COMP2119 Introduction to Data Structures and Algorithms
Assignment 2 - Algorithm Analysis, Data Structures and Hashing

Due Date: Oct 21, 2024 7:00pm

Question 1 - What have You Done? [30%]

One day, handsome Jeff finished grading n assignment scripts. He sorted the scripts in ascending order of UID (assume all UIDs are unique). To inspect his work, Ben read k scripts from the top, and put them back at the bottom of the pile without changing the order of the k scripts.

Suppose the UID of the n scripts are stored in list A as integers with the current placing order. The first element in A represents UID of the script on the top. To arrange the papers back in order, Jeff needs to find out the value of k.

You may assume that n is in power of 2, and $0 \leq k \leq n-1$.

Example:
For input $A_1 = [43, 60, 71, 3, 14, 23, 26]$ and $n = 7$, the output will be $k = 4$. Ben has read scripts with UID 3, 14, 23, 26.

Your Tasks:

(a) [10%] Jeff has written the function reOrder1(A, n) to solve the problem. Explain briefly how his code solve the problem and analyze it's worst case time complexity.

Algorithm 1.1 Pseudocode of reOrder1

```
1:  function reOrder1(A, n)
2:
3:      k = 0
4:      for (i = 0 to n − 2) do
5:          if (A[i] > A[i + 1]) then
6:              k = n − i − 1
7:          end if
8:      end for
9:
10:     return k
11:
12: end function
```

(b) [15%] If n is large, it would be a nightmare to run Jeff's algorithm by human efforts. Write the pseudocode of another algorithm reOrder2(A, n) to solve the problem. The algorithm must be asymptotically faster than reOrder1(A, n).

Briefly explain what the code does, and analyze the algorithm's worst case time complexity.

Question 2 - Linked List [30%]

Implement a linked list and solve the following sub-problems with your implementation. Feel free to select any programming language you prefer.

(a) [15%] Merging: Implement a function that takes two sorted linked lists and merges them into a single sorted linked list.

(b) [15%] Cycle detection: A cycle in a linked list occurs when a node's next pointer points back to a previous node, creating a loop, while the beginning of a cycle is the first node in that loop where traversal starts repeating.

Implement a function that detects if there is a cycle in the linked list. If a cycle exists, it returns the node where the cycle begins.

Question 3 - Maximum Frequency Stack [20%]

Design a stack-like data structure to push elements to the stack and pop the most frequent element from the stack.

Specifically, for a *FreqStack* class: *FreqStack*() constructs an empty frequency stack (called *fstack*). *fstack.push*(*val*) pushes an integer *val* onto the top of the stack. *fstack.pop*() removes and returns the most frequent element in the stack. If there are multiple different elements of the same highest frequency, the element closest to the stack's top is removed and returned.

Suppose at most N elements will be inserted into *fstack* and each *val* is smaller than 10.

Example:

```
FreqStack fstack = new FreqStack();
fstack.push(5); // The stack is [5]
fstack.push(7); // The stack is [5,7]
fstack.push(5); // The stack is [5,7,5]
fstack.push(7); // The stack is [5,7,5,7]
fstack.push(4); // The stack is [5,7,5,7,4]
fstack.push(5); // The stack is [5,7,5,7,4,5]
fstack.pop(); // return 5, as 5 is the most frequent. The stack becomes [5,7,5,7,4].
fstack.pop(); // return 7, as 5 and 7 are the most frequent, but 7 is closest to the
    top. The stack becomes [5,7,5,4].
fstack.pop(); // return 5, as 5 is the most frequent. The stack becomes [5,7,4].
fstack.pop(); // return 4, as 4, 5 and 7 are the most frequent, but 4 is closest to
    the top. The stack becomes [5,7].
```

Your Tasks:

(a) [15%] Implement the *FreqStack* class as described above.

(b) [5%] Analyse the worst-case time complexity of *fstack.push* and *fstack.pop* of your implementation.

Question 4 - Hash Table [20%]

Insert the keys $\{17, 94, 86, 22, 98, 79, 54, 38\}$ into the following hash tables.

For each case, calculate the average number of slots inspected for an unsuccessful search after the keys are inserted (An empty slot or a NIL pointer access should also be counted as one inspection.). Assume that each slot in a hash table has equal chance to be accessed at the first probe in the search process.

(a) [5%] A hash table of size 7 with a single hash function of $h(x) = x \bmod 7$. Collisions are handled by chaining.

(b) [7%] A hash table of size 11 with a single hash function of $h(x) = x \bmod 11$. Collisions are handled by open addressing with linear probing, using the function $f(i) = 3i$.

(c) [8%] A hash table of size 11 with double hashing using the probe sequence $h(k, i) = (h_1(k) + i\ h_2(k)) \bmod 11$, where $h_1(x) = x \bmod 11$ and $h_2(x) = 2 - (k \bmod 2)$.

Submission

Please submit your assignment (one **PDF** file) to moodle by the deadline. Make sure the content is readable. Feel free to contact the TAs if you encounter any difficulty in this assignment. We are happy to help you!