

## CMPSC 130A: Winter 2011

### Programming Assignment 1

Assigned: Jan 27

Due: Feb 8 (11:59 PM)

Hashing is useful if we are only interested in insert, delete, and find operations; e.g., delete (10) or insert (5) or find (100). But a hash table provides no assistance for searches that depend on the order or rank of an element in a set, e.g., find the smallest element, delete the smallest element, etc. For the latter type of operation, heaps are more appropriate. On the other hand heaps do not support a general delete operation, e.g., delete (100); they only support deleting the min element.

In this programming assignment, you will develop a “compound” data structure called *Quash*, which is composed of both a min-heap (priority queue) and a hash table! The Quash supports insert, delete and lookup using its hash component, and deleteMin using its heap component. In particular, each element in the set will be inserted in **both** the heap and the hash table. You will need to include pointers in both directions between the 2 instances of the element in the heap and in the hash table. Operations should be executed as follows:

- *insert(i)*: Insert element  $i$  in both the heap and the hash table, and link them correctly. Your program should return either “item successfully inserted,” or “item already present.”
- *lookup (i)*: Use the hash table to determine if  $i$  is in the data structure. Your program should return either “item found” or “item not found.”
- *deleteMin*: Use the heap to delete the minimum element, and use the pointer to delete the element in the hash table. Your program should return the key value of the item deleted.
- *delete(i)*: Use the hash table to determine where  $i$  is, delete it from the hash table and use the pointer to delete it from the heap. (Note you will need to generalize the heap operations to support this “internal” delete operation.) Your program should return either “item successfully deleted” or “item not
- *print()*: Print out the hash table as well as the heap (in the array format).

Some specific implementation issues:

1. Assume all elements to be inserted are integers (but they can be negative).
2. Use the array implementation for the min-heap.
3. For this assignment, you can use “mod 43” as the hash function. (In general, it would be better to use an internal resize operation, which doubles the table size when the table becomes too full, and shrinks it when it becomes too empty, using appropriate thresholds for the full and empty. You are not required to implement resize, but you are welcome to try it if you like.)
4. For collision resolution, use **linear probing**.

5. Use the “lazy deletion” method by marking slots in the hash table as deleted; use a boolean *deleted* to indicate if a slot was once used but is now deleted.

For consistency, you can assume that the sequence of operations is provided in ascii format, with each command on a separate line.

**Example:**

```
insert 10
insert -50
insert 76
lookup 12
insert 12
lookup 12
deleteMin
delete 76
insert 12
deleteMin
lookup 76
```

**The TA should be able to run your program as “prog1” with input from stdin. Be sure to include a Makefile, and name the executable prog1.** TAs should be able to terminate your program by Ctrl-D (EOF for linux) or Ctrl-Z (EOF for windows). When we test using files, the EOF will be there.

In addition, your program should print out (one line per command) information after each operation, as follows:

- insert(i): “item i successfully inserted” or “item i already present”
- lookup(i): “item i found” or “item i not found”
- deleteMin: “min item i deleted” or ”min item not present since table is empty”
- delete(i): “item i successfully deleted” or ”item i not present”
- print(): prints out the hash table and heap array on two separate lines, with elements separated by a blank.