Recitation 13

Priority Queues and Hashing

13.1 Announcements

- PASLLab is due this **Friday, May 5**.
- The final exam is on **Friday, May 13**.
- A review session for the final is upcoming. Stay tuned!
- A practice final and its solutions will be released soon on the course website.

13.2 Leftist Heaps

Task 13.1. *Identify the defining properties of a leftist heap.*

Task 13.2. *What is an upper bound on the rank of the root of a leftist heap?*

13.2.1 Building A Leftist Heap

Consider the following pseudo-SML code implementing leftist heaps.

Data Structure 13.3. *Leftist Heap*

```
datatype PQ = Leaf | Node of int × key × PQ × PQ

fun rank Q =
  case Q of
    Leaf => 0
  | Node (r,_,_,_) => r

fun makeLeftistNode (k,A,B) =
  if rank A < rank B
  then Node (1 + rank A, k, B, A)
  else Node (1 + rank B, k, A, B)

fun meld (A,B) =
  case (A,B) of
    (_, Leaf) => A
  | (Leaf, _) => B
  | (Node (_,k_a,L_a,R_a), Node (_,k_b,L_b,R_b)) =>
      if k_a < k_b
      then makeLeftistNode (k_a, L_a, meld (R_a,B))
      else makeLeftistNode (k_b, L_b, meld (A,R_b))

fun singleton k = Node (1,k,Leaf,Leaf)

fun insert (Q,k) = meld (Q, singleton k)

fun fromSeq S = Seq.reduce meld Leaf (Seq.map singleton S)

fun deleteMin Q =
  case Q of
    Leaf => (NONE, Q)
  | Node (_,k,L,R) => (SOME k, meld (L,R))
```

Task 13.4. *Diagram the process of executing the code*

$\texttt{fromSeq}\ \langle 3, 5, 2, 1, 4, 6, 7, 8 \rangle$

Task 13.5. *What are the work and span of* ($\texttt{fromSeq}\ S$) *in terms of* $|S| = n$*?*

13.3 Removing Duplicates

Removing duplicates is a crucial substep of many interesting algorithms. For example, in BFS, consider the step where we construct a new frontier. One viable method would to be to generate the sequence of all out-neighbors, and then remove duplicates:

$$F' = \texttt{removeDuplicates}\ \left\langle v : u \in F, v \in N_G^+(u) \right\rangle$$

So, how fast is it to remove duplicates? Can we do it in parallel?

13.3.1 Sequential

Before we think about parallelism, we should acquaint ourselves with a good sequential algorithm solving the same problem. This way, we know what to shoot for in terms of work bounds, since we want our parallel algorithm to be asymptotically work-efficient.

Task 13.6. *Describe a sequential algorithm which performs expected $O(n)$ work to remove duplicates from a sequence of length n. Also argue that $\Omega(n)$ work is necessary in order to solve this problem, and conclude that your algorithm is asymptotically optimal.*

Hint: try hashing elements one at a time.

13.3.2 Parallel

Task 13.7. *Implement a function*

val *removeDuplicates* : $(\alpha \times \textit{int} \rightarrow \textit{int}) \rightarrow \alpha\ \textit{Seq.t} \rightarrow \alpha\ \textit{Seq.t}$

where ($removeDuplicates$ h S) retuns a sequence of all unique elements of S, given that $h(e, m)$ hashes the element e to a uniform random integer in the range $[0, m)$ (thus the probability of collision for any two distinct elements is $1/m$).

Hint: as a first attempt, try simultaneously hashing as many elements as possible all at the same time. What do you do when elements collide?

13.4 Additional Exercises

Exercise 13.8.

Task 13.9. *Design a data structure which supports the following operations:*

	Work	*Span*	*Description*
`fromSeq` S	$O(\|S\|)$	$O(\log^2 \|S\|)$	*Constructs a dynamic median data structure from the collection of keys in* S
`median` M	$O(1)$	$O(1)$	*Returns the median of all keys stored in* M
`insert` (M, k)	$O(\log \|M\|)$	$O(\log \|M\|)$	*Inserts* k *into* M

For simplicity, you may assume that all elements inserted into such a structure are distinct.

Exercise 13.10. *Prove a lower bound of* $\Omega(\log n)$ *for* `deleteMin` *in comparison-based meldable priority queues. That is, prove that any meldable priority queue implementation which has a logarithmic* `meld` *cannot support* `deleteMin` *in faster than logarithmic time.*