Bubble sort

1. Analyze the following code of bubble sort. Write down the time complexity in the best case, average case and the worst case.(O-notation)
2. Modify ***one line*** of the code to optimize bubble sort. Write down the time complexity of the modified code in the best case, average case and the worst case.(O-notation) Note that after your optimization, it should run faster in the best case.

```
1 void bubbleSort(int arr[], int n)
2 {
3     int swapped = 1;
4     int i,j = 0;
5     int tmp;
6     for(j =0; j<n-1; j++){
7         swapped = 0;
8         for (i = 0; i < n - j-1; i++){
9             if (arr[i] > arr[i + 1]) {
10                tmp = arr[i];
11                arr[i] = arr[i + 1];
12                arr[i + 1] = tmp;
13                swapped = 1;
14            }
15        }
16        if(swapped == 1)
17            printf("swapped!\n")
18    }
19 }
```

1. **Best O(n^2), avg O(n^2), worst O(n^2)**
   不論給定的array為何，都需要跑完兩層loop。因此time complexity皆為O(n^2)。
2. **比較swapped的質，若是1的話就跳出line 6 的for loop。**
   e.g.

```
6     for(j =0; j<n-1 && swapped; j++){
```

Best O(n), avg O(n^2), worst O(n^2)

　如果兩兩比較相鄰的element後，都沒有進行swap，代表已經排序好了，所以就不需要繼續sort，可以直接跳出 line 6 的for loop。

　假若給定的array是已經排序好的，只需要跑 line 8 的 for loop進行相鄰element的比較 n-1次，因此best case為 O(n)。

The Same Birthday

Consider the following three algorithms for determining whether anyone in the room has the same birthday as you.

- *Algorithm 1*: You say your birthday, and ask whether anyone in the room has the same birthday. If anyone does have the same birthday, they answer yes.
- *Algorithm 2*: You tell the first person your birthday, and ask if they have the same birthday; if they say no, you tell the second person your birthday and ask whether they have the same birthday; etc, for each person in the room.
- *Algorithm 3*: You only ask questions of person 1, who only asks questions of person 2, who only asks questions of person 3, etc. You tell person 1 your birthday, and ask if they have the same birthday; if they say no, you ask them to find out about person 2. Person 1 asks person 2 and tells you the answer. If it is no, you ask person 1 to find out about person 3. Person 1 asks person 2 to find out about person 3, etc.

1. For each algorithm, what is the factor that can affect the number of questions asked (the "problem size")?
2. In the worst case, how many questions will be asked for each of the three algorithms?
Suppose there are N people(not including you) in the room.
3. For each algorithm, say whether it is constant, linear, or quadratic in the problem size in the worst case.

1: 房間內的人數
2: Algorithm 1: 1次。
Algorithm 2: N次.。Worst case 是沒有人和你同一天生日，需要問每個人。
Algorithm 3: N(N+1)/2次。Worst case 是沒有人和你同一天生日，總共需要發問次數為1+2+3+...+N-1+N。
3: Algorithm 1: constant
Algorithm 2: linear
Algorithm 3: quadratic

Reference: http://pages.cs.wisc.edu/~vernon/cs367/notes/3.COMPLEXITY.html#youtry2

Queue using stack

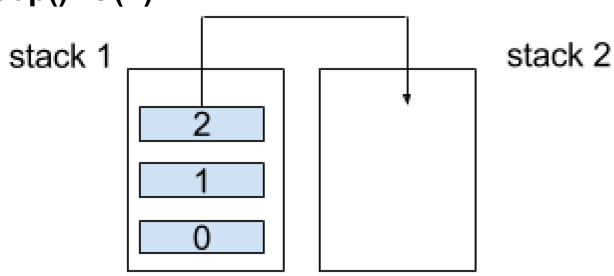
Use 2 stacks to support the following operations of a queue:

- pop(): removes the element from in front of the queue
- push(x): push an element x to the back of the queue

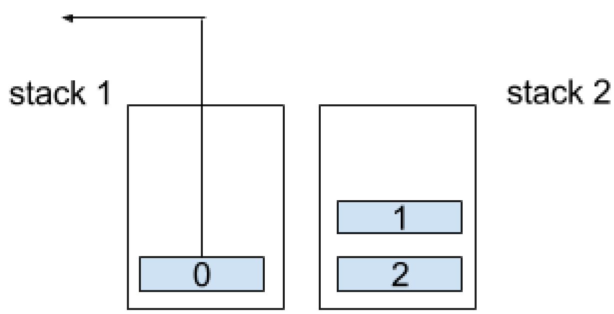
What is the time complexity of pop() and push(x)? Write down in O-notation.
Besides, briefly explain how you implement pop() and push(x).

方法一：

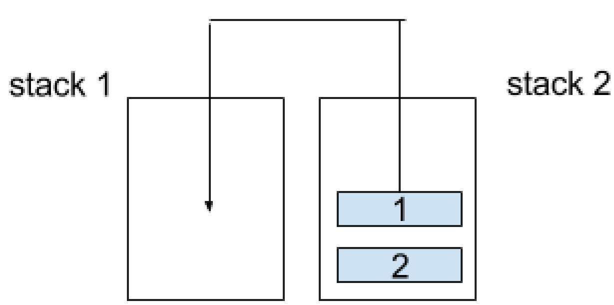
- **pop(): O(n)**



1. 把stack 1的element一個一個放到stack 2中。



2. stack 1中最後一個element不放到stack 2，直接拿掉。

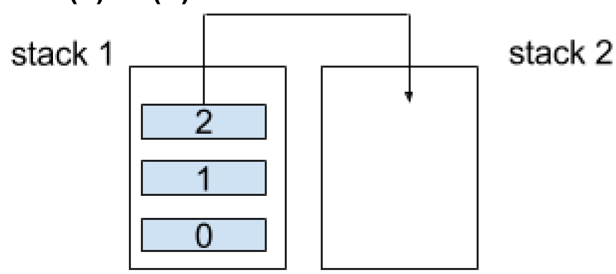


3. 把stack 2中的element全部放回stack 1。

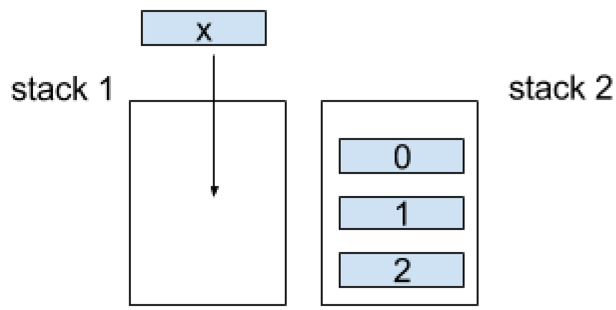
- **push(x): O(1)**
push(x)時直接將x放到stack 1當中。

方法二：

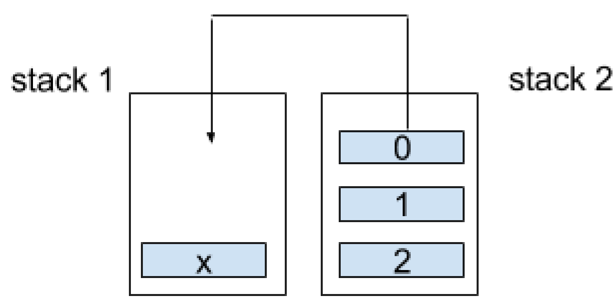
- **pop(): O(1)**
  直接pop stack 1。
- **push(x): O(n)**



1. 把stack 1的element全部放到stack 2。



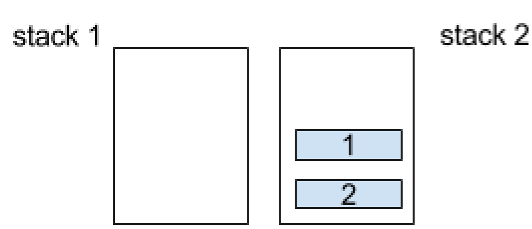
2. 將element x 放到stack 1



3. 把stack 2中的element全部放回stack 1。

方法三：

- **pop(): O(n)**
  兩種情況而worst case為O(n)：
  a. O(n):
  同方法一的步驟1、2，但不作步驟3，也就是原本的element會以相反的堆疊順序存於stack 2中，而stack 1則是empty。



b. O(1):

當stack 1是empty，而所有element都在stack 2並呈相反堆疊順序如上圖時，pop()時只要直接將stack 2中的top element拿掉即可。

- **push(x): O(n)**

  兩種情況而worst case為O(n)：

  a. O(n):

  當stack 1是empty，而所有element都在stack 2並呈相反堆疊順序時，要將stack 2當中所有的element都放回stack 1後，再將push(x)的element放到stack 1。

  B. O(1):

  若stack 2為empty，直接將element x放入stack 1即可。