CMPUT 204 — Problem Set 3

(Partial solutions provided by Lisheng)

Topics covered in Part I are heapsort and priority queues; in Part II are quicksort and lower bounds for comparison based sorting.

It is highly recommended that you read pages **135–168** very carefully and do **all** the exercises. The following are some of them that you are **REQUIRED** to practice on.

Quiz questions are mostly based on this list, with some minor modifications necessary. Consult your instructor and TAs if you have any problem with this list.

Part I

1. P136, Ex 6.4-1.
2. P136, Ex 6.4-2.

   Hints:

   Proof of correctness via Loop Invariant(s) involves 3 phases: Initialization, Maintenance, and Termination.

   Proof of a Loop Invariant does not require you to "show the correctness of algorithm when the LI terminates", but you need to argue that the LI does terminate.
3. P136, Ex 6.4-3.

   Hints:

   Recall the BC running time analysis for "all keys are distinct". You can apply the sandwich property to claim that the running time is in $\Theta(n \log n)$ for both cases.
4. P140, Ex 6.5-1.
5. P140, Ex 6.5-2.
6. P142, Ex 6.5-5.
7. P142, Ex 6.5-6.

   Hints:

   First-in-first-out queue: use a min-priority queue.

   - use a counter k to denote the number of elements in the queue, initialized to 0
   - use another counter ℓ to denote the number of elements removed from the queue, initialized to 0
   - whenever a new element comes in, assign it priority k and add it to the queue;

     $k \leftarrow k + 1$

     running time $\Theta(1)$, since the newly added element has the maximum priority and thus stay at the last position
   - whenever an element is removed from the queue, $\ell \leftarrow \ell + 1$

     WC running time $\Theta(\log(k - \ell))$, since one min-heapify required
   - at the time k is increased and becomes larger than 2ℓ, perform the following "cleanup":

     reduce the priority for each element in the queue from x to $x - \ell$

     running time $\Theta(k - \ell)$

     $k \leftarrow k - \ell$

     $\ell \leftarrow 0$
   - the last "cleanup" step is necessary in the actual implementation, for otherwise the priority of a new element would keep increasing and would ultimately "overflow"

   Last-in-first-out queue: use a max-priority queue.

   - use a counter k to denote the number of elements in the queue, initialized to 0

     notice that $(k-1)$ is the maximum priority in the queue
   - whenever a new element comes in, assign it priority k and add it to the queue;

     $k \leftarrow k + 1$

WC running time $\Theta(\log k)$, since the newly added element has the maximum priority and thus has to be pop up the root

- whenever an element is removed from the queue, $k \leftarrow k - 1$

  running time $\Theta(\log(k-\ell))$, since one max-heapify required

8. P142, Ex 6.5-8.

   Hints:

   Use a k-element min-heap.

   - the elements in the min-heap are the k sorted lists
   - every element has the key value equal to the first number in the list
   - make them into a min-heap takes $\Theta(k)$ time
   - put the first number from the list at the root node into the first position of the sorted array

     update the root element to have a key value equal to the second number in the list

     min-heapify it

     WC running time is dominated by the min-heapify, and so $\Theta(\log k)$
   - repeat the above process until every list becomes empty

     overall running time is: adding one key to the sorted array takes $\Theta(\log k)$ time and there are in total n keys

     Therefore, $\Theta(n \log k)$ in total.

     Note: building heap is minor