Data Structures
Homework #5

Due: Dec 19, 2023 (Before Class)

1. Design a variation of binary search for performing operation `findAll(`k`)` in a dictionary implemented with an ordered search table, and show that it runs in time $O(\log n + s)$, where n is the number of elements in the dictionary and s is the size of the iterator returned.

2. Suppose that each row of an $n \times n$ array A consists of 1's and 0's such that, in any row of A, all the 1's come before any 0's in that row. Assuming A is already in memory, describe a method running in $O(n \log n)$ time (not $O(n^2)$ time!) for counting the number of 1's in A.
   HINT: Think first about how you can determine the number of 1's in any row in $O(\log n)$ time.

3. Suppose we are given two ordered search tables S and T, each with n entries (with S and T being implemented with arrays). Describe an $O(\log^2 n)$-time algorithm for finding the kth smallest key in the union of the keys from S and T (assuming no duplicates).

   HINT: Do a "double" binary search.

4. Draw the 11-entry hash table that results from using the hash function, $h(i) = (2i + 5) \bmod 11$, to hash the keys 34, 22, 2, 88, 23, 72, 11, 39, 20, 16, and 5, assuming collisions are handled by the following approaches respectively.

   (a) *chaining.*

   (b) *linear probing.*

   (c) *double hashing* using the secondary hash function $h'(k) = 7 - (k \mod 7)$. Note that double hashing uses $(h(k) + j \times h'(k)) \mod N$, for $j = 1, 2, \ldots, N-1$, when collisions occurs.

5. Recall the skip list data structures we introduced in class where the concept of randomization is introduced. Now we consider a deterministic version and suppose that we use only two levels, *i.e.*, two linked lists, for n elements. Each element is in linked list S_0 and some elements are also in the other linked list S_1. Please show that the search cost can be minimized to O($\sqrt{n}$).

6. (50 pts) (**Programming Exercise**)
   This exercise is about to implement a ***heap*** by means of a ***linked structure*** with Python, in stead of using an array with level-numbering. A sample definition for the node and heap class is given on the course website. Each node has an entry with two attributes, key and name, where key is an integer representing the priority and name is a string. A smaller key has a higher priority. For the methods in the heap includes

   `removeMin()`: This is to remove the object with the minimum key from the heap;

   `Insert(v)`: This is to insert a heap node into the heap;

`upwardHeapify(v)` This method performs upward adjustment from the current node v;

`downwardHeapify(v)`: This method will adjust the heap downward from heap node v;

`printHeapPreOrder(v)`: For management or verification, we print the heap in pre-order from node v with this method.

The Python program starts with function `HeapwithEntriesInserted()` which reads the input file, `inFile.txt`. In the input file, each line contains only one entry: key and string and as follows:

```
10 mary
25 john
35 mars
50 lowe
```

An example of input file is also provided on the course website. The execution should be as below and corresponding list operations is posted on the website.

```
>>> HeapwithEntriesInserted()
Heap size= 4 The highest priority is 10
pre-order traversal:
Node [ 10 mary ]
Node [ 25 john ]
Leaf [ 50 lowe ]
Leaf [ 35 mars ]
deleteMin
deleteMin
deleteMin
deleteMin
The heap is now empty
deleteMin
The heap is empty and no entry can be removed
insert 35, resume
insert 15, second
insert 20, fourth
Heap size= 3 The highest priority is 15
pre-order traversal:
Node [ 15 second ]
Leaf [ 35 resume ]
Leaf [ 20 fourth ]
```