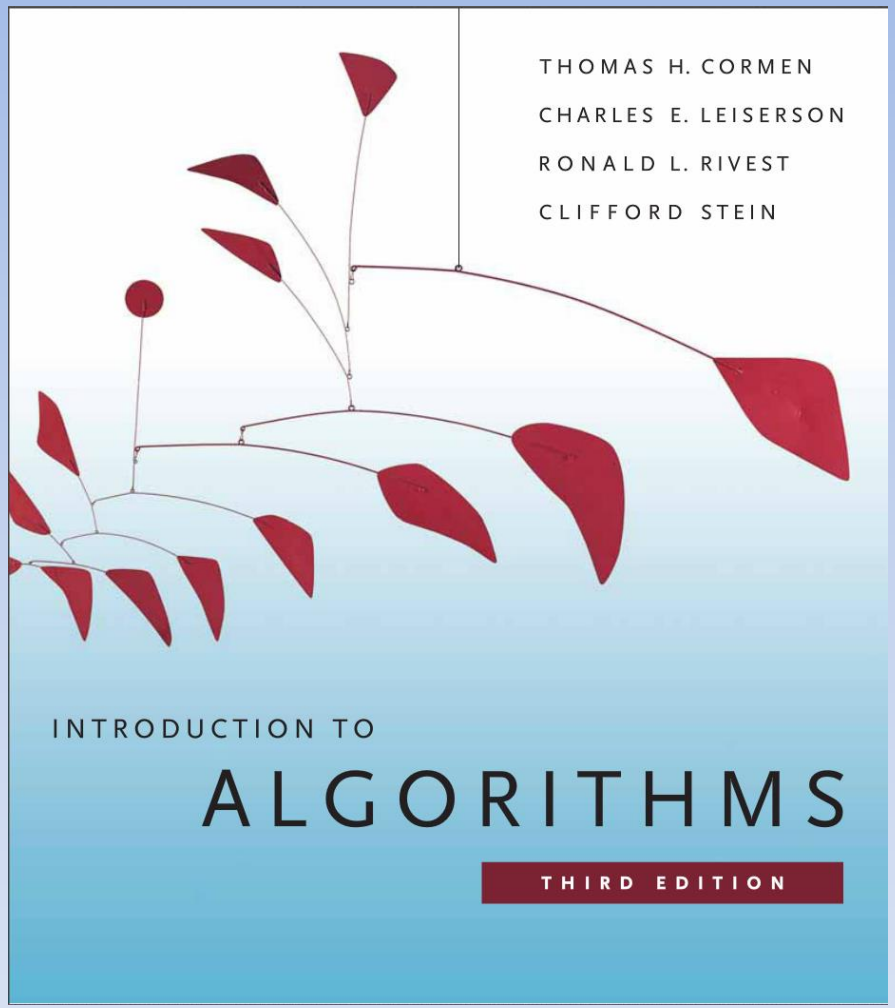
Design and Analysis of Algorithms



Midterm Results

(1) Use the Master Method to solve the recurrences below?

(a) $T(N) = 3*T(N/2) + N^3$

(b) $T(N) = 5*T(5N/8) + 8$

(c) $T(N) = 64*T(N/4) + N^3$

Midterm Results

- (a) $T(N) = 3T(N/2) + N^3$
  - a=3
  - b=2
  - d=3
  - a=3 < 8=b^d

$$T(N) = O(N^d)$$
$$T(N) = O(N^3)$$

- (b) T(N) = 5T(5N/8) + 8
  - a=5
  - b=(8/5)
  - d=0
  - a=5 > 1=b^d

$$T(N) = O(N^{\log_b a})$$
$$T(N) = O(N^{\log_{8/5} 5})$$

- (c) $T(N) = 64T(N/4) + N^3$
  - a=64
  - b=4
  - d=3
  - a=64 = 64=b^d

$$T(N) = O(N^d \log N)$$
$$T(N) = O(N^3 \log N)$$

(2) For each of the following patterns state yes or no as to whether a hash table could provide significant speed-up in a computer program and explain why:

(a) Repeated Maximum Computations:

why?

(b) Repeated lookups:

why?

(c) Repeated minimum computations

why?

- (a) Repeated Maximums:
  - No, calculating the maximum still requires looking at all items.
- (b) Repeated Lookups:
  - Yes, Reason for hash tables.
- (c) Repeated Minimums:
  - No, calculating the minimum still requires looking at all items.

(3) Consider the following scenario:

Let's say your friend has just brought you this new recursive algorithm to improve upon a problem solution you developed. He says he's sure it's significantly faster than your version because it uses a really interesting recursive approach.

You are intrigued, so you look carefully at his recursive algorithm. You develop the following pseudocode to model his algorithm:

```
friendRecursiveSolution(Problem)

1:      // If problem is real small, then solve in constant time.

2:      // Break Problem into 27 subproblems each 1/3 of original size in constant time.

3:      for each subproblem:

4:              friendRecursiveSolution(subProblem)

5:      // Return Merge of all 27 subproblems in O(n^2) time (Complex Tricky Routine)
```

Now model this algorithm with a recurrence relation like we did in class, then solve the recurrence using the Master Method given in class and explain the time complexity of this algorithm.

- $T(N) = 27T(N/3) + N^2$
- a=27, b=3, d=2
- a=27 > 9 = b^d

$$\boldsymbol{T(N) = O(N^{log_b a})}$$
$$\boldsymbol{T(N) = O\left(N^{log_3 27}\right)}$$
$$T(N) = O(N^3)$$

(4) Your friend from previous questions returns to explain he has been working diligently on his divide and conquer strategy! He explains that he has managed to reduce the time complexity of the final Merge routine used to merge subsolutions, getting time complexity down to CONSTANT TIME! Model the new recurrence relation and explain how your friend's improvement affects the time complexity of his algorithm.

- T(N) = 27T(N/3) + c
- a=27, b=3, d=0
- a=27 > 1 = b^d

$$\boldsymbol{T(N) = O(N^{log_b\, a})}$$

$$\boldsymbol{T(N) = O\left(N^{log_3\, 27}\right)}$$

$$T(N) = O(N^3)$$

- Performance has not changed because merge performance is not impacting total performance because of the high number of subproblems relative to the size reduction.

(5) Our balanced binary tree algorithms depended upon rotating the tree at a node. Below is an incomplete version of one of those algorithms. Complete the algorithm and draw the tree resulting from applying algorithm to root in example on the next page:

___Left__-Rotate(T, x)

1: y = ___x.right_____// y set here & used below

2: x.right = __y.left_____// Now change right-child of x

3: **if** y.left ≠ T.nil

4: y.left.p = x

5: y.p = _x.p_____// Now change the parent of y

6: **if** x.p == T.nil

7: T.root = ___y_____// Update root in this case

8: **elseif** x == x.p.left

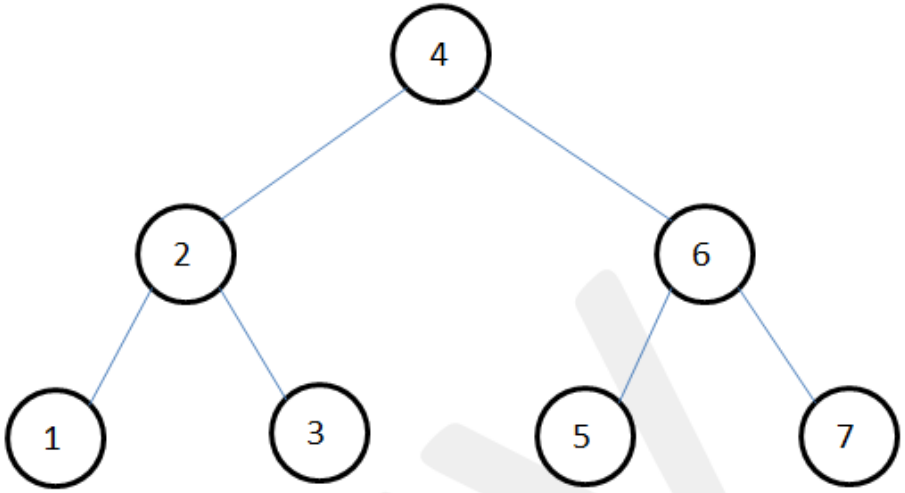
9: x.p.left =__ y _____// update x's parent node

10: **else** x.p.right = __ y _____// update x's parent node

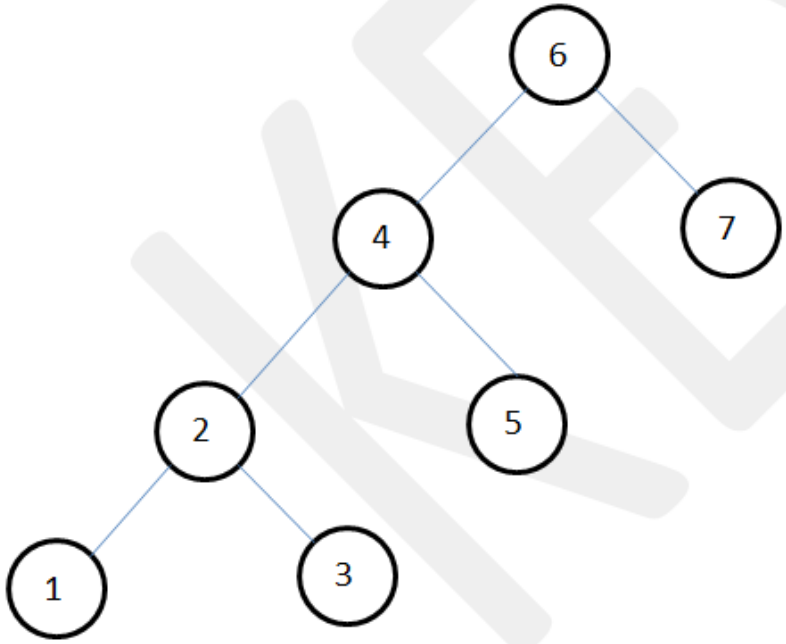
11: y.left = _____ x _____// y's new left

12: x.p = _____ y _____// x's new parent

TREE BEFORE ROTATION:



TREE AFTER ROTATION AT ROOT:



(6) With our rotations in hand, we can fixup problems with our red-black tree after inserting a new node. Fill in the missing pieces of the RB-Insert-Fixup routine then draw a before and after example of one case on the following page:

```
RB-Insert-Fixup( T, z )
1:    while z.p.color == RED
2:        if z.p == z.p.p.left
3:            y = z.p.p.right
4:            if y.color == RED
5:                z.p.color     = BLACK
6:                y.color       = BLACK
7:                z.p.p.color   = RED
8:                z = z.p.p
9:            else if z == z.p.right
10:                   z = z.p
11:                   Left-Rotate(T, z)
12:               z.p.color     = BLACK
13:               z.p.p.color   = RED
14:               Right-Rotate(T, z.p.p)
15:       else (same as then clause
                with "right" and "left" exchanged)
16:   T.root.color = BLACK
```

(7) In order to solve the Bay Bridges Challenge we needed to determine which bridges intersected. Below is the algorithm provide by our textbook. Fill in the code missing from the new function MidtermConditionalExpression on the following page to complete the algorithm.

```
Direction( pi, pj, pk )
1:      return (pk – pi) X (pj – pi)

Segments-Intersect( p1, p2, p3, p4 )
1:      d1 = Direction (p3, p4, p1)
2:      d2 = Direction (p3, p4, p2)
3:      d3 = Direction (p1, p2, p3)
4:      d4 = Direction (p1, p2, p4)
5:      if MidtermConditionalExpression(d1, d2, d3, d4)
6:              return TRUE
7:      elseif d1 == 0 and On-Segment(p3, p4, p1)
8:              return TRUE
9:      elseif d2 == 0 and On-Segment(p3, p4, p2)
10:             return TRUE
11:     elseif d1 == 0 and On-Segment(p3, p4, p1)
12:             return TRUE
13:     elseif d2 == 0 and On-Segment(p3, p4, p2)
14:             return TRUE
15:     else return FALSE
```

NOW COMPLETE THE FUNCTION ON THE NEXT PAGE

MidtermConditionalExpression(d1, d2, d3, d4)

1: **if (**

2: _____

3: _____

4: _____

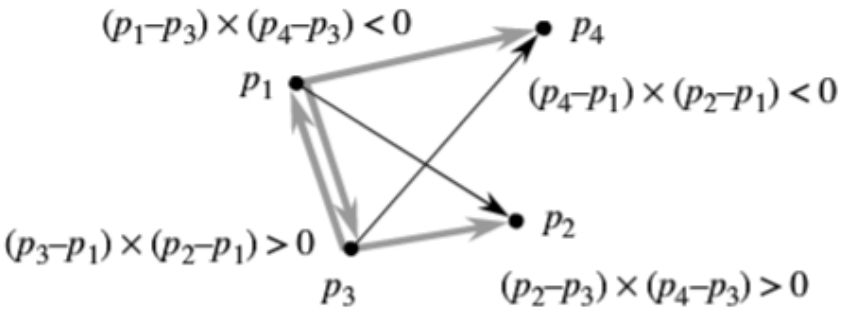
5: _____**)**

6: return TRUE

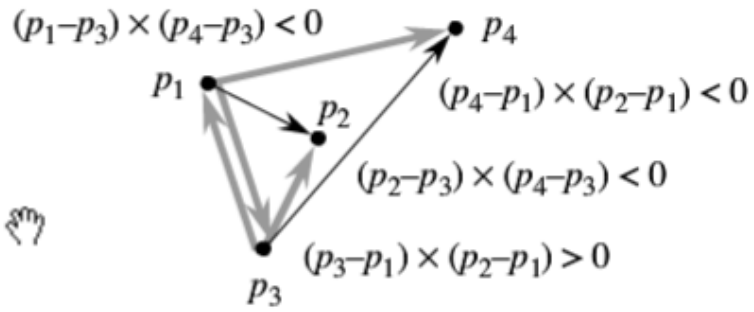
7: **else return** FALSE

NOW DESCRIBE THE FUNCTION ON THE NEXT PAGE

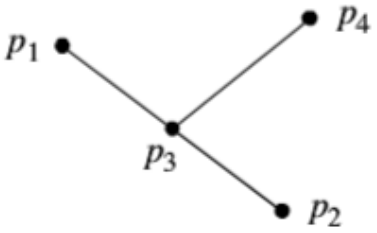
```
SEGMENTS-INTERSECT(p1, p2, p3, p4)
 1  d1 = DIRECTION(p3, p4, p1)
 2  d2 = DIRECTION(p3, p4, p2)
 3  d3 = DIRECTION(p1, p2, p3)
 4  d4 = DIRECTION(p1, p2, p4)
 5  if ((d1 > 0 and d2 < 0) or (d1 < 0 and d2 > 0)) and
        ((d3 > 0 and d4 < 0) or (d3 < 0 and d4 > 0))
 6      return TRUE
 7  elseif d1 == 0 and ON-SEGMENT(p3, p4, p1)
 8      return TRUE
 9  elseif d2 == 0 and ON-SEGMENT(p3, p4, p2)
10      return TRUE
11  elseif d3 == 0 and ON-SEGMENT(p1, p2, p3)
12      return TRUE
13  elseif d4 == 0 and ON-SEGMENT(p1, p2, p4)
14      return TRUE
15  else return FALSE
```


$(p_1-p_3) \times (p_4-p_3) < 0$
p_4
p_1
$(p_4-p_1) \times (p_2-p_1) < 0$
p_2
$(p_3-p_1) \times (p_2-p_1) > 0$
p_3
$(p_2-p_3) \times (p_4-p_3) > 0$

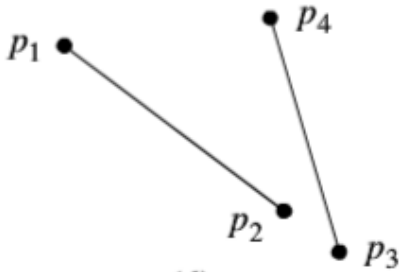
(a)


$(p_1-p_3) \times (p_4-p_3) < 0$
p_4
p_1
p_2
$(p_4-p_1) \times (p_2-p_1) < 0$
$(p_2-p_3) \times (p_4-p_3) < 0$
$(p_3-p_1) \times (p_2-p_1) > 0$
p_3

(b)


p_1
p_4
p_3
p_2

(c)


p_1
p_4
p_2
p_3

(d)