CS101
midterm review

(but, don't do THAT)

Let's get started

Stack & Queue

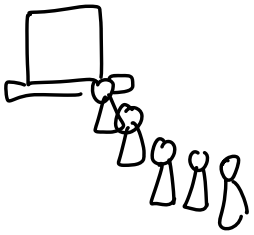
- Abstraction

Stack:

Access:
top

LIFO

Queue



Access:
front
rear.

FIFO

- Implementation

Array ? single Linked-list ?

Limit: ' pop ' cannot happen at the end of the LL it O(1)

如何用两个堆栈模拟实现一个队列？如果这两个堆栈的容量分别是m和n（m>n），你的方法能保证的队列容量是多少？

stack
reverse
2 stack
re-reverse

2n+1

容量为m的堆栈做储存空间，容量为n的堆栈做缓存空间。

1. 先将n个元素放入m中，如图中图1。

2. 将储存空间栈中的每个元素pop出，然后根据pop的顺序push到缓存空间栈，此时两个栈的空间如图中图2。

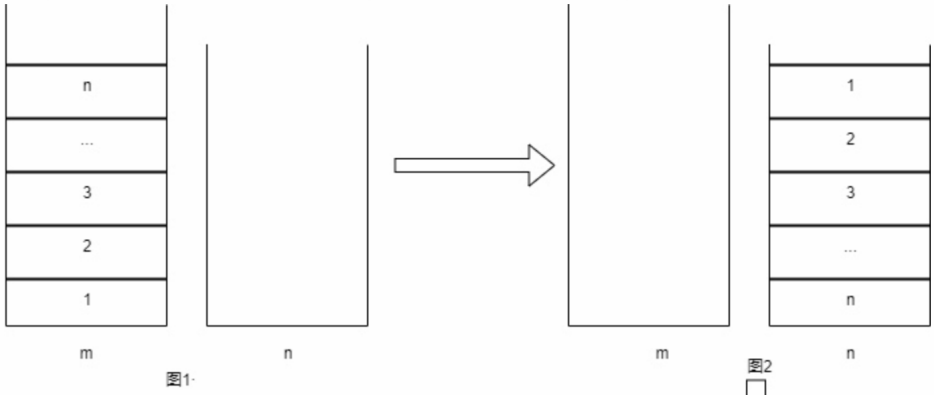
3. 然后是后n+1个元素放进储存空间栈中，如图中图3。

4. 此时已经入队了2n+1个元素，然后出队，先是缓存空间栈pop，得到的是1,2,3，...，n

5. 然后重复第2部中的元素，得到图中图4的效果。

6. 最后先pop储存空间的n+1元素，再pop缓存空间的元素，得到的结果和栈的效果一样。

但是要继续放元素，那么也就不能满足队的顺序入队，顺序出队（先进先出）了，所以连个栈所模拟的队的最大



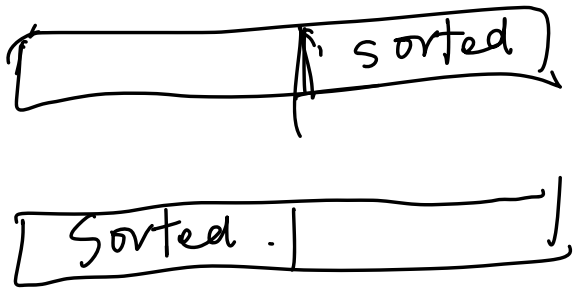
容量为2n+1。

Sorting

- Bubble Sort
- Insertion Sort
- Merge Sort
- Quick Sort

Sort a sub-range of array

Reverse bubble ✗
insertion



Question 6. *In the lecture we have learnt that different sorting algorithms are suitable for different scenarios. Then which of the following options is/are suitable for insertion sort?*

(A) Each element of the array is close to its final sorted position.

(B) A big sorted array concatenated with a small sorted array

(C) An array where only few elements are not in its final sorted position.

(D) None of the above.

A. $a_1 \quad a_2 \quad \cdots \quad a_j \cdots a_i \cdots a_n$

4.1: Given an array of n elements, where each element is at most k away from its target position, insertion sort can give a $\mathbf{O}(kn)$ performance.

k.

if $i > j$ $a[i] > a[j]$

inversions:

$$\sum_i \sum_{j<i} \mathbf{1}\{a[j] > a[i]\} \leq \sum_i \sum_{j<i} \mathbf{1}\{j+k > i-k\}$$

$$\leq \sum_i (2k-1) \leq 2kn$$

Insertion Sort:

$\Theta(n+d)$ $\quad \Theta(n) \quad d = O(n)$

$\Theta(n^2) \quad d = O(n^2)$

Question 7. *Applying insertion sort and the most basic bubble sort without a flag respectively on the same array, for both algorithms, which of the following statements is/are true? (simply assume we are using swapping for insertion sort)*

(A) There are two for-loops, which are nested within each other.

(B) They need the same amount of swaps. ✓

(C) They need the same amount of element comparisons. insertion sort # swaps < bubble sort (no "break")

(D) None of the above.

Implementation and Analysis

Recall: each time we perform a swap, we remove an inversion

```
template <typename Type>
void insertion_sort( Type *const array, int const n ) {
    for ( int k = 1; k < n; ++k ) {
        for ( int j = k; j > 0; --j ) {
            if ( array[j - 1] > array[j] ) {
                std::swap( array[j - 1], array[j] );
            } else {
                // As soon as we don't need to swap, the (k + 1)st
                // is in the correct location
                break;
            }
        }
    }
}
```

swaps = # inversions

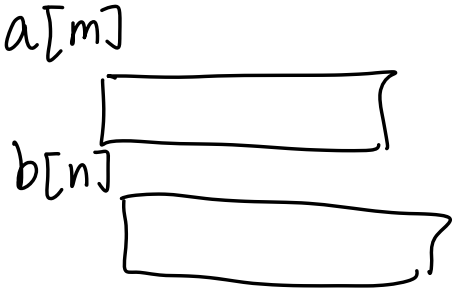
(B) *Given 2 sorted lists of size m and n respectively, and we want to merge them to one sorted list by mergesort. Then in the worst case, we need $m + n - 1$ comparisons.*

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$$

$\Downarrow$

put $m+n$ elements

Most intuitionly: if $m=n$



and in this order

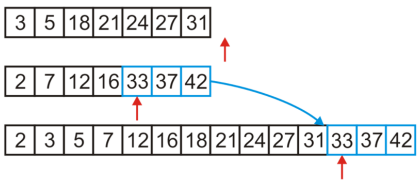
$a_0\ b_0\ a_1\ b_1\ \ldots\ldots$

$\Downarrow$

$m+n-1$ comparisons

Actually. as long as we avoid this:

After this, we simply copy over all remaining entries in the non-empty array



$m+n-1$ ✓

$\Uparrow$

i.e. If one array is empty.

at most 1 element left in another array

Question 9. *Given extra information about the input array, we may design sorting algorithms that perform faster than $O(NlogN)$. Which of the following prior knowledge will lead to worst time complexity* ***slower*** *than $O(N)$?*

(A) Knowing the input array has no more than N inversions. Insertion Sort

(B) Knowing the input array has exactly $(N^2 - N)/2$ inversions. Reverse Order

(C) Knowing the input array has less than N pairs of numbers that are not inversions.

(D) None of the above.

→ C → ✓
O(n) O(n)