Homework #2

Due: September 19–22, 2017

This is an oral presentation assignment. Again, there are three regular problems (#1-#3) and one programming problem (#4). You should work in groups of three. The sign-up sheet will be online soon (details on Piazza) and your group should sign up for a 45 minute slot. Each person in the group must be able to present every regular problem. The TA/Professor will select who presents which regular problem. You are not required to hand anything in at your presentation, but you may if you choose.

The programming problem will be submitted to autolab, similar to HW#1. It is due at 11:59pm on Friday September 22nd. You will not have to present anything orally for the programming problem. Please include a comment in your program explaining the algorithm you used. You can discuss the problem with your group-mates, but must write the program by yourself. Please do not copy.

Problem #B1 is a “bonus” problem. It is optional, and you won’t get hints or assistance from the course staff. It is worth 1% under the reci/quiz/bonus category if you solve it correctly. No partial credit, sorry. The solution to this problem must be written up and submitted via gradescope by September 22nd.

(25 pts) 1. (Balls in Bins) There are $n$ balls and an infinite number of bins. A bin can have 0 or more balls in it. A move consists taking all the balls of some bin and putting them into distinct bins. The cost of a move is the number of balls moved.

Define the potential of a state of this system as the sum of the potentials of all the bins. The potential of a bin with $k$ balls in it is:

$$\Phi(k) = \max(0, k - z)$$

Where for convenience $z = \lfloor \sqrt{n} \rfloor$.

Prove that the amortized cost of a move is at most $2z$. 


2. (Cheap Heaps) A heap is a data structure to store a multi-set of keys which supports the following operations:

meld(A, B): Return the heap representing the union of the contents of the two heaps A and B.
findmin(A): Return the minimum key of heap A.
deletemin(A): Delete the minimum key from A and return the resulting heap.
isert(A, x): Insert the key x into the heap A and return the new heap.

A *cheap heap* implements this interface as follows. The cheap heap is a binary tree, where each node stores one key. The tree is heap ordered, meaning that the key in a node is at most that of its children.

`findmin(A)` simply returns the key in the root of A. All the other operations are implemented using `meld`. I.e. `deletemin(A)` deletes the root of A and melds its two children together. `insert(A, x)` creates a new tree with one node containing x and melds it with A.

To implement `meld(A, B)` we walk down the right paths from the roots of A and B and merge those paths together to form a new right path of non-decreasing keys. Now for each node along that path (except the last one) we swap the left and right children. The following figure shows the melding of two trees. (The subtrees labeled A, B, etc. are arbitrary and do not change. They could be empty.)

```
1            2                                 1
 / \          / \                               / \n A   5        D   3           meld              2   A
 / \          / \        =======>           / \n B   7        E   8                         3   D
 /            /                         /   \
 C            F                         5   E
 / \           /                         /   
7   B           8   C
 /   
F
```

WLOG define the running time of `meld` to be the number of nodes on the melded paths. (This would be six in the example above.)

(a) Define a potential function of a cheap heap. The potential of a collection of cheap heaps is just the sum of each individual tree’s potential. Your potential will be used to solve the remaining parts of this problem.

Hint: Define the size of a node in the cheap heap as the number of nodes in the subtree rooted there (as we did with splay trees). This allows you to classify edges of the tree to be *light* or *heavy* as follows. If the node at the top of the edge has size A and the one at the bottom has size B, and $A \geq 2B$ then the edge is light. Otherwise it is heavy. You can also classify an edge to be *right* if node B is a right child of A and *left* otherwise. The potential function will be the number of edges of a certain type in this classification system.
(b) Say that two trees are melded to produce a tree of size $n$. Using your potential function prove that the amortized cost of this operation is at most $1 + 4\log n$. (Even better is $1 + 3\log n$.)

(c) Suppose initially there are no cheap heaps, and a sequence of $m$ of the above listed operations are done, and that no heap ever exceeds size $n$. Prove that the total cost of all the operations done is $O(m \log n)$.

By the way, below is the complete Ocaml code to implement cheap heaps.

```ocaml
type 'a tree = Empty | Node of 'a tree * 'a * 'a tree

let rec meld a b =
  match (a,b) with
  | (Empty, b) -> b
  | (a, Empty) -> a
  | (Node(al, ak, ar), Node(bl, bk, br)) ->
    if (ak <= bk) then Node((meld ar b), ak, al)
    else Node((meld a br), bk, bl)

let insert a k =
  meld a (Node(Empty, k, Empty))

let findmin a =
  match a with Empty -> failwith "Findmin on empty heap"
  | Node(al, ak, ar) -> ak

let deletemin a =
  match a with Empty -> failwith "Deletemin on empty heap"
  | Node(al, ak, ar) -> meld al ar
```

(25 pts) 3. (Trade-offs) Suppose we want a data structure that represents a sequence of numbers $x_1, x_2, \ldots, x_n$ (initially 0), with the following operations:

- Increment($i, v$): $x_i \leftarrow x_i + v$
- SumToLeft($j$): return $\sum_{1 \leq i \leq j} x_i$

Let $f(n)$ be an asymptotic bound on the cost of the Increment() operation and let $g(n)$ be an asymptotic bound on the cost of the SumToLeft() operation.

(a) Show how to obtain $(f(n), g(n)) = (O(1), O(\sqrt{n}))$.
(b) Show how to obtain $(f(n), g(n)) = (O(1), O(n^{1/3}))$.
(c) Show how to obtain $(f(n), g(n)) = (O(\sqrt{n}), O(1))$.
(d) Show how to obtain $(f(n), g(n)) = (O(n^{1/3}), O(1))$. 
(25 pts) 4. (Simulating Move to Front) Start with a list \([0, 1, 2, \ldots, n - 1]\). Now you get a sequence of \(m\) move to front requests, where \(\text{mtf}(i)\) moves element \(i\) to the front of the list. When it does the move, your program should output the index of the element being moved. (Zero-based indexing is used, so \(i = 0\) is the front of the list.)

In this way, a sequence of \(m\) number in the range \([0, n - 1]\) is transformed into another such sequence. This transformation (and its inverse) is useful in implementing a certain data compression algorithm.

So, for example, say \(n = 5\). The initial list is \([0, 1, 2, 3, 4]\). If \(\text{mtf}(3)\) is executed, then the list becomes \([3, 0, 1, 2, 4]\), and \(3\) is output. If the next request is \(3\), then the list stays the same and \(0\) is output. Finally if the next request is \(\text{mtf}(4)\) then the list becomes \([4, 3, 0, 1, 2]\) and \(4\) is output.

INPUT: The first line contains blank-separated numbers \(n\) and \(m\) with \(1 \leq n \leq 10^6\) and \(1 \leq m \leq 5 \times 10^5\). The second line consists of the numbers \(p_1, p_2, \ldots, p_m\), each of which is in the range \([0, n - 1]\). These are the items to which the move-to-front operation is applied. The time limit is 10 seconds.

OUTPUT: The output consists of one line containing \(m\) space-separated numbers. The \(i\)th of these is the index of the item \(p_i\) when it is requested.

For example, if the input is:

```
5 3
3 3 4
```

then the output would be:

```
3 0 4
```

Or if the input is:

```
6 8
5 0 4 0 3 0 2 0
```

then the output would be:

```
5 1 5 1 5 1 5 1
```

(1 bonus pt.) B1. (No Short Jumps Allowed) You’re given a list of \(n\) numbers \(a_1, a_2, \ldots, a_n\). Your goal is to rearrange the numbers in such a way as to maximize the minimum gap between consecutive numbers. In other words, we seek a reordering \(b\) of the numbers of \(a\) such that the following sum is maximized:

\[
\min_{i \in 2, \ldots, n} |b_i - b_{i-1}|
\]

Give an algorithm for this problem that runs in time \(O(n \log n)\), and prove that it’s correct.