HW4

Ronan Wallace

April 3, 2019

1) Program

Program located in separate file (see .zip folder).
File Name: **HW4 - Longest Common Prefix.py**

2) Binary Tree Pseudocode

ALGORITHM checkFull(node)
//Input: A binary tree (technically, the root of the tree is initially passed as a parameter into the algorithm).
//Output: True - tree is full; False - tree is not full
if left node **is** Null and right node **is** Null **do**
return True
else if left node **is not** Null and right node **is not** Null **do**
if *checkFull*(left node) **is** True and *checkFull*(right node) **is** True **do**
return True
else do
return False
else do
return False

Time Complexity:
Best Case: $\Theta(1)$
Worst Case: $\Theta(n)$ where n is the number of nodes in the tree

3-4) Section 5.5

1a) ALGORITHM Divide(A[n])

//Input: An array of n real numbers where $n \geq 2$ and is sorted in non-decreasing order.

//Output: Returns the pair of closest numbers

if $n > 3$

$middle \leftarrow n/2$

$leftHalf \leftarrow A[0 : \lfloor middle \rfloor]$

$rightHalf \leftarrow A[\lceil middle \rceil : n]$

$ClosestPairLeft \leftarrow Divide(leftHalf)$

$ClosestPairRight \leftarrow Divide(rightHalf)$

return $FindClosestPair(ClosestPairLeft,\ ClosestPairRight)$

ALGORITHM FindClosestPair($leftHalf[0..1]$, $rightHalf[0..1]$)

//Input: Two arrays, each containing two integers

//Output: An array of two integers (Closest Pair of Numbers)

if $leftHalf[1]$ - $leftHalf[0] < rightHalf[1]$ - $rightHalf[0]$ **do**

$ClosestPair = leftHalf$

else do

$ClosestPair = rightHalf$

if $rightHalf[0]$ - $leftHalf[1] < dist$ **do**

$ClosestPair = [leftHalf[1], rightHalf[0]]$

return $ClosestPair$

Time Complexity:

$\Theta(n \log n)$

1b) It is not a good algorithm because it can easily be done using Brute Force (iterating through the array of integers and making comparisons every two elements).

6a) Picture(s) located in separate file (see .zip folder).
File Name: **HW4 - Section 5.5 - Question 6a.pdf**

6b) After several observations using Voronoi visualization websites, I think I have figured out how the solution to the previous problem is applied to the general case. When there are three points, you draw a perpendicular line exactly at the half way point of every edge between each pair of points in a given set. Where these lines intersect with other lines is where they would terminate.

5-7) Section 6.3

2a) Picture(s) located in separate file (see .zip folder).
File Name: **HW4 - Section 6.3 - Question 2a.pdf**

7a) Picture(s) located in separate file (see .zip folder).
File Name: **HW4 - Section 6.3 - Question 7a.pdf**

9) ALGORITHM findDifference(tTT) where tTT is the root of a given Two Three Tree

```
//Input: A standard 2-3 tree consisting of integers.
//Output: The difference between the largest integer and the smallest integer in the given tree.
return findLargest(tTT) - findSmallest(tTT)

define findLargest(tTT):
    if tTT.right is not equal to None do
        findLargest(tTT.right)
    if tTT.right is equal to None do
        return tTT

define findSmallest(tTT):
    if tTT.left is not equal to None do
        findSmallest(tTT.left)
    if tTT.left is equal to None do
        return tTT
```

8) AVL Tree Rotation

Picture(s) located in separate file (see .zip folder).
File Name: **HW4 - AVL Tree Rotation.pdf**

9) 2-3 Tree

Picture(s) located in separate file (see .zip folder).
File Name: **HW4 - 2-3 Tree.pdf**

10a-e) Max-Heap

a) The second largest value will be one of the children of the root of the heap.

b) The third largest value will either be the other child of the root,

or one of the children of the second largest value.

c) If the second largest and the third largest values are the children of the root, the fourth largest will be one of the children of either the second largest value or third largest value. If the third largest value is a child of the second largest value, then the fourth largest value will either be the other child of the root, the other child of the second largest value, or one of the children of the third largest value.

d) The smallest value will be one of the leaves of the heap.

e) The second smallest value will either be one of the leaves of the heap or the parent of the smallest value of the heap.

11-14) Section 6.4

1a) Picture(s) located in separate file (see .zip folder).
File Name: **HW4 - Section 6.4 - Question 1a.pdf**

1b) Picture(s) located in separate file (see .zip folder).
File Name: **HW4 - Section 6.4 - Question 1b.pdf**

1c) No, the outputs for both algorithms will not always be the same.

2) PROGRAM (See .zip folder)
File Name: **HW4 - Heap Validation.py**

6a) Unsorted array:
Insert: $\Theta(1)$
Delete: $\Theta(n)$
Find: $\Theta(n)$

6b) Sorted array:
Insert: $\Theta(n)$
Delete: $\Theta(n)$
Find: $\Theta(1)$

6c) Binary search tree:
Insert Average: $\Theta(logn)$
Insert Worst: $\Theta(n)$
Delete Average: $\Theta(logn)$
Delete Worst: $\Theta(n)$
Find Average: $\Theta(logn)$
Find Worst: $\Theta(n)$

6d) AVL tree:
Insert: $\Theta(logn)$

Delete: $\Theta(logn)$
Find: $\Theta(logn)$

6e) Heap:
Insert Best/Average: $\Theta(1)$
Insert Worst: $\Theta(logn)$
Delete: $\Theta(logn)$
Find: $\Theta(n)$

7a) Picture(s) located in separate file (see .zip folder).
File Name: **HW4 - Section 6.4 - Question 7a.pdf**

7b) Picture(s) located in separate file (see .zip folder).
File Name: **HW4 - Section 6.4 - Question 7b.pdf**

7c) Picture(s) located in separate file (see .zip folder).
File Name: **HW4 - Section 6.4 - Question 7c.pdf**