CSCE 221

1. The description of an assignment problem
   The purpose of this assignment was to create a binary search tree that could handle linear, perfect and random values and display the search cost for each number inserted. This was to be accomplished using vectors, pointers or queues. All trees with values lower than 2^4 was to output a printed tree with an 'X' character in place of an empty node.

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)
   void MakeNode//Pushes the first element into the binary search tree vector.
   void setleft//Places the element into the corresponding vector space that is left of the last element.
   void setright//Places the element into the corresponding vector space that is right of the last element.
   void Insert//Inserts the next element into its proper location in the vector.
   void Print//Prints out the key, search cost, total number of nodes, and average search cost.
   void PrintTree//Prints out a physical tree with the character 'X' in place of blank nodes.
   void Remove//Asks the user for the key of the node that they would like to remove and reprints the information and tree of the new tree with the node removed.

   (b) Write about the ADTs implementation in C++.
   void MakeNode//Places the first element into location zero of v1 and a searchcost of 1 into location zero of v2.
   void setleft//Uses the algorithm 2*index + 1 to place the element in that specific location of the vector, it also changes the highest index if the current index is larger than the highestindex and sets the height of the left side of the tree using the algorithm int((log(currIndex + 1)/log(2)) + 1).
   void setright//Uses the algorithm 2*index + 2 to place the element in that specific location of the vector, it also changes the highest index if the current index is larger than the highestindex and sets the height of the right side of the tree using the algorithm int((log(currIndex + 1)/log(2)) + 1).
   void Insert//Uses if statements to check to see if the current element is higher or lower than the last element scanned on either the right or left side of the tree and uses the functions setright and setleft accordingly. The function also sets all the search costs for the empty nodes using a for loop at the end of the function.
   void Print//Prints out the keys and search costs in v1 and v2 using a for loop, and skips over any blank nodes using if statements. Everytime the element in v1 is not equal to zero, one is added to the sum of the total nodes and the according search cost is added to the total search cost. The total search cost is divided by the sum of the nodes and both values are printed in the console.
   void PrintTree//Uses a doubly nested for loop to scan through v1 and v2 to print out an 'X' when v1 is equal to zero and prints the key when v1 is not equal to zero. everytime the search cost changes, a new line is started.
   void Remove//Asks the user for the key of the node that they would like to remove and reparses through the file, skipping over that number to create a new tree that does not include the given number, placing the new tree in two new vectors and setting them equal to v1 and v2 after parsing through the file.

(c) Describe algorithms used to solve the problem.
There are three main algorithms in this assignment, finding the individual search cost, average search cost and finding the updated search cost.
Individual Search Cost-The formula int((log(currIndex + 1)/log(2)) + 1) calculates the "height" of the left or right side of the tree that the element is being inserted in. This formula is derived from the max amount of nodes that can be on each row exponentially increasing by two.

Average Search Cost-To find the average search cost, the vector is put into a loop and the search cost for all non-empty locations in the vector are summed up and divided by the total number of nodes. The loop is as follows.

```
for (int i = 0; i <= highestindex; i++)
if(v1[i] != 0)
sum = sum + v2[i];//Sum up all the search costs of the non-blank nodes

double average = sum / size;
```

Updated Search Cost-The updated search cost is calculated by using the same algorithms for the individual and average search costs. The search cost is "updated" since the updated binary seach tree does not contain the "removed" key.

All of the search costs are based on the insert function.

(d) Analyze the algorithms according to assignment requirements.
The time complexity of the search costs are as follows (h is the height of the binary tree).
Individual Search Cost-O(h) since the calculation does not contain any loops.
Average Search Cost-O(nh) since the calculation is contained within one loop.
Updated Search Cost-O(h) since the calculatino is contained within one loop.
Perfect Binary Search Tree-O(logn)
Linear Binary Search Tree-O(n)

3. A 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.
   The program was made using two separate STD vectors that corresponded to the location of the key and the location of the specific search cost.

   (b) Include in the report the class declarations from a header file (.h) and their implementation from a source file (.cpp).
   On top of using the STD vector class, the math.h header file was included in the program to use logs in order to calculate the correct indexes corresponding to hwo the binary search tree was formulated. The fstream was included as well in order to parse through the files given to us via the student web page. The files gave us the numbers in a specific order to create perfect, linear and random binary trees.

   (c) 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.
   The two main paradigms used in the assignment are templates and polymorphism
   Polymorphism-Since vectors are able to hold different forms of data, it is inferred that polymorphism is applicable to the code since the vector class morphs to the data type the vector or list is initialized with.
   Templates-Furthermore, it is inferred that templating will be used when using vectors. Vectors must be initialized with a type, and since the numbers for the keys do not get relatively large in the assignment, the vector was initialized with the int type.

4. 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.
   In order to compile the program, simply change the directory to "Kaiser-Alexander-A4" and compile the file "main.cpp" using the command g++ main.cpp. Make sure that the files of numbers for the binary search tree are in the same directory as the compliation.

   (b) run the program: specify the name of an executable file.
   After compiling using g++, run the program by using the command ./a.out.

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

   (a) The type of files: keyboard, text files, etc (if applicable).
   The program inputs data from a given file within the same directory that contains the organized data to print a perfect, random or linear binary search tree using getline and outputs the data onto the console in PuTTY.

   (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.
   The file input uses getline to parse through a given file to create a binary search tree based on the order that the numbers are received. There is one number per line and the getline function runs through the complete file until it runs into a

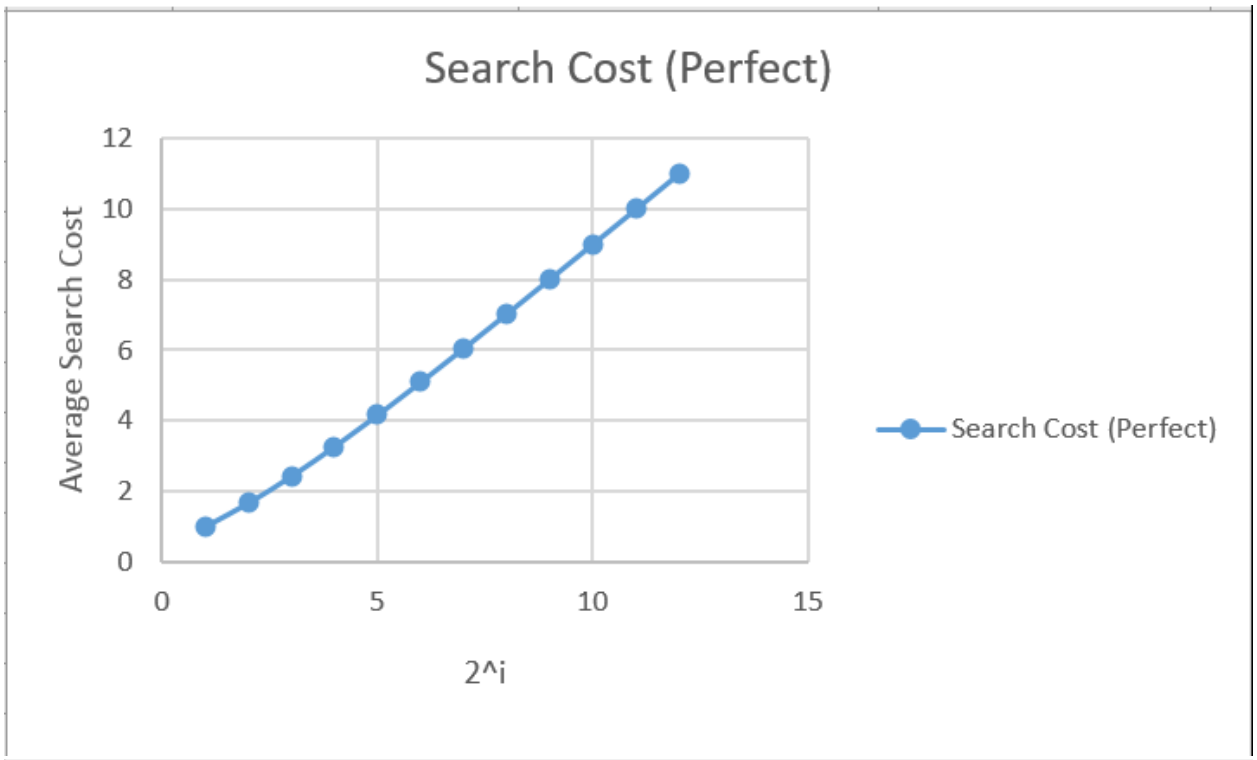
   (c) Discuss possible cases when your program could crash because of incorrect input (a wrong file name, strings instead of a number, or such cases when the program expects 10 items to read and it finds only 9.)
   There are many instances that could crash the program. For a wrong file name, the program would simply exit if the file name given is not listed in the same directory as the file. However, the problem comes when outputting the linear trees. Due to the limitation with vector sizes, any number past 27 will cause the program to segment fault specifically for linear trees. This is due to the fact that the assignment files contain right sided linear trees rather than left. The empty nodes grow exponentially with each increase in search cost to where the STD vector simply cannot be initialized with a value big enough to hold a linear tree of this type.

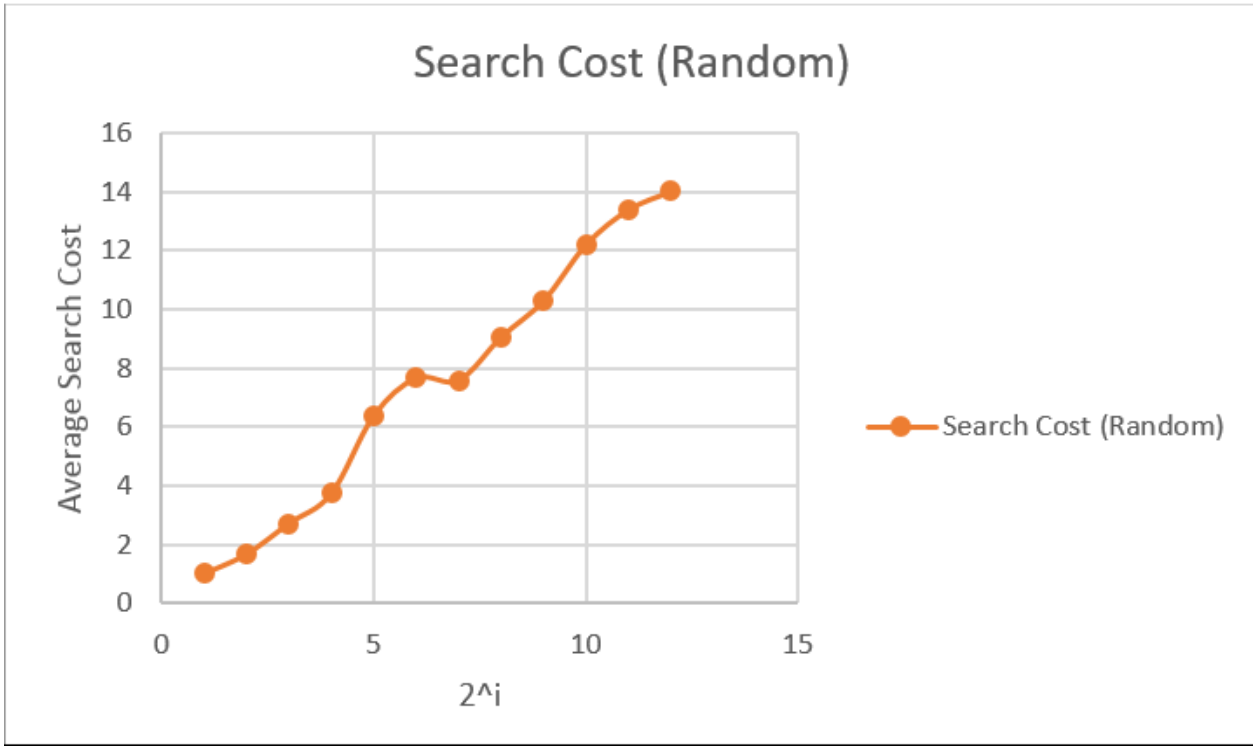
6. Provide types of exceptions and their purpose in your program.

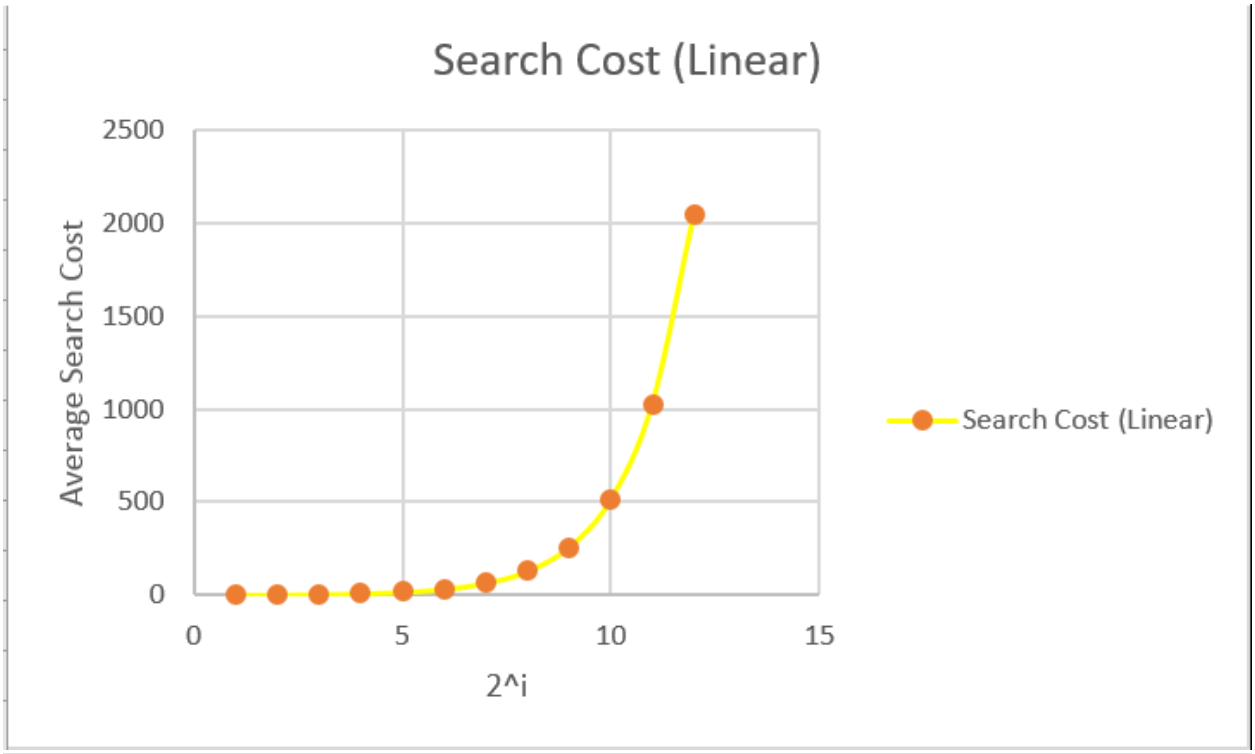
   (a) logical exceptions (such as deletion of an item from an empty container, etc.).
   In order to eliminate logical exceptions in the program, if statements are used to ensure that the data is being handled responsibly. For example, if the file that is listed is not within the same directory as a program, "No file" will be output and the program will exit. However, the issue listed in problem 5c is unavoidable for all linear trees past 2^4.

   (b) runtime exception (such as division by 0, etc.)
   To prevent runtime exception, a try-catch loop was implemented in the main function. When a runtime exception is encountered (such as a segmentation fault) the program is terminated completely and a reason is given as to why the termination occurred.

7. 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.
The first section of testing includes the given graphs and tables that are required by the report instructions







2^i	Search Cost (Perfect)	Search Cost(Random)	Search Cost (Linear)
1	1	1	1
2	1.667	1.667	2
3	2.429	2.714	4
4	3.267	3.733	8
5	4.161	6.387	16
6	5.095	7.667	32
7	6.055	7.591	64
8	7.031	9.067	128
9	8.018	10.303	256
10	9.01	12.216	512
11	10.005	13.397	1028
12	11.003	14.024	2048

The data in the graphs supports the big-o notation given in problem 2d. The perfect binary tree has a significantly smaller search cost than the linear binary tree.