CSCI 3104-Spring 2015: Assignment #2.

Assigned date: Monday 1/26/2015,
Due date: Tuesday 2/3/2015 before 3:30 PM
Maximum Points: 45 points + 5 for legibility

Note: This assignment *must be turned in on paper, before end of class.* Please do not email: it is very hard for us to keep track of email submissions. Further instructions are on the class page: `http://csci3104.cs.colorado.edu`

P1 (5 points) Prove using strong induction that the recurrence relation

$$T(n) = \begin{cases} T(n-1) + \cdots + T(1) + n & \text{if} \quad n > 1 \\ 1 & \text{if} \quad n = 1 \end{cases}$$

has the closed form solution $T(n) = 2^n - 1$ for all $n \geq 1$.

Solution. Proof is by *strong induction* on n.
Base Case: For $n = 1$, verify that $T(n) = 1 = 2^1 - 1$.
Induction Hypothesis: Assume that for all $1 \leq j \leq n$, we have $T(j) = 2^j - 1$. We wish to prove that $T(n+1) = 2^{n+1} - 1$.

Since $n \geq 1$, we have $n + 1 \geq 2$. Therefore, applying the recurrence relation,

$$\begin{aligned} T(n+1) &= \textstyle\sum_{j=1}^{n} T(j) + (n+1) && \text{(* simple application of the recurrence *)} \\ &= \textstyle\sum_{j=1}^{n} (2^j - 1) + n + 1 && \text{(* Applying induction hypothesis on j = 1 to n *)} \\ &= 2(2^n - 1) - n + n + 1 && \text{(* Summing up geometric series *)} \\ &= 2^{n+1} - 1 && \text{(* Simplification *)} \end{aligned}$$

Thus the inductive hypothesis stands proven and overall, the theorem is proven using induction.

▲

P2 (30 points) This assignment concerns the analysis of the so-called saddleback search algorithm.

A $n \times n$ matrix is *sorted* if each row is sorted left to right in ascending order, and each column is sorted from top to bottom in an ascending order. Here is an example of a sorted 6×6 matrix

2	4	5	7	11	15
3	6	7	11	17	21
4	7	8	17	19	22
5	8	9	18	20	25
11	15	19	21	26	31
13	17	22	28	32	33

Consider the problem of searching for whether or not an element k belongs to a sorted $n \times n$ matrix A.

Inputs: A sorted $n \times n$ matrix A and a number k.

Outputs: True if k is equal to an entry of the matrix A. False otherwise.

(a, 5 points) The saddleback search algorithm runs as follows:

```
def saddleBackSearch(A, n, k):
    # A is a n * n matrix (list of lists in python).
    i = n-1
    j = 0
    # start from the lower left corner of the matrix
    while (i >= 0 and j < n):
        if (A[i][j] == k):
                return True # Found it!
        elif (A[i][j] > k):
                i = i -1 # Move up
        else: # A[i][j] must be < k
                j = j + 1 # Move the right
    return False # Could not find it.
```

Using the example matrix provided above, show the working of the algorithm for (a) $k = 20$ and (b) $k = 10$. Specifically show the value of i, j and $A[i][j]$ at the beginning of each loop iteration and show how $A[i][j]$ compares with k. Use $0 \ldots, n-1$ as the valid range of indices for rows and columns. For each value of k fill up a table like this:

Loop Iteration #	i	j	$A[i][j]$	comparison with k.
0	5	0	13	$A[5][0] > k$
1	4	0	11	$A[4][0] > k$
$\vdots$				$\vdots$
				$A[...][...] == k$ (return True)

Solution. For $k = 20$, we have the following steps.

Loop Iteration #	i	j	$A[i][j]$	comparison with k.
0	5	0	13	$A[5][0] < 20$
1	5	1	17	$A[5][1] < 20$
2	5	2	22	$A[5][2] > 20$
2	4	2	19	$A[4][2] < 20$
4	4	3	21	$A[4][3] > 20$
5	3	3	18	$A[3][3] < 20$
6	3	4	20	$A[3][3] == 20$ (return True)

For $k = 10$, we have the following steps:

Loop Iteration #	i	j	$A[i][j]$	comparison with k.
0	5	0	13	$A[5][0] > 10$
1	4	0	11	$A[4][0] > 10$
2	3	0	5	$A[3][0] < 10$
3	3	1	8	$A[3][1] < 10$
4	3	2	9	$A[3][2] < 10$
5	3	3	18	$A[3][2] > 10$
6	2	4	17	$A[2][2] > 10$
7	1	4	11	$A[1][2] > 10$
8	0	4	7	$A[0][2] < 10$
9	0	5	11	$A[0][5] > 11$
10	-1	5	-	return False

▲

(b, 5 points) Prove the following property for the saddleback algorithm using induction.

> At the beginning of each loop iteration, if k belongs to the matrix A, it must belong specifically to the submatrix $A[0 : i + 1][j : n]$ (Recall: in Python notation the range $a : b$ contains a but not b).

Use weak induction on m: the number of loop iterations in `saddleBackSearch` method.

Solution. We prove the theorem using weak induction on m the number of loop iteration in `saddleBackSearch` method.
Base Case: $m = 0$. At the beginning, we have $i = n - 1$ and $j = 0$. Therefore, the statement is trivially true. Since $A[0 : i + 1][j : n]$ now includes the entire matrix and trivially if k is found in A, it will be found in $A[0 : i + 1][j : n]$
Ind. Hyp.: Assume that the statement is true for m, we wish to prove that it remains true for iteration $m + 1$.

Let i_m, j_m be the values of i, j at the start of the m^{th} loop iteration and i_{m+1}, j_{m+1} denote the values at the start of the $(m + 1)^{th}$ loop iteration.
Case -1 : $i_m \geq 0$ and $j_m < n$ holds. The loop executes one step. We now distinguish between three cases:

- $A[i_m][j_m] == k$ In this case, the algorithm returns True, correctly.
- $A[i_m][j_m] > k$ Assuming that the range $A[0 : i_m + 1][j_m : n]$ has the element k in it, and the array A is sorted, we know that the row i_m cannot have k in it. Therefore, k can only be found in $A[0 : i_m][j_m : n]$. Note that in this case, $i_{m+1} =: i_m - 1, j_{m+1} =: j_m$ and therefore, we conclude that k can only be found in $A[0 : i_{m+1}][j_{m+1} : n]$.
- $A[i_m][j_m] < k$ In this case, assuming that the range $A[0 : i_m + 1][j_m : n]$ has the element k in it, and the array A is sorted, we know that the column j_m cannot have k in it. Therefore, k can only be found in $A[0 : i_m + 1][j_m + 1 : n]$. Note that in this case, $j_{m+1} =: j_m + 1, i_{m+1} =: i_m$ and therefore, we conclude that k can only be found in $A[0 : i_{m+1}][j_{m+1} : n]$.

Case -2: The loop condition does not hold. In this case, we have that the range $A[0 : i+1][j : n]$ is the empty range and by induction hypothesis, the element k will not be in the range. Therefore, the algorithm correctly returns False.

▲

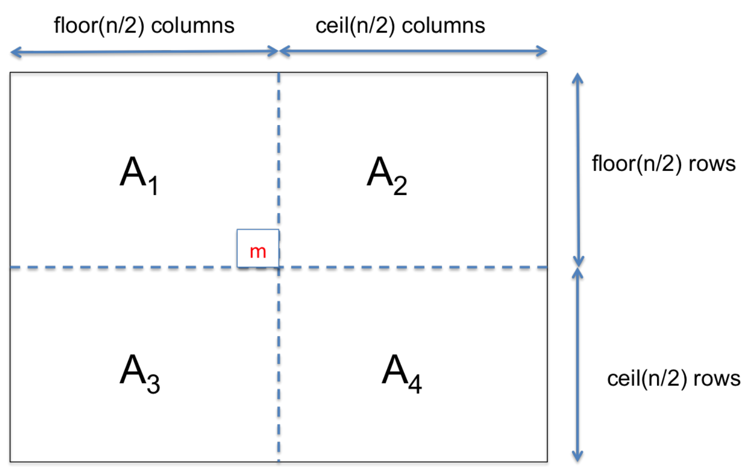
(c, 5 points) What is the worst case time complexity of `saddleBackSearch` as a function of n? Express your answer using the Θ notation.

Solution. At each step, either the value of i decreases or j increases. As we know, $i \in [0, n]$ and $j \in [0, n]$. In the worst case, the algorithm can run for $2n$ loop iterations. This is $\Theta(n)$ worst case running time. ▲

(d, 5 points) Prof. X, a renowned expert in search wishes to solve this problem using a divide and conquer algorithm. She formulates the following idea as an adaptation of binary search:

1. Compare k with $m =: A[\lfloor n/2 \rfloor][\lfloor n/2 \rfloor]$, the "center" element of the matrix A.
2. Partition the matrix into four *roughly equal* submatrices of sizes $\frac{n}{2} \times \frac{n}{2}$.
3. Find out which submatrices can possibly contain k based on the outcome of the comparison in the first step.

Pictorially, we depict the partitioning of A as below:



Let A_1, A_2, A_3, A_4 be the four parts as shown above.

1. If $m < k$ then prove that A_1 cannot contain k.
2. If $m > k$ then prove that A_4 cannot contain k.

Solution. First, we note the following simple fact due to the sorted nature of matrix A.
Every element of A_1 is $\leq m$ and every element of $A_4 \geq m$.
Therefore, if $m < k$ then every element of A_1 is also less than k.
If $m > k$ then every element of A_4 is also greater than k.
Therefore, we conclude that

1. If $m < k$ then prove that A_1 cannot contain k.
2. If $m > k$ then prove that A_4 cannot contain k.

▲

(e, 5 points) Based on the observations above, complete the divide and conquer algorithm for searching a sorted matrix. You may simply write your algorithm down as pseudocode or python code. Please use recursion. This will not need more than 10 lines or so.

Solution.

```
import numpy

def divConquerSearch(A,m,n,k):
        if ( n < 2 or m < 2):
                # Base case simply scan every element of A
                return scanAndSearch(A,m,n,k)
    r = n//2
    s = n - r
    m = A[r][r]
    # Create four sub matrices
    A1 = A[0:r][0:r]
    A4 = A[r:n][r:n]
    A2 = A[r:n][0:r]
    A3 = A[0:r][r:n]
    if (m == k):
        return True

    if (m < k):
        return ( divConquerSearch(A2,s,r,k) or
                 divConquerSearch(A3,r,s,k) or
                 divConquerSearch(A4,s,s,k) )
    if ( m > k):
        return ( divConquerSearch(A2,s,r,k) or
                 divConquerSearch(A3,r,s,k) or
                 divConquerSearch(A1,r,r,k) )

    # I cannot reach this point of the code.
```

▲

(f, 5 points) Derive a recurrence for $T(n)$ the worst case running time for the divide and conquer search algorithm on a $n \times n$ matrix A.

Solve the recurrence using master theorem. Mention the case of master theorem you will need to use and the worst case running time obtained.

Solution.

The recurrence will be

$$T(n) = 3T(\frac{n}{2}) + cn,\ T(n) = 1 \text{ for } n \leq 1$$

Master theorem case-1 applies, and yields

$$T(n) = \Theta(n^{\log_2(3)})$$

▲

P3 (10 points) Use the expansion method to solve the following recurrences. Express your final answer in the Θ notation. **Do not use master method**, though wherever possible you can compare what you obtain with the results from applying master method.

1. $T(n) = 4T(\frac{n}{2}) + n$ with $T(n) = 1$ whenever $n \leq 1$.
2. $T(n) = 8T(\frac{n}{2}) + n^3$ with $T(n) = 1$ whenever $n \leq 1$.
3. $T(n) = T(\sqrt{n}) + c$ with $T(n) = 1$ whenever $n \leq 2$.

Solution.

1. $T(n) = 4T(\frac{n}{2}) + n$ with $T(n) = 1$ whenever $n \leq 1$.

$$\begin{aligned}
T(n) &= 4T(\tfrac{n}{2}) + n \\
&= 4^2T(\tfrac{n}{4}) + 4(\tfrac{n}{2}) + n \\
&= 4^3T(\tfrac{n}{8}) + 4^2(\tfrac{n}{4}) + 4\tfrac{n}{2} + n \\
&\vdots \\
&= 4^jT(\tfrac{n}{2^j}) + 4^{j-1}\tfrac{n}{2^{j-1}} + \cdots + 4^1\tfrac{n}{2^1} + n \\
&= 4^{\log_2(n)}T(1) + n\textstyle\sum_{j=0}^{\log_2(n)} \tfrac{4^j}{2^j} \\
&= 4^{\log_2(n)} + n(2^{\log_2(n)+1} - 1) \\
&= n^{\log_2(4)} + n(2n-1) \ = \ \Theta(n^2)
\end{aligned}$$

2. $T(n) = 8T(\frac{n}{2}) + n^3$ with $T(n) = 1$ whenever $n \leq 1$.

$$\begin{aligned}
T(n) &= 8T(\tfrac{n}{2}) + n^3 \\
&= 8^2T(\tfrac{n}{2^2}) + 8(\tfrac{n}{2})^3 + n^3 \\
&= 8^3T(\tfrac{n}{2^3}) + 8^2(\tfrac{n}{2^2})^3 + 8(\tfrac{n}{2})^3 + n^3 \\
&\vdots \\
&= 8^jT(\tfrac{n}{2^j}) + \textstyle\sum_{i=0}^{k} 8^i \tfrac{n^3}{2^{3i}} \\
&= 8^{\log_2(n)} + \textstyle\sum_{i=0}^{\log_2(n)} n^3 \\
&= n^3 + n^3\log_2(n) \ = \ \Theta(n^3\log_2(n))
\end{aligned}$$

3. $T(n) = T(\sqrt{n}) + c$ with $T(n) = 1$ whenever $n \leq 2$.

$$\begin{aligned} T(n) &= T(\sqrt{n}) + c \\ &= T(n^{\frac{1}{4}}) + 2c \\ &= T(n^{\frac{1}{8}}) + 3c \\ &\vdots \\ &= T(n^{\frac{1}{2^k}}) + kc \qquad \text{Note: } n^{\frac{1}{2^k}} - = 2 \text{ for } k = \log_2(\log_2(n)) \\ &= T(2) + c\log_2(\log_2(n)) \ = \ \Theta(\log(\log(n))) \end{aligned}$$

▲