**CSCE 221-200: Honors Data Structures and Algorithms  
Assignment Cover Page  
Spring 2021**

|  |  |
| --- | --- |
| **Name:** | Priyanshu Barnwal |
| **Email:** | priyanshu@tamu.edu |
| **Assignment:** | Homework 2 |
| **Grade (filled in by grader):** |  |

Please list below all sources (people, books, webpages, etc) consulted regarding this assignment (use the back if necessary):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CSCE 221 Students** | **Other People** | **Printed Material** | **Web Material (give URL)** | **Other Sources** |
| 1. | 1. | 1. | 1. <http://homes.ieu.edu.tr/culudagli/files/CE221/Week_10/Recitation%206%20-%20AVL.pdf> | 1. |
| 2. | 2. | 2. | 2. https://www.google.com/search?q=empty+binary+tree&rlz=1C1RXQR\_enUS931US931&source=lnms&tbm=isch&sa=X&ved=2ahUKEwivjqfC3OruAhUEXc0KHUt7AkAQ\_AUoAXoECBQQAw&biw=1396&bih=744#imgrc=0hgAEf7vp52KeM | 2. |
| 3. | 3. | 3. | 3. https://stackoverflow.com/questions/6483860/exam-question-about-inserting-to-a-empty-binary-search-tree | 3. |
| 4. | 4. | 4. | 4. | 4. |
| 5. | 5. | 5. | 5. | 5. |

Recall that TAMU Student Rules define academic misconduct to include acquiring answers from any unauthorized source, working with another person when not specifically permitted, observing the work of other students during any exam, providing answers when not specifically authorized to do so, informing any person of the contents of an exam prior to the exam, and failing to credit sources used. *Disciplinary actions range from grade penalty to expulsion.*

"On my honor, as an Aggie, I have neither given nor received unauthorized aid on this academic work. In particular, I certify that I have listed above all the sources that I consulted regarding this assignment, and that I have not received or given any assistance that is contrary to the letter or the spirit of the collaboration guidelines for this assignment."

|  |  |
| --- | --- |
| **Signature:** |  |
| **Date:** |  |

Exercise 3.20:

1. Advantages – We save times when trying to shift arrays because there is no shifting to be done. Also, since we do not have to shift anything, the runtimes for lazy deletion are much shorter.

Disadvantages – It is a waste of memory to keep a deleted item in a list, not to mention difficult to deal with overall because it has to be searched over when looking for a specific item. Since we have to mark items before deleting them, we will also be forced to search for the item twice instead of once like normal deletion.

Exercise 3.24:

We start with two indexes, one at the front and one at the back, to show exactly where the array starts and ends. Each of these ends is one stack and it splits down the middle. After this we can simply use normal array operations to add and remove from the linked list.

Exercise 3.25:

1. Using a Stack ADT we can minimize the total running time to O(1), even for push and pop operations. We can build two stack in our ADT, one to store minimum values and another to store all other values. This way, when we want the minimum value, we simply have to pull from the minimum value stack. When pushing, we will simply push to the normal value stack no matter what, and then we can check whether or not the top value in the minimum value stack is less than the value we are pushing. If it is, we will not put onto the minimum value stack, but if it is not, we can add it to the second stack as the new minimum value. For pop, we can pop the top element of the normal stack. If it is equal to the top element of the minimum stack, we can pop it from there as well. And now, GetMin is easy, as we can just return the top value from the second stack.

Exercise D:

#include <iostream>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

char\* stack;

int top = -1;

void push(char letter)

{

stack[top+1] = letter;

top++;

}

char pop()

{

top--;

return stack[top];

}

bool PalindromeCheck(char word[])

{

int length = strlen(word);

stack = (char\*)malloc(length \* sizeof(char));

int middle = length / 2;

int i;

for (i = 0; i < middle; i++) {

push(word[i]);

}

if (length % 2 != 0) {

i++;

}

while (word[i] != '\0') {

char letter = pop();

if (letter != word[i])

return false;

i++;

}

return true;

}

Exercise E:

The postorder and preorder traversal cannot visit the nodes in the same order. The postorder must hit an external node while the preorder must hit the root node. It is possible to have the same path in reverse order, because of how the traversal works, though. With the same starting and ending points, there is only one possible path.

Exercise 4.5:

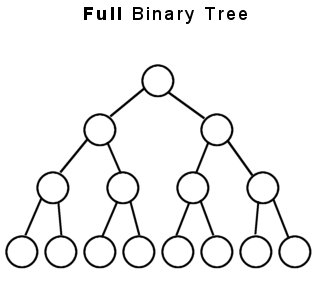
Since each node can have two subnodes, and everything starts with a root node, a binary tree with a height of 1 has a root node with two subnodes, or three nodes in total. After that, the number of nodes increases by 2^h. if there is a height of two, then the nodes increase by 2^2, or four nodes, because there are two subnodes that split off into four. Then it would be 2^2 + 2^1 + 1. As it keeps increasing, it adds 2^h each time. This is clearly a binary function that grows to 2^(h+1) – 1.

Exercise 4.8:

Prefix Expression: - e \* + c d \* a b

Infix Expression: ((a + b) \* (c \* d)) – e

Postfix Expression: a b + c d \* \* e –

Exercise 4.9:

9

3

1. 4

2 6

7

2. Same thing as above, except the 1 and 2 are now subnodes of 4 instead of 3. (On the left side).

Exercise 4.18:

1. 2583

Exercise 4.19:

I cannot make another tree like I did in the previous question, formatting the text in that thing was the hardest part of this assignment.

4 -> (2, 6) -> (1,3,5,9) -> ( , , , , , ,7, )

Exercise 4.25:

1. 1.44 log(N + 2)−1.328 is the formula for height of an AVL tree. We can cut this down (because of the way runtime works) to O(log(N)). Since that formula must be equal to (2^k) +1. This means that log(log(N)) is the number of bits per node of an AVL tree.
2. The max integer overflow limit is 256, thus, if the height of a tree is 256, it will overflow, since log(256) is 2^8 is 256 which overflows itself.