Sliding window

Mr. Akarapon Watcharapalakorn
Doctor of Optometry (O.D.)

Problem

Given a list L of n numbers $[a_1, a_2, \ldots, a_n]$ *and a number k satisfying* $1 \leq k < n$, *let* $b_i = \min[a_i, a_{i+1}, \ldots, a_{i+k-1}]$ *be the minimum of the k elements in the length k sublist of L starting at* a_i. *Describe an efficient algorithm which outputs the sequence* $b_1, b_2, \ldots, b_{n-k+1}$.

A	3	2	1	4	5	-1	7	0	9	2

B	3	4	5	5	7	7	9	9

K = 3

Heap

- ใช้ max heap เก็บ pair ของ index และ value
- ใช้ค่า top ทุกครั้งในแต่ละรอบ เช็ค index ว่าอยู่ในช่วงรึเปล่า? ถ้าไม่ pop ทิ้ง

  => ต้องการค่ามากสุดที่อยู่ในช่วง

O(NlogN)

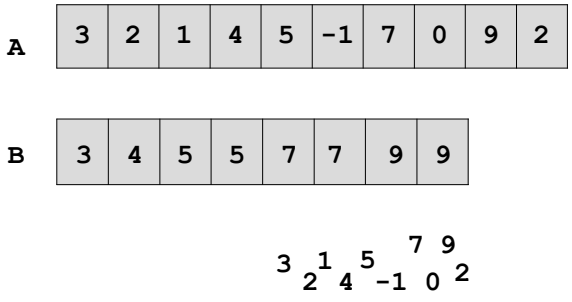
Sliding window

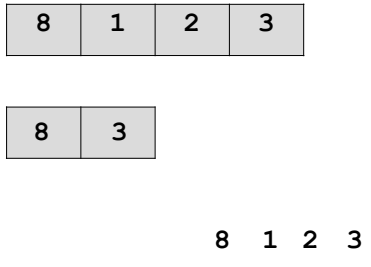
ลองลดขั้นตอน heapify จาก O(logN) ให้เหลือ O(1)

....ตอนที่เพิ่มเข้ามา 1 ตัว ถ้าค่าของมันมากกว่าตัวก่อนๆหน้า ให้ pop ตัวก่อนๆหน้าทิ้ง

=> ไม่มีการค้างของตัวที่ไม่มีโอกาส

O(N)





Sliding window

- 1. เพิ่มสมาชิกใหม่ และ pop สมาชิกใน window ที่ไม่มีวันเป็นคำตอบทิ้ง
- 2. เช็คว่าสมาชิกที่มากที่สุดของ window อยู่ในช่วงรีเปล่า? ถ้าไม่ก็ pop ทิ้ง
- 3. คำตอบในแต่ละรอบ(ช่วง k ตัว) คือตัวแรกของ window (front)

- Notice!! สมาชิกใน window จะเรียงตลอด
- Notice!!! สมาชิกแต่ละตัวจะเพิ่มและลบออกอย่างละครั้ง รวมทั้งหมดเป็น 2N ครั้ง = O(N)

```
void maxSlidingWindow(int A[], int n, int w, int B[]) {
  deque<int> Q;
  for (int i = 0; i < w; i++) {
    while (!Q.empty() && A[i] >= A[Q.back()])
      Q.pop_back();
    Q.push_back(i);
  }
  for (int i = w; i < n; i++) {
    B[i-w] = A[Q.front()];
    while (!Q.empty() && A[i] >= A[Q.back()])
      Q.pop_back();
    while (!Q.empty() && Q.front() <= i-w)
      Q.pop_front();
    Q.push_back(i);
  }
  B[n-w] = A[Q.front()];
}
```

Implement ด้วย linked list queue, Array queue หรือ Dequeue