Data Structures and Algorithms

A Summary of Important Stuff

Author: Robby Renz

Important Points

Snapshot Iterators

- maintains its own private copy of the sequence of elements
- constructed at the time the iterator object is created
- records a "snapshot" of the sequence of elements at the time the iterator is created
- therefore, it is unaffected by any subsequent changes to the primary collection that may happen
- advantages:
  - implementing snapshot iterators is very easy
  - as it requires a simple traversal of the primary structure
- disadvantages:
  - requires *O(n)* time upon construction to copy and store a collection of n elements

Lazy Iterators

- does not make an upfront copy
- instead, it performs a piecewise traversal of the primary structure only when the `next()` method is called to request another element
- advantages:
  - typically be implemented so the iterator requires only *O(1)* construction time
- disadvantages (feature):
  - its behaviour is affected if the primary structure is modified by means other than by the iterator's own *remove* method before the iteration completes

Map ADT

- something here

Heaps

- more of something here

Merge-Sort

- it has a time complexity of *O(nlogn)*

Quick-Sort

- randomized sorting algorithm based on the divide-and-conquer paradigm
- it has a expected time complexity of *O(nlogn)*
- code:

```
// quick-sort contents of a queue
public static <K> void quickSort(Queue<K> S, Comparator<K> comp) {
int n = S.size();
if (n < 2)
    return; // queue is trivially sorted
// divide
K pivot = S.first(); // using first as an arbitrary pivot
Queue<K> L = new LinkedQueue<>();
Queue<K> E = new LinkedQueue<>();
Queue<K> G = new LinkedQueue<>();
while (!S.isEmpty()) { // divide original into L, E and G
    K element = S.dequeue();
    int c = comp.compare(element, pivot);
    if (c < 0) // element is less than pivot
        L.enqueue(element);
    else if (c == 0) // element is equal to pivot
        E.enqueue(element);
    else // element is greater than pivot
        G.enqueue(element);
}
// conquer
quickSort(L, comp); // sort elements less than pivot
quickSort(G, comp); // sort elements greater than pivot
// concatenate results
while (!L.isEmpty())
    S.enqueue(L.dequeue());
while (!E.isEmpty())
    S.enqueue(E.dequeue());
while (!G.isEmpty())
    S.enqueue(G.dequeue());
}
```

Divide-and-Conquer

- explain

LIFO and FIFO

- which data structure uses it?