} after deleting the i^{th} column. With this in mind, we consider the following claim:

Theorem 1. Claim($v = i + j$): *Right before* CORRECT-TABLEAU(Y, i, j) *is invoked,* R_{Y_i} *and* C_{Y_j} *satisfy Young tableau conditions.*

Proof. We proceed by mathematical induction on the variable $v = i + j$, from 2 to $m + n$. Clearly, for $v = 2$, the claim holds since the inconsistency exists only at $(1, 1)$. Assuming, the result holds $\forall v \leq v^*$, we prove that the claim is true for $v = v^* + 1$. Without loss of generality, we assume i and j such $i + j = v^*$. Now, to reach the state $v = v^* + 1$, there are 2 possible means.

A $Y[i, j] \leftrightharpoons Y[i + 1, j]$: This happens when $Y[i + 1, j] < Y[i, j + 1]$ or $j = n$. In either case, all we need to prove is that row r_i gets sorted before the next function call to $(i + 1, j)$ takes place. Note $Y[i + 1, j]$ now becomes either the last element in the i^{th} row or $Y[i+1, j] < Y[i, j+1]$. Again, noting R_{Y_i} is a Young tableau, $Y[i+1, j] \geq Y[i+1, j-1]$. And, $Y[i + 1, j - 1] \geq Y[i, j - 1]$ due to C_{Y_i}. Therefore, $Y[i + 1, j] \geq Y[i, j - 1]$. Noting that exlcuing the j^{th} position, r_i is already sorted. However, after the swap, the complete row r_i is sorted, meaning $R_{Y_{i+1}}$ is sorted.

B $Y[i, j] \leftrightharpoons Y[i, j + 1]$:This happens when $Y[i+1, j] \geq Y[i, j+1]$ or $i = m$. In either case, all we need to prove is that column c_j gets sorted before the next function call to $(i, j + 1)$ takes place. Note $Y[i, j + 1]$ now becomes either the last element in the j^{th} column or $Y[i+1, j] \geq Y[i, j+1]$. Again, noting C_{Y_i} is a Young tableau, $Y[i, j+1] \geq Y[i-1, j+1]$. And, $Y[i - 1, j + 1] \geq Y[i - 1, j]$ due to R_{Y_i}. Therefore, $Y[i, j + 1] \geq Y[i - 1, j]$. Noting that exlcuing the i^{th} position, c_j is already sorted. However, after the swap, the complete column c_j is sorted, meaning $C_{Y_{j+1}}$ is sorted.

□

When the function call to (m, n) is made C_{Y_n} and R_{Y_m} are Young tableaues. Further, $Y[m, n] = \infty$. This means Y is a Young tableau once again.

Bound on running time: Since each function call to CORRECT-TABLEAU takes a constant amount of time before transfering control to some other invocation, the running time is proportional to number of function calls made. Now, consider $i + j$. At the end of every function call, the next function call starts with an increased value of i or j. So, number of function calls is bounded by $m + n$ (the difference between maximum and minimum values of $i + j$). Hence, the running time is bounded by $O(m + n)$.
Alternatively, one could argue along the following steps. Let $T(i, j)$ be the running time associated with CORRECT-TABLEAU(Y, i, j).

$$T(i, j) \leq max\{T(i + 1, j), T(i, j + 1)\} + c$$
$$T(n, m) \leq c$$

It suffices to verify that $T(i, j) \leq c(m + n + 2 - i - j)$. Hence, $T(1, 1) \leq c(n + m)$. The proof in this case, however, must proceed through mathematical induction.

d. The algorithm INSERT does the job.

```
Data: Young tableau Y
Set $Y[m,n] \leftarrow$val;
CORRECT-TABLEAU-AGAIN(Y, m, n);
    Algorithm 4: INSERT(Y, val)
```

```
Data: Young tableau Y, i, j
if $i > 1$ and $j = 1$ and $Y[i,j] < Y[i-1,j]$ then
    swap $Y[i,j] \leftrightharpoons Y[i-1,j]$;
    CORRECT-TABLEAU-AGAIN(Y,i-1,j);
if $i = 1$ and $j > 1$ and $Y[i,j] < Y[i,j-1]$ then
    swap $Y[i,j] \leftrightharpoons Y[i,j-1]$;
    CORRECT-TABLEAU-AGAIN(Y,i,j-1);
if $i > 1$ and $j > 1$ then
    if $Y[i-1,j] < Y[i,j-1]$ and $Y[i,j] < Y[i,j-1]$
    then
        swap $Y[i,j] \leftrightharpoons Y[i,j-1]$;
        CORRECT-TABLEAU-AGAIN(Y,i,j-1);
    if $Y[i,j-1] \leq Y[i-1,j]$ and $Y[i,j] < Y[i-1,j]$
    then
        swap $Y[i,j] \leftrightharpoons Y[i-1,j]$;
        CORRECT-TABLEAU-AGAIN(Y,i-1,j);
Algorithm 5: CORRECT-TABLEAU-AGAIN(Y, i, j)
```

Proof of Correctness: As a part of the previous proof, we have managed to show that $Y[m,n]$ is a maximum among the set of all elements in Y. So, $Y[m,n] = \infty$ before the algorithm operates. Setting $Y[m,n]$ to *val* ensures the count of other elements do not change, while count of *val* increases by one. It remains to show that CORRECT-TABLEAU-AGAIN constructs a valid Young tableau from now inconsistent Y (inconsistent at $(i,j) = (m,n)$), which would mean INSERT results in a meaningful operation.
We establish a few notations. Let R_{Y_i} be the tableau formed after deleting the i^{th} row, and C_{Y_i} after deleting the i^{th} column. With this in mind, we consider the following claim:

Theorem 2. Claim($v = m+n-\overline{i+j}$)**:** *Right before* CORRECT-TABLEAU(Y,i,j) *is invoked,* R_{Y_i} *and* C_{Y_j} *satisfy Young tableau conditions.*

Proof. We proceed by mathematical induction on the variable v, from 0 to $m+n-2$. Clearly, for $v = 0$, the claim holds since the inconsistency exists only at (m,n). Assuming, the result holds $\forall v \leq v^*$, we prove that the claim is true for $v = v^*+1$. Without loss of generality, we assume i and j such $m+n-\overline{i+j} = v^*$. Now, to reach the state $v = v^*+1$, there are 2 possible means.

A $Y[i,j] \leftrightharpoons Y[i-1,j]$: This happens when $Y[i-1,j] \geq Y[i,j-1]$ or $j = 1$. In either case, all we need to prove is that row r_i gets sorted before the next function call to $(i-1,j)$ takes place. Note $Y[i-1,j]$ now becomes either the first element in the i^{th} row or $Y[i-1,j] \geq Y[i,j-1]$. Again, noting R_{Y_i} is a Young tableau, $Y[i-1,j] \leq Y[i-1,j+1]$. And, $Y[i-1,j+1] \geq Y[i,j+1]$ due to C_{Y_i}. Therefore, $Y[i,j+1] \geq Y[i-1,j]$. Noting that exlcuing the j^{th} position, r_i is already sorted. However, after the swap, the complete row r_i is sorted, meaning $R_{Y_{i+1}}$ is sorted.

B $Y[i,j] \leftrightharpoons Y[i,j-1]$:This happens when $Y[i-1,j] < Y[i,j-1]$ or $i = 1$. In either case, all we need to prove is that column c_j gets sorted before the next function call to $(i,j-1)$ takes place. Note $Y[i,j-1]$ now becomes either the first element in the j^{th} column or $Y[i-1,j] < Y[i,j-1]$. Again, noting C_{Y_i} is a Young tableau, $Y[i,j-1] \geq Y[i+1,j-1]$.

And, $Y[i+1, j-1] \geq Y[i+1, j]$ due to R_{Y_i}. Therefore, $Y[i, j-1] \geq Y[i+1, j]$. Noting that exlcuing the i^{th} position, c_j is already sorted. However, after the swap, the complete column c_j is sorted, meaning $C_{Y_{j+1}}$ is sorted.

□

The algorithm terminates as soon as:

A $Y[i, j] \geq Y[i-1, j]$ and $j = 1$

B $Y[i, j] \geq Y[i, j-1]$ and $i = 1$

C $Y[i, j] \geq Y[i-1, j]$ and $Y[i, j] \geq Y[i, j-1]$

In all these cases, the appropirate i^{th} row and j^{th} column become sorted. C_{Y_j} and R_{Y_i} are already valid Young tableau. Hence, the entire Y becomes a valid Young tableau.

Bound on running time: Since each function call to CORRECT-TABLEAU-AGAIN takes a constant amount of time before transfering control to some other invocation, the running time is proportional to number of function calls made. Now, consider $i + j$. At the end of every function call, the next function call starts with an decreased value of i or j. So, number of function calls is bounded by $m + n$ (the difference between maximum and minimum values of $i + j$). Hence, the running time is bounded by $O(m + n)$.
Alternatively, one could argue along the following steps. Let $T(i, j)$ be the running time associated with CORRECT-TABLEAU(Y, i, j).

$$T(i, j) \leq max\{T(i-1, j), T(i, j-1)\} + c$$
$$T(1, 1) \leq c$$

It suffices to verify that $T(i, j) \leq c(i+j)$. Hence, $T(m, n) \leq c(n+m)$. The proof in this case, however, must proceed through mathematical induction.

e. The algorithm FIND achieves the objectives.

```
Data: Young tableau Y, val
Return FIND-RANGE(Y, val, 1, n);
    Algorithm 6: FIND(Y, val)
```

```
Data: Young tableau Y, val, i ,j
if $Y[i, j] = val$ then
    Return True;
if $Y[i, j] < val$ then
    if $i < m$ then
        Return FIND-RANGE(Y, val, i+1, j);
    else
        Return False;
    end
if $Y[i, j] > val$ then
    if $j > 1$ then
        Return FIND-RANGE(Y, val, i, j-1);
    else
        Return False;
    end
    Algorithm 7: FIND-RANGE(Y, val, i, j)
```

Proof of Correctness: It is immedaitely clear that the result returned is true only if the algorithm successfully finds a pair (i, j) such $Y[i, j] = val$. It remains to show if the algorithm returns false, then Y does not contain val. We use $\phi_{i,j}$ to deonte union of $Y(1 \ldots \overline{i-1}, 1 \ldots n)$ and $Y(1 \ldots m, \overline{j+1} \ldots n)$. We consider the following claim:

Theorem 3. Claim($v = n - \overline{j - i}$): *If* Correct-Tableau(Y, i, j) *invoked,* $\phi_{i,j}$ *does not contain val.*

Proof. We proceed by mathematical induction on the variable v, from 1 to $m+n-1$. Clearly, for $v = 1$, the claim holds since $\phi_{1,n}$ is empty. Assuming, the result holds $\forall v \leq v^*$, we prove that the claim is true for $v = v^* + 1$. Without loss of generality, we assume i and j such $n - \overline{j - i} = v^*$. Now, to reach the state $v = v^* + 1$, there are 2 possible means.

A Find-Range(Y,i+1,j) is called: This happens when $Y[i, j] < val$ and $i < m$. By induction hypothesis, val is not present in $\phi_{i,j}$. We need to establish that val is not present in $Y(i, 1 \ldots j)$ does not contain val. Since, $Y[i, j] < val$, $\forall j' \leq j$, we have $Y[i, j'] \leq Y[i, j] < val$. Hence, $Y(i, 1 \ldots j)$ does not contain val; and neither does $\phi_{i+1,j}$.

A Find-Range(Y,i,j-1) is called: This happens when $Y[i, j] > val$ and $j > 1$. By induction hypothesis, val is not present in $\phi_{i,j}$. We need to establish that val is not present in $Y(i \ldots m, j)$ does not contain val. Since, $Y[i, j] > val$, $\forall i' \geq i$, we have $Y[i', j] \geq Y[i, j] > val$. Hence, $Y(i \ldots m, j)$ does not contain val; and neither does $\phi_{i,j-1}$.

□

The algorithm terminates returns false when:

A $Y[m, j] < val$: Here $\phi_{m,j}$ does not contain val; neither does $Y(m, 1 \ldots j)$ since $val > Y[m, j]$. So Y does not contain val.

A $Y[i, 1] > val$: Here $\phi_{i,1}$ does not contain val; neither does $Y(i \ldots m, 1)$ since $val < Y[i, 1]$. So Y does not contain val.

Bound on running time: Since each function call to Find-Range takes a constant amount of time before transfering control to some other invocation, the running time is proportional to number of function calls made. Now, consider $i - j$. At the end of every function call, the next function call starts with an increased value of i or $(-j)$. So, number of function calls is bounded by $m + n$ (the difference between maximum, m, and minimum, $(-n)$, values of $i + j$). Hence, the running time is bounded by $O(m + n)$.
Alternatively, one could argue along the following steps. Let $T(i, j)$ be the running time associated with Correct-Tableau(Y, i, j).

$$\begin{aligned} T(i, j) &\leq max\{T(i+1, j), T(i, j-1)\} + c \\ T(m, 1) &\leq c \end{aligned}$$

It suffices to verify that $T(i, j) \leq c(m + 1 - \overline{i - j})$. Hence, $T(1, n) \leq c(n + m)$. The proof in this case, however, must proceed through mathematical induction.