Remi Galvez
CSCI 3212 - Assignment II
Monday, October 12, 2015

Problem 1

(a) Let t denote the number of iterations that will take place in the while-loop.

The loop will run as long as $s < n$, that is:

$$s = \Sigma_{i=1}^{t} i = \frac{t(t+1)}{2} < n$$

Because we are using the order notation, $t(t+1) \sim t^2$.

Therefore,

$$\begin{aligned} \frac{t^2}{2} < n \quad &\Longleftrightarrow \quad t^2 < 2n \\ &\Longleftrightarrow \quad t < \pm\sqrt{2n} \\ &\therefore \quad f = O(\sqrt{n}) \end{aligned}$$

The algorithm will take $O(\sqrt{n})$ time.

(b)

$$\begin{aligned} &O\left(n \cdot \left(n^2 + \left(log\,(n) \cdot \left(3n + \frac{3n^2}{log\,(n)}\right)\right)\right)\right) \\ = \quad &O\left(n \cdot \left(n^2 + 3n \cdot log\,(n) + 3n^2\right)\right) \\ = \quad &O\left(n^3 + 3n^2 \cdot log\,(n) + 3n^3\right) \\ = \quad &O\left(n^3\right) \end{aligned}$$

(c) To maximize the number of interior nodes in a trie, we must minimize the number of leaves. For any trie with at least 2 nodes, the tree will have at least two leaves, and therefore at most $(n-2)$ interior nodes, which is on the order o n.

(d) Flesh out the pseudocode with details (base cases, and the recursive calls). Then, analyze the algorithm's execution time: what is the worst-case execution time in terms of m and n?

```
**Algorithm**: textSearch ($text, pattern$)
**Input**: two char arrays text and pattern of lengths $n$ and $m$
    for $i = 0$ to $i = n - 1$
        if recursiveMatch ($i, 0, m - 1$)
            return true
        endif
    endfor
    return false
```

```
**Algorithm**: recursiveMatch ($i, p, q$)
**Input**: See if there's a match at position $i$ in text with the chars in the pattern
between positions $p$ and $q$
    // Base cases
    if $p == q$
        if $text[p] == text[q]$
            return true
        else
            return false
        endif
    endif

    // Find the middle index.
    $m = (p + q)/2$
    // Call recursive functions.
    if recursiveMatch ($i, p, m$) is false
        return false
    else if     recusiveMatch ($i + 1, m + 1, q$) is false
        return false
    endif
    return true
```

The recursive call will take $log(m)$ time, as the pattern is split into two equal parts continuously until we are left with single characters. This method, in turn, will be called n times. Hence,

The best case execution for this algorithm is $\boxed{O(log(m))}$

The worst case execution time for this algorithm is $\boxed{O(n \cdot log(m))}$

(e) The algorithms hinted at in the article are different from the ones we have studied in class as they are machine learning algorithms, which have error rates, and often require human parameterization. This means that these algorithms will misclassify data points and will make mistakes. Furthermore, the data used is subject to human bias, as the parameterization of the algorithm (how attributes are weighted, setting up the base parameter, how many iterations to run the algorithm, etc.). There are infinite ways classify two human beings, while there is only one way for an array to be sorted.

(f) The 'K' in the KMP text search algorithm stands for Knuth, obviously referring to Donald Ervin Knuth, one of the creators of the KMP algorithm. Donald Ervin Knuth is a very influential mathematician and computer scientist, and has contributed major progress to both of these fields. His biggest work is the 7 volume work, The Art of Computer Programming. He was initially asked to write a book about compilers, however he felt that the foundations needed to be written about before he could write the initially assigned book, and that these writings could be split up into 6 volumes.

Dr. Knuth also contributed a great deal to algorithm analysis; he has been called the "father" of the field. Dr. Knuth was the first to popularize the Big O notation for algorithm analysis.

The two priorly listed contributions were major contributions from a theoretical standpoint, but Knuth also created and implemented the TEXcomputer typesetting system (on which this file was created). This tool is very widely used around the world by a variety of people and professions, like researchers and students.

Problem 3

Algorithm: makeFilterTree ($rectSet$)
Input: Set of rectangles containing n rectangles

```
    $root$ = createNode(0,100)
    for $r$ in $rectSet$
        makeFilterTreeRecursive ($r, root$)
    endfor
    return root
```

Algorithm: makeFilterTreeRecursive ($r, quadrantNode$)
Input: Single rectangle to be placed in tree and current iteration node.

```
    // Compute appropriate subquadrant as an int to place node
    $quadrant$ = computeQuadrant ($r, quadrantNode$)

    // If rectangle intersects with several quadrants
    if $quadrant == -1$
        // Add rectangle to parent input node
        add $r$ to $quadrantNode$
        return
    endif

    // If smaller subquadrant doesn't exist
    if $quadrantNode[quadrant]$ is $null$
        // Create subquadrant
        Add new node to $quadrantNode.quadrants[quadrant]$
    endif

    // Call recursive function with new quadrant
    makeFilterTreeRecursive ($r, quadrantNode.quadrants[quadrant]$)
```

Algorithm: computeQuadrant ($r, quadrantNode$)
Input: Single rectangle and parent node

```
    // Determine which half (horizontally) holds the top left corner's $x$-coordinate
    if ($r.topLeft.x < quadrantNode.midX$)
        $x = 0$
    else
        $x = 1$
    endif

    // Determine which half (vertically) holds the top left corner's $y$-coordinate
    if ($r.topLeft.y < quadrantNode.midY$)
```

```
        $y = 0$
    **else**
        $y = 1$
    **endif**

    // Compute subquadrant for bottom right corner
    **if** $(x == 0 \quad \&\& \quad y == 0)$ $topLeftQuad = 2$
    **else if** $(x == 0 \quad \&\& \quad y == 1)$ $topLeftQuad = 1$
    **else if** $(x == 1 \quad \&\& \quad y == 0)$ $topLeftQuad = 3$
    **else** topLeftQuad = 0
    **endif**

    // Determine which half (horizontally) holds the bottom right corner's $x$-
coordinate
    **if** $(r.bottomRight.x < quadrantNode.midX)$
        $x = 0$
    **else**
        $x = 1$
    **endif**

    // Determine which half (vertically) holds the bottom right corner's $y$-
coordinate
    **if** $(r.bottomRight.y < quadrantNode.midY)$
        $y = 0$
    **else**
        $y = 1$
    **endif**

    // Compute subquadrant for bottom right corner
    **if** $(x == 0 \quad \&\& \quad y == 0)$ $bottomRightQuad = 2$
    **else if** $(x == 0 \quad \&\& \quad y == 1)$ $bottomRightQuad = 1$
    **else if** $(x == 1 \quad \&\& \quad y == 0)$ $bottomRightQuad = 3$
    **else** bottomRightQuad = 0
    **endif**

    // If both corners are in the same subquadrant, return taht subquadrant's
index
    **if** $(topLeftQuad == bottomRightQuad)$
        **return** $topLeftQuad$
    **endif**

    **return** -1
```