ECS 60

Xie Zhou

ID: 912143385

Written Homework 2

1. Because there is 1 integer that is repeated, among the n indices (from 0 to n-1) in the array, there will be one empty spot if all integers are moved to the spots of the corresponding indices (index-1). Also, there will be a conflict over a spot between two same integers. Therefore, a fast algorithm could be: starting from the first item in the array, move the integer to the place with corresponding index -1 (eg. int 8 would be moved to the array[7] ). Store the number that was originally in that spot in a temporary array for later swapping. If the spot is occupied by the same number before swapping, then this number is the repeated number.
2. 0.5^n. The worst case occurs when all the blocks are of height 1. In this case the skip list would have a linear search. Because the probability for a block of height 1 to occur is 0.5, the probability for a worst case to occur when searching a skip list with n items is 0.5^n.
3. a. Move-to-front: accessing the first item

Transpose: accessing the first item

Count: the count change in the accessed element doesn’t change its overall “ranking”

Ordering: simply keeps the order in the list; doesn’t change anything in the list.

b. Move-to-front: accessing the last item

Transpose: no exhaustive search since swapping is between 2 elements

Count: accessing the last item in the list; all other items’ counts all become 1 less than the last item’s after the access.

Ordering: searching for the largest/smallest number when it’s in

ascending/descending order.

c. Move-to-front: O(N)

Transpose: O(1)

Count: O(1)

Ordering: O(logN) when searching/inserting

d. Move-to-front: O(N)

Transpose: O(N)

Count: O(N)

Ordering: O(N) when searching/inserting

e. Move-to-front: O(N)

Transpose: O(N)

Count: O(N)

Ordering: O(N)

1. Pop all the elements of S and pushback into D. Then do the same thing to the elements in T. Now the elements are all in D, in reversed order. Finally, popback all the elements in D and push them into S. S should have all its items in original order with all elements from T below.
2. Include two stacks in this data structure, one called R and the other called S. When pushing an item into the stack, push it straight into R. If S is empty or the item’s value is smaller than the element on top of S, push it into S also. When popping an element from the data structure, simply pop it from R; if the element is on top of S, pop it from S also. The findMin operation can be implemented by call top for S stack.
3. Suppose there are two stacks, T1, T2. To make them work as a stack, T1 should be working as a “pushing queue”—receiving pushes; T2 should be working as a “popping queue”. When T2 is empty, pop the elements from T1 and insert them into T2. Because after the insertion into T2, the order of the elements is reversed, popping from T2 becomes “queue popping”. Pushing is simply done by pushing into T1. The worst case for operation time is when T2 is empty; popping all elements from T1 and pushing them into T2 has time complexity O(N)
4. Suppose there are two queues, Q1, Q2. To make them work as a stack, the only change needed is the way how they pop elements. Suppose there already are elements that were pushed into Q1. Because a queue always pops items from the bottom/back, to pop an element at the top as a stack, all elements in Q1 should be transfer to Q2 except for the top one. The one element left in Q1 then should be popped. In this way, the top item is popped. The elements switch places between Q1 and Q2 for every popping. Pushes are done exactly the same. The running time complexity is O(1) for pushing and O(N) for popping.
5. a.

Stack<int> T, W;

while (! S.isEmpty())

{

T.push(S.pop());

W.push(T.pop());

}//copy elements of S to W, keep order

while (! W.isEmpty())

{

S.push(W.pop());

}//pop all elements in W and push into S, reverse order

b.

Queue<int> Q;

while (! S.isEmpty())

{

Q.enqueue(S.pop())

}// pop all elements in S and enqueue them into Q, reverse order

while (!Q.isEmpty())

{

S.push(Q,dequeue());

}// transfer the elements (already in reversed order) in Q to S.

c.