CSCE 221 Cover Page
Program Assignment # 2

First Name: Rong Last Name: Xu UIN: 928009312

User Name: Abby-xu E-mail address: rongx0915@tamu.edu

Please list all sources in the table below including web pages which you used to solve or implement the current homework. If you fail to cite sources you can get a lower number of points or even zero, read more in the Aggie Honor System Office http://aggiehonor.tamu.edu/

Type of sources					
People					
Web pages (provide URL)					
Printed material					
Other Sources					

I certify that I have listed all the sources that I used to develop the solutions/code to the submitted work.

"*On my honor as an Aggie, I have neither given nor received any unauthorized help on this academic work.*"

Your Name (signature) Rong Xu Date 2020/10/020 The Pro-

gramming Assignment Report Instructions
CSCE 221

1. The description of an assignment problem.

   (a) The objective of this assignment is to create a Binary Search Tree. In doing this students learn how to create a Binary Search Tree, learn about recursive functions, and learn how to implement recursive functions to do various tasks. This program includes the functions for creating a Ninary Search Tree, calculationg a average search time for each node, inserting and removing a node, and printing the tree out.

2. The description of data structures and algorithms used to solve the problem.

   (a) Provide definitions of data structures by using Abstract Data Types (ADTs)

      i. The data structure I used in this assignment is Binary Tree. During the implementation of the tree, I degined a struct to defined each binary node, it has two pointers point to the ledt and right. Also, each node has a integer value and a search time. The BinarySearchTree class is a friend class of the BinaryNode, which has all the functions that are called in the main function.

   (b) Write about the ADTs implementation in C++.

      i. We implement the Binary Tree. So for the binary node we define a struct called Node. Each Node stores two integers which are the value and the search time related to the height, and two pointers that are point to the left child noe and right child node.

      ii. For the binary dearch tree, for the private members we have a node pointer which is the root of the tree, and the size of the tree itself, which is how many nodes the tree has.

   (c) Describe algorithms used to solve the problem.

      i. Binary Tree:
         A. delete tree
         B. insert
         C. search
         D. inorder traversal
         E. print level by level
         F. copy/move constructor
         G. copy/move operator
         H. output operator

   (d) Analyze the algorithms according to assignment requirements.

      i. Binary Search Tree
         A. insert O(log(n)): insert a node into the tree by comparing the value, create a new node, change the pointer of parent node to the node inderted.
         B. search O(log(n)):
         C. O(n) for worst case(linear tree)
         D. O(log(n)) for best case(perfect tree)
         E. inorder traversal
         F. print level by level
         G. copy/move constructor O(n): this function will copy every element in another list
         H. copy/move assignment O(n): the function will clear the linked lisit first(O(n)). Then copy the whole list from the other list (O(n)). O(n) + O(n) = O(n)
         I. output operator O(n): this function will print out every nodes content in the linked list

3. A description of the implementation of (a) individual search cost and (b) average search cost. Analyze the time complexity of the functions that (a) calculatie the individual search cost and (b) sum up the search costs over all the nodes

   (a) individual search cost

      i. Implementation: The indivisual search cost of each node was calculated in the insert_node function. Insert+node is implemented by recursion. If the function does not find the right place to enter the node it calls itself either its left or right child. Every time it is called the function, it will keep stracking of the previous nodes search time.

      ii. Time complexity: The time complexity of calculating the individual search time would be O(log(n)) where the n is the number of nodes in the tree at that point. This is because, on average, about half of the nodes will be visited for a node to be inserted into the tree and since the search time is calculated during insertion, it will have the same big-o as the insertion funciton.

   (b) average search cost

      i. Implementation: The average search time has two part. The size and the total search time of the tree. To calculate the size, the size will plus 1 everytime a node was inserted. The total search time is calculated in the inorder function. Every time it output a node, it also access the nodes search time. We have a private function called get_total_Search_time which allowed us to get the total search time of all nodes. Then we can use it divided by the size.

   (c) Upadated search time

      i. The time complexity of updating the search time would be O(log(n)) where n is the number of nodes in the tree at that point. This is about hald of the nodes have to be visited for a node to be (inserted/ deleted) to/from teh tree.

   (d) sum up the search cost overall

      i. Time complexity: O(n) where n is the number of nodes in the tree at taht point. Every nodes must be visited inorder to sum up all search time.

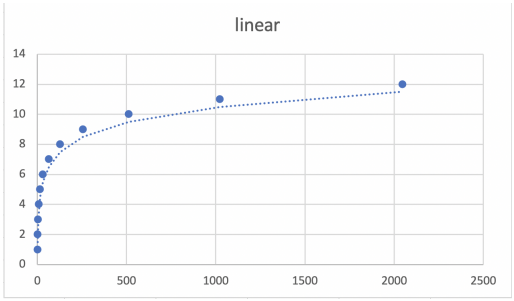
   (e) If we let be the infividual search time of each node of a ninary tree, then the best and average case for a binary tree would be O(log(n)). These cases are teh results from a perfect tree or random binary tree. The worst case for a binary tree would be O(n), which comes with a linear tree. For the perfect tree and random tree, the height is log(n), on each level there are 2^k nodes, and the total search time is (log1+1)+2(log2+1)+2^2(log3+1)+...+2^(log(n+1))-1(log(n)+1)=(n+1)log(n+1)-n. This means that the average time is O(log(n)). For the linear tree, the height is n-1 and the total search time is n(n+1)/2. Dividing these we have (n+2)/2 which is O(n).

4. Give an individual search cost in terms of n using big-O notation. Analyze and give the average search costs of a perfect binary tree and a linear binary tree using big-O notation, assuming that the following formulas are true (n denotes the total number of input data). To be clear, part 3 asks you to analyze the running time of the functions implemented to compute the individual and average search cost, while here you must analyze the asymptotic behavior of the values of the total and average search cost itself.

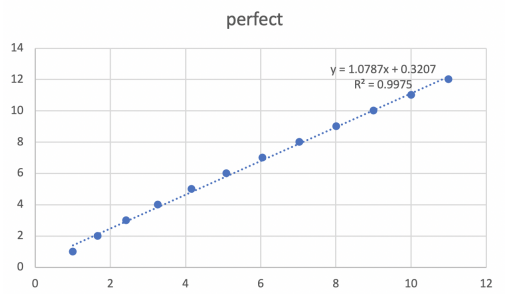
   (a)

	A	B	C	D
1	Node	linear	perfect	random
2	1	1	1	1
3	2	2	1.6667	1.6667
4	3	4	2.42857	2.71429
5	4	8	3.26667	3.73333
6	5	16	4.16129	6.3871
7	6	32	5.09524	7.6667
8	7	64	6.05512	7.59055
9	8	128	7.03137	9.06667
10	9	256	8.01761	10.3033
11	10	512	9.00978	12.2463
12	11	1024	10.0054	12.3972
13	12	2048	11.0029	14.0237

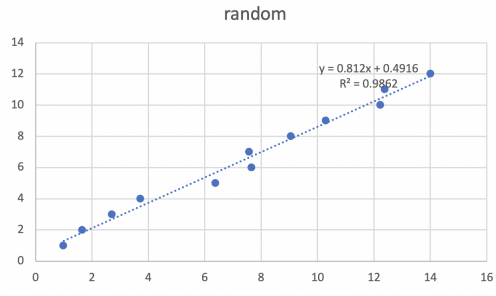
(b)



(c)



(d)



(e) After analyzing the tables and graphs we see that the theoretical results in part 4 match the results shown here, Analyzing both the Perfect and Random trees, we see that they are both O(log(n)) which is what we found in part 3. Analyzing teh linear tree, we see that teh average case for the linear tree is 2n which is O(n) which is also what we found in the part3.

5. C++ organization and implementation of the problem solution

(a) Provide a list and description of classes or interfaces used by a program such as classes used to implement the data structures or exceptions.

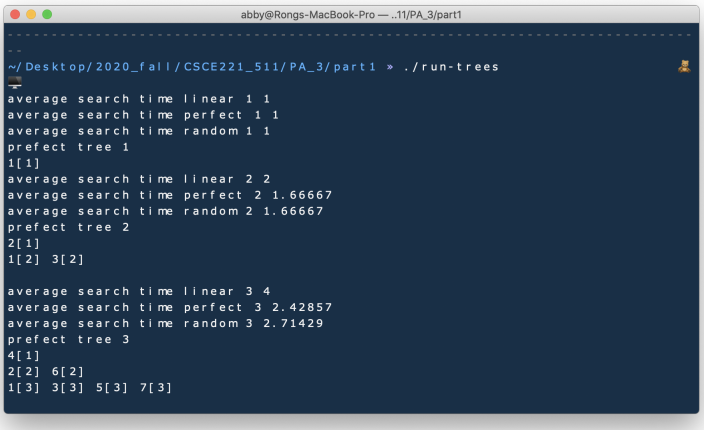
```
struct Node {
        int value;
        Node* left;
        Node* right;
        int search_time; //see the problem description

        // Node constructor
        Node(int val = 0,Node* l = nullptr,Node* r = nullptr):value(val),
        left(l), right(r), search_time(0) {}
};


struct BSTree {
public:
// constructor
BSTree():size(0),root(nullptr){};

//copy constructor
BSTree(const BSTree& other);

// move constructor
BSTree(BSTree&& other);
```

```
// copy assignment
BSTree& operator=(const BSTree& other);

// move assignment
BSTree& operator=(BSTree&& other);

//destructor
~BSTree();

const Node* get_root() const { return root; }
const int get_size() const { return size; }

// insert a node. with helper function
Node* insert(int obj);
Node* insert_helper(int obj, Node*&node, int st);

// search a node with a value, return the opinter. with helper function
Node* search(int obj);
Node* search_helper(int obj, Node* node);

// update the search time for every node, get average search time of a tree. with helper function
void update_search_times();
void update_st_helper(Node* node, int st);
float get_average_search_time();

// inorder traversal. with helper function
void inorder(std::ostream& out);
void inorder_helper(Node* node, std::ostream& out);

//void print_tree(std::ostream &out, Node* node);

// print a tree by level by level. with helper function print_node for print each node out
void print_node(std::ostream &out, Node* node);
void print_level_by_level(std::ostream& out);
// helper function for get the height of tree
int getHeight() {return (root == NULL) ? 0 : this->height(root);};
int height(Node* t);

private:
int size;
Node* root;
 int get_total_search_time(Node* node);
// you can add recursive helper functions to help you
// this is one is implemented for you:
void copy_helper(Node*& newNode,
const Node* sourceNode);
void deleteTree(Node* root);
};
```

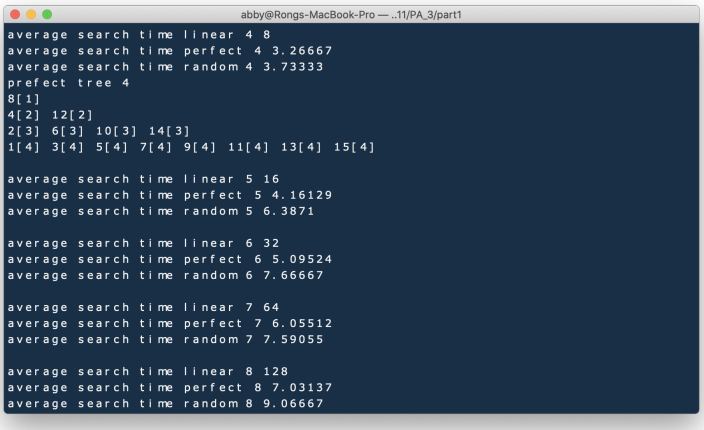
(b) Include in the report the class declarations from a header file (.h) and their implementation from a source file (.cpp).

   i. See the mimir submission.

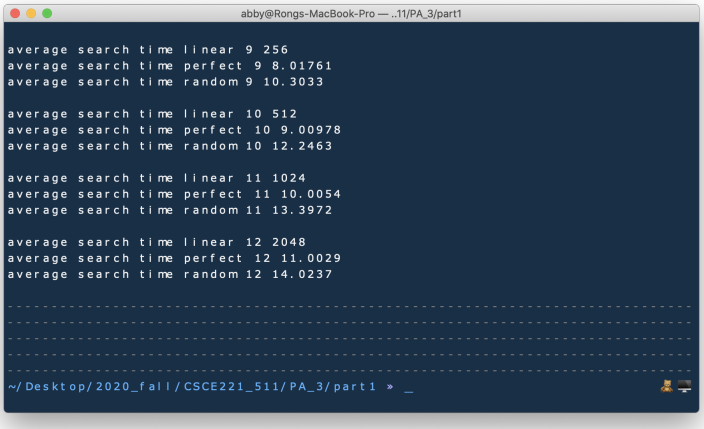
(d) Provide features of the C++ programming paradigms like Inheritance or Polymorphism in case of object oriented programming, or Templates in the case of generic programming used in your implementation.



(e)



(f)



(g)

6. A user guide description how to navigate your program with the instructions how to:

(a) compile the program: specify the directory and file names, etc.

```
all: run-trees

run-trees: BSTree.o BSTree_main.o
        c++ -g -std=c++11 BSTree.o BSTree_main.o -o run-trees

BSTree.o: BSTree.cpp BSTree.h
        c++ -g -std=c++11 -c BSTree.cpp

BSTree_main.o: BSTree_main.cpp BSTree.h
        c++ -g -std=c++11 -c BSTree_main.cpp
clean:
        rm *.o run-trees
```

(b) run the program: specify the name of an executable file.

```
» cd ./PA_3/part1/
» make clean
» make
» ./run-trees
```

7. Specifications and description of input and output formats and files

   (a) The type of files: keyboard, text files, etc (if applicable).

      i. Input file: a list of the value of each nodes in a file

   (b) A file input format: when a program requires a sequence of input items, specify the number of items per line or a line termination. Provide a sample of a required input format.

      i. Input file: input file should be an .txt file which have the format like this:

         A. Test file for "4p"

         B.
```
8
12
14
15
13
10
11
9
4
6
7
5
2
3
1
```

   (c) Discuss possible cases when your program could crash because of incorrect input (a wrong file name, strings instead of a number, or such cases when the program expects 10 items to read and it finds only 9.)

      i. none.

      ii. no special case.

8. Provide types of exceptions and their purpose in your program.

   (a) logical exceptions (such as deletion of an item from an empty container, etc.).

      i. No logical error has been found in testing

   (b) runtime exception (such as division by 0, etc.)

      i. Empty tree will be excuted in a different way

9. Test your program for correctness using valid, invalid, and random inputs (e.g., insertion of an item at the beginning, at the end, or at a random place into a sorted vector). Include evidence of your testing, such as an output file or screen shots with an input and the corresponding output

   (a) no implementation of invalid input.

   (b) All tests passes.