The Programming Assignment Report Instructions
CSCE 221

1. The description of an assignment problem.
   The purpose of this assignment was to implement an ADT Skip List that could accept values up to 2^{12} in a perfect and random manner. The average insertion cost and deletion cost was to be calculated and for all files and the skip list was to be printed as well as the comparison cost for each insertion and deletion was to be displayed for all files smaller than or equal to 2^4.

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 newLevel//Adds a new level to the skip list.
   void insertFirstRow//Adds a number to the bottom row of the skip list.
   void insertTopRow//Adds a number to one of the top rows of the skip list.
   int insertComparisons//Finds number of comparisons for each insert operation.
   int find//Finds the node within the skiplist and returns the comparisons for deletion.
   bool searchRows//Checks to see if the number is in the doubly linked list.
   int deleteRows//Deletes the number from the skiplist one by one whne the number is plugged into the function.

   DListNode Class
   int getElem//Returns the number the node contains.
   DListNode*getNext//Returns what the next object the node is pointing to.
   DListNode*getPrev//Returns what the previous object the node is pointing to.
   DListNode*insert_before//Inserts a new node before the node the function is being called with.
   DListNode*insert_after//Inserts a new node after the node the function is being called with.
   void delete_before//Deletes a node before the node the function is being called with.
   void delete_after//Deletes a node after the node the function is being called with.

   DoublyLinkedList Class
   DListNode*getFirst//Returns the node that is in front of the header of the doubly linked list.
   DListNode*getAfterLast//Returns the trailer of the doubly linked list.
   bool isEmpty//Checks to see if the doubly linked list is empty.
   int first//Returns the number of the node after the header.
   int last//Returns the number of hte node before the trailer.
   void insertFirst//Inserts a node after the header and before the first node.
   void insertLast//Inserts a node before the trailer and after the last node.
   int removeFirst//Removes the node after the header.
   int removeLast//Removes the node before the trailer.

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

void newLevel//Creates a new doubly linked list and adds the "key" to the first position of the doubly linked list using the function insertFirst, then pushes the doubly linked list onto the vector of doubly linked list.
void insertFirstRow//Sorts where the the number should go within the first row of the doubly linked list using a while statement to parse through the doubly linked list. The node is then inserted using the insert_before function.
void insertTopRow//Uses a random function to simulate a coin flip and if it lands heads (1) then another node is added on top of the first row of the skip list. If the max size of the height of the skip list is exceeded then the newLevel function is called and another doubly linked list is pushed onto the main vector.
int insertComparisons//Uses skip list ADT to parse through each section of the vector and when the number is smaller than the "key" and when it reaches a number that is greater than the key it drops down the the vector position (next row down of skip list) until the correct position is reached.
int find//Parses though the skip list starting at the largest position in the vector at negative infinity and drops down when the next element is larger than the "key" using the getNext function. When the key is reached the loop is broken and the comaprisons are returned.
bool searchRows//Parses thorough the doubly linked list in the function definition using a while loop and returns a boolean value corresponding whether or not any of the nodes contain the "key".
int deleteRows//Parses through each element of the vector and deletes every instance of the "key". If the row becomes empty then the doubly linked list is deleted and the vector position is deleted.

DListNode Class
int getElem//Returns object contained within the node using a return statement.
DListNode*getNext//Returns a pointer to the next node that is in the doubly linked list.
DListNode*getPrev//Returns a pointer to the previous node that is in the doubly linked list.
DListNode*insert_before//Inserts a new node with a corresponding "key" before the node that is pointing to the function. The next and previous pointers are rewired to include the new node.
DListNode*insert_after//Inserts a new node with a corresponding "key" after the node that is pointing to the function. The next and previous pointers are rewired to include the new node.
void delete_before//Deletes a node with a before the node that is pointing to the function. The next and previous pointers are rewired to include the new node.
void delete_after//Deletes a node with a after the node that is pointing to the function. The next and previous pointers are rewired to include the new node.

DoublyLinkedList Class
DListNode*getFirst//Returns a pointer to the first node in the list that is ahead of the header.
DListNode*getAfterLast//Returns a pointer to the trailer of the doubly linked list.
bool isEmpty//Checks to see if the header and trailer are the only two items in the list pointing to one another and returns a boolean value.
int first//Returns the object that the first node is pointing to after the header.
int last//Returns the object that the last node is pointing to before the trailer.
void insertFirst//Inserts a new node with a corresponding "key" after the header. The next and previous pointers of the previous node in that location and the header are rewired to include the new node.
void insertLast//Inserts a new node with a corresponding "key" before the trailer. The next and previous pointers of the previous node in that location and the trailer are rewired to include the new node.
int removeFirst//Deletes a node after the header. The next and previous pointers of the previous node in that location and the header are rewired to include the lack of a new node.
int removeLast//Deletes a node before the trailer. The next and previous pointers of the previous node in that location and the trailer are rewired to include the lack of a new node.

(c) Describe algorithms used to solve the problem.
There are five main algorithms in the assignment, the comparison cost of insertion, the comparison cost of search, the comparison cost of deletion and the average cost of insertion and deletion.

Cost of Insertion-a series of while loops parse each level of the doubly linked list vector adding a comparison each time the iterator moves right or drops down. When the iterator hits an number greater than the "key" on the bottom row the number of comparisons is returned.
Cost of Deletion-a series of while loops parse each level of the doubly linked list vector adding a comparison each time the iterator moves right or drops down. When the iterator hits the key, the function terminates regardless of the level the skip list is on.
Cost of Search-a series of while loops parse each level of the doubly linked list vector adding a comparison each time the iterator moves right or drops down. When the iterator hits the key, the function terminates regardless of the level the skip list is on.
Average of Insersion and Deletion-every time a number of comparisons is made, the number is added to a vector in a location correspoding the the number the program is attempting to find. The vector is then totaled up and divided by the total number of elements in the vector to find the averages.

(d) Analyze the algorithms according to assignment requirements.
Cost of Insertion-using the insertComparisons function, each level of the doubly linked list vector is parsed until the node comes into contact with a node that is greater than the key needing to be inserted. Every time the node moves right one is added to the variable of comparisons. When a node greater than the key is come into contact, the number that the node stops on is saved in the variable elementholder and the vector space one below the previous vector space is parsed until it reaches the value of elementholder. When dropping down a level in the skip list another comparison is added as well. This process is repeated until the node tries to drop down beyond the last level.
Cost of Deletion-the concept is the same as the cost of insertion except rather than going to the very bottom of the skip list to finish with comparisons, the function terminates and the comparisons are returned when the while loop returns a node that contains the key the function is looking for.
Cost of Search-the concept is the same as the cost of insertion except rather than going to the very bottom of the skip list to finish with comparisons, the function terminates and the comparisons are returned when the while loop returns a node that contains the key the function is looking for.
Average of Insertion and Deletion-these figures take all of the values of comparisons for insertion and deletion and averages them to display them at the end. This is done very simply using for loops.

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 used a series of vectors that contained integers and doubly linked lists. The doubly linked list contained nodes that held integers.

   Vectors-The vector of doubly linked lists was contained each row of the skip list where a row contained one doubly linked list. Standard integer vectors contained the values of the numbers that were input, the number of comparisons of deletion and the number of comparisons of insertion.
   Doubly Linked List-Each row of the skip list contained a doubly linked list with a header of negative infinity and a trailer of positive infinity. Each node of the doubly linked list contained the integers of the skip list.

   (b) Include in the report the class declarations from a header file (.h) and their implementation from a source file (.cpp).
   On top of including the STD vector class and the "DoublyLinkedList.cpp" file, ctime and limits were included as well. Ctime was used to propagate the random values for the coin flips and limits was used to implement the header and trailer of the doubly linked lists as positive and negative infinity.

   (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.
   This assignment focused on the paradigms of polymorphism and encapsulation.

   Polymorphism-The assignment would have not been possible if it were not for the capabilities of the vector class. The two forms of the STD vector was one that held doubly linked lists and one that held integers. The ability that the vector class can adapt to the variable type put into it.
   Encapuslation-The assignment used pointers to the next and previous nodes that were private within the DListNode class. This allowed for that data not to be changed using an outside class, inevitably skewing the pointers.

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.
   Compile the program in the Kaiser-Alexander-A5 using the command g++ SkipList2.cpp. Make sure the number files, DoublyLinkedList.cpp and DoublyLinkedList.h files are in the same directory as well.

   (b) run the program: specify the name of an executable file.
   Run the program after compiling the file using the method above 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 a file in given in the same directory as the program that contains numbers from 2^1 to 2^{12} in a perfect, random and linear manner and parses them in order in to the skip list.

   (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 program uses getline in order to parse throught the files. There is one item per line and getline terminates when it runs out of integers to parse through. The input format is as follows.

   string file = "4r";

   Place the name of the file within the quotation marks that the user would like to parse through.

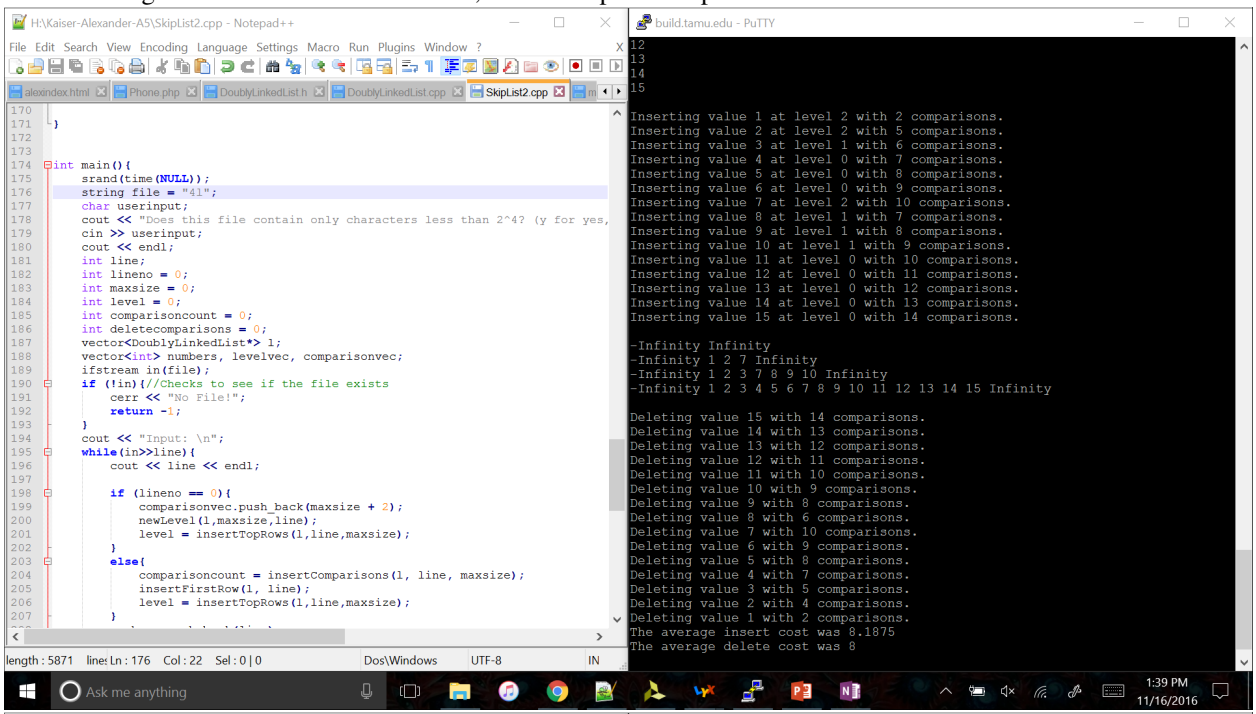
   (c) Discuss possible cases when your program could crash because of incorrect input (a wrong file name, strings instead of a number, or such cases when the program expects 10 items to read and it finds only 9.)
   The only time that the file could crash would involve a misplacement of files in the directory. If the file that is contained within the variable "file" is not in the same directory as the program, the program will not run and an error will be thrown. All other cases of crashing are accounted for with exceptions.

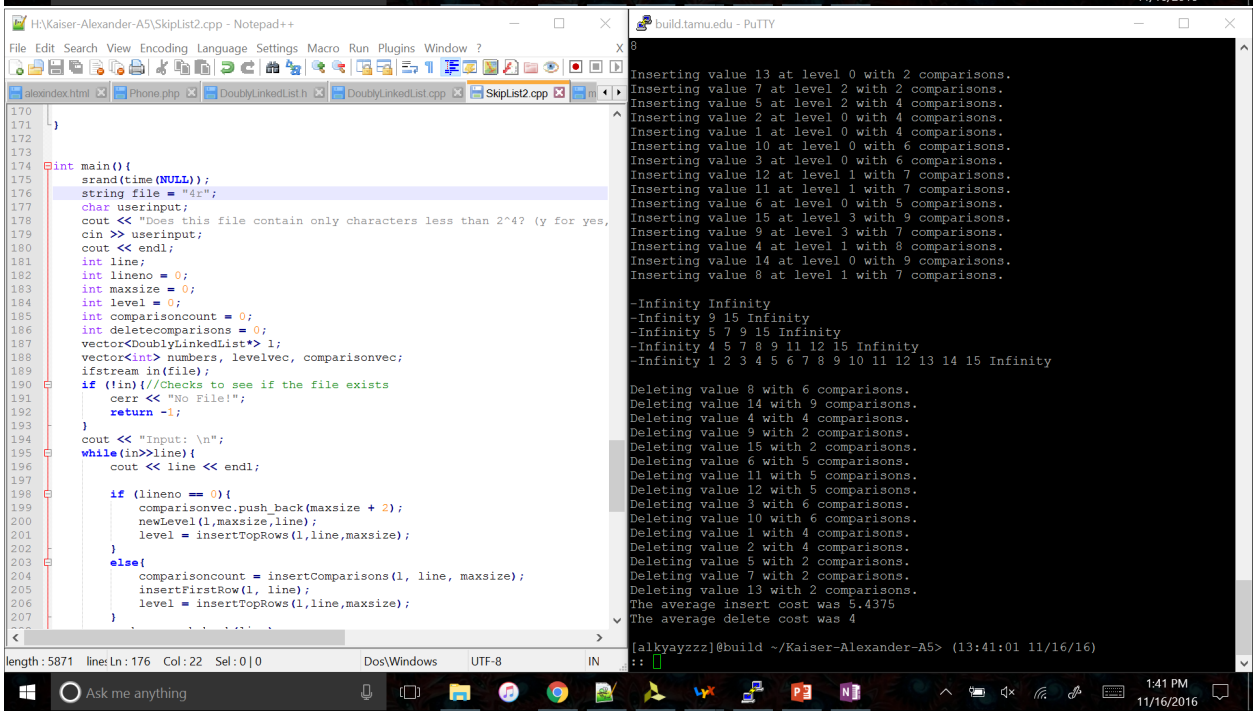
6. Provide types of exceptions and their purpose in your program.

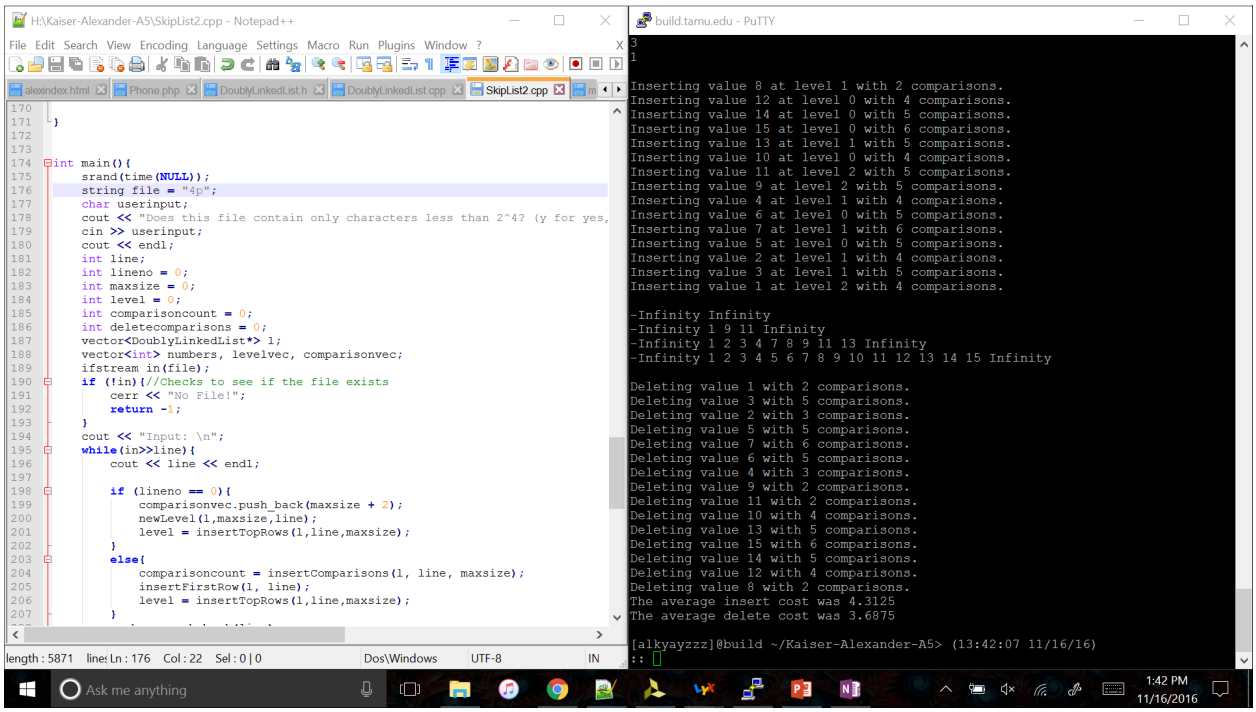
   (a) logical exceptions (such as deletion of an item from an empty container, etc.).
   In order to eliminate logical exceptions in the program, if statements are used to ensure that the data is being handled responsibly. For example, if the file that is listed is not within the same directory as a program, "No file" will be output and the program will exit.

   (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.

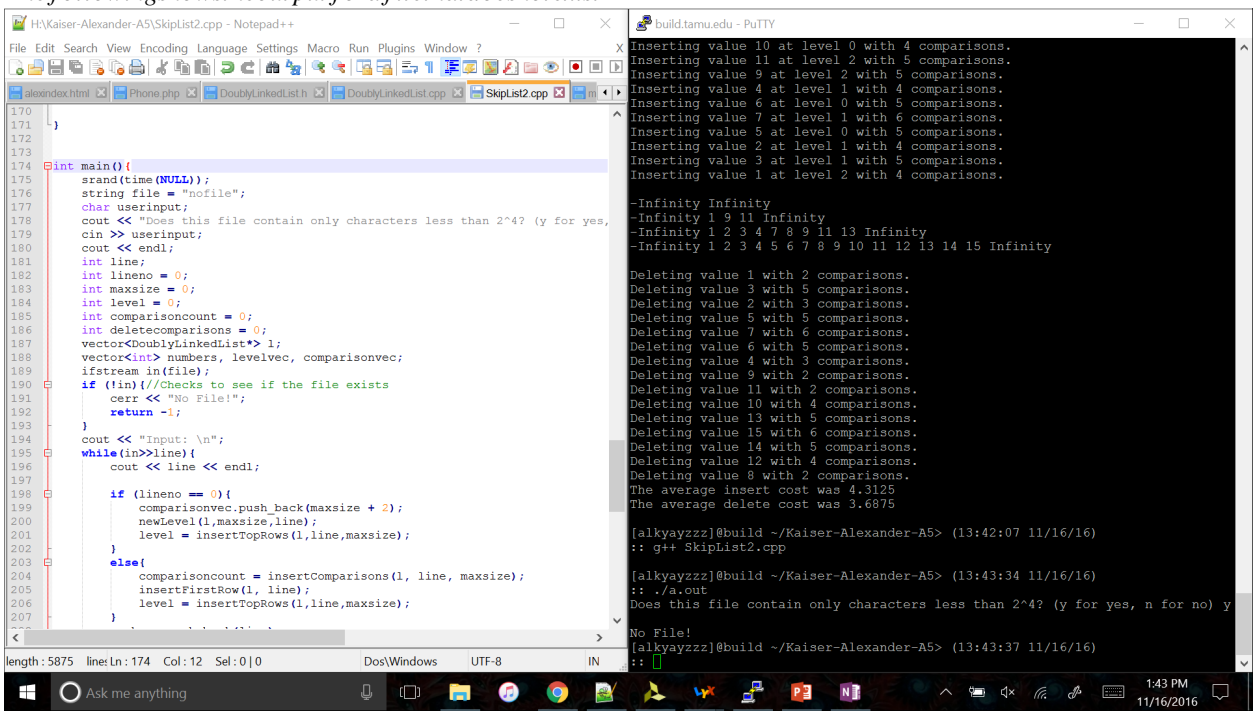
7. Test your program for correctness using valid, invalid, and random inputs (e.g., insertion of an item at the beginning, at the end, or at a random place into a sorted vector). Include evidence of your testing, such as an output file or screen shots with an input and the corresponding output.
The following shows test trials for a random, linear and perfect input of 2^4







The following shows the output for a file that does not exist.



8. Additional Questions.

1.Describe the node structure utilized in the implementation of your skip list

The node structure used in this implementation of a skip list is the doubly linked list from assignment three part 1. It was an easy structure to use with the skip list to insert and delete nodes of integers. The header and trailer was a useful addition to hold the negative and positive infinities of the skip list as well. The code of the doubly linked list was modified to be able to insert nodes anywhere within the doubly linked list using the functions insert_before, insert_after, delete_before and delete_after.

2.Describe the overall data structure of your skip list

The skip list was made using a vector of doubly linked lists where each row of the skip list corresponded to a separate doubly linked list. The higher the element of the vector, the farther away from the bottom row the skip list grew. The doubly linked list allowed the nodes to point to the previous and next nodes that surrounded it that allowed for easy traversal of the rows. The "coin flips" were accomplished by using a random function that returned a 1 or 0 value. If the value was 1 then the coin flip was output as heads. The data was sorted using an algorithm of while loops. A list of all functions used in the implementation of the skip list are contained in item 2 of this report.

3.Explain how the nodes are used in the structure of your skip list

Each node in the doubly linked list corresponds to a number in the skip list. The function getElem returned the object that the node pointed to in order to extract the numbers to make the skip list. The header and trailer of the doubly linked lists were used to hold the values of negative and positive infinity. Each node pointed to the node before and in front of it and the doubly linked lists were stacked on top of one another using the vector of linked lists.

4.Describe how levels of your skip list are traversed (up and down)

The traversal of the skip list was accomplished using a series of while loops. The parsing started at the top left corner and the doubly linked list was parsed until it came accross a node that contained a number that is greater than or equal to the "key" that was being looked for. If the number is equal to the key the while loop is broken out of and the function stops, however if the number is greater than the key, the last number parsed is held in a variable and the next row of the skip list (the doubly linked list in the element location one below the original element location) is parsed up until this variable. This is essentially a "drop down" to the next row of the skip list and the process is repeated until the key is found or until the function tries to drop down below the bottom row of the skip list. When inserting, the coin flip operation checks to see if a new row needs to be made, and makes one if need be by pushing a new doubly linked list onto the main vector.

5.A description of how you implemented the calculation of (a) insert cost (b) search cost (c) delete cost, and worst case and average case theoretical runtimes (Big O) for the insert, search, and delete functions

See item two parts c and d for the answer to this question.
For all three operations, the average case is $O(logn)$ and the worse case is $O(n)$.

6.Run your program several times with the same set of data. Is the number of comparisons for the search operation the same each time? Why or why not? Provide justification.

The number of comparisons is not the same each time because the height of the skip list is randomized by the coin flips. Since a skip list is a randomized data structure, then the number of comparisons cannot be held constant for numerous trials.

7.How likely is it that an item will be inserted into the nth level of the skip list?

The liklihood that an item will be inserted into the nth level is $1/2^i$ which means that the expected size of the skip list is $n/2^i$ or $O(n)$.

8.How does height affect the number of comparisons for the skip list operations?

The higher the skip list, the greater the number of comparisons for the numbers on the lower rows.

9.Does the order of the data (sorted, reverse sorted, random) affect the number of comparisons? How?

There is a small difference in comparisons between the linear and random/perfect input files. Since comparisons are based on the height of the skip list and how far away from the top left of the skip list the item is, this accounts for a small increase in comparisons when inputing a linear file. However, since skip lists use skip pointers, the comarisons vary little. A skip pointer allows for comparisons to be skipped depending on the height of the skip list. If the data was to be compared using a vector, the input of the data would play a large role in the number of comparisons per trial.

10.How does the runtime compare to a Binary Search Tree for the insert, search, and delete operations?

A binary tree's operations are $O(logn)$ for the average and $O(n)$ for the worst for all operations, so they are essentially the same.

11.In what cases might a Binary Search Tree be more efficient than a skiplist? In what cases might it be less efficient?

A binary search tree is less efficient when working with concurrent access/modification due to the fact it has to be rebalanced often. It is more efficient in any circumstance where the tree does not have to be rebalanced.