Priority Queues

<table>
<thead>
<tr>
<th>Action</th>
<th>Quantity</th>
<th>Company</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sell</td>
<td>100</td>
<td>IBM</td>
<td>$122</td>
</tr>
<tr>
<td>Sell</td>
<td>300</td>
<td>IBM</td>
<td>$120</td>
</tr>
<tr>
<td>Buy</td>
<td>500</td>
<td>IBM</td>
<td>$119</td>
</tr>
<tr>
<td>Buy</td>
<td>400</td>
<td>IBM</td>
<td>$118</td>
</tr>
</tbody>
</table>
Outline and Reading

- PriorityQueue ADT (§8.1)
- Total order relation (§8.1.1)
- Comparator ADT (§8.1.2)
- Sorting with a priority queue (§8.1.3)
- Selection-sort (§8.2.3)
- Insertion-sort (§8.2.3)
Priority Queue ADT

- A priority queue stores a collection of items
- An item is a pair (key, element)
- Main methods of the Priority Queue ADT
  - \texttt{insert}(k, o)
    - inserts an item with key \( k \) and element \( o \)
  - \texttt{removeMin}()
    - removes the item with smallest key and returns its element

Additional methods

- \texttt{minKey}()
  - returns, but does not remove, the smallest key of an item
- \texttt{minElement}()
  - returns, but does not remove, the element of an item with smallest key
- \texttt{size()}, \texttt{isEmpty}()

Applications:

- Standby flyers
- Auctions
- Stock market
Total Order Relation

- Keys in a priority queue can be arbitrary objects on which an order is defined.
- Two distinct items in a priority queue can have the same key.

Mathematical concept of total order relation $\leq$

- Reflexive property: $x \leq x$
- Antisymmetric property: $x \leq y \land y \leq x \Rightarrow x = y$
- Transitive property: $x \leq y \land y \leq z \Rightarrow x \leq z$
Comparator ADT

A comparator encapsulates the action of comparing two objects according to a given total order relation.

A generic priority queue uses an auxiliary comparator.

The comparator is external to the keys being compared.

When the priority queue needs to compare two keys, it uses its comparator.

Method of the Comparator ADT, all with int return type:

- compare(a, b)
- If the return is 0
  - a = b
- If the return is greater than 0
  - a > b
- If the return is less than 0
  - a < b
Sorting with a Priority Queue

We can use a priority queue to sort a set of comparable elements
1. Insert the elements one by one with a series of insert(e, e) operations
2. Remove the elements in sorted order with a series of removeMin() operations

The running time of this sorting method depends on the priority queue implementation

Algorithm \(PQ\text{-Sort}(S, C)\)

Input sequence \(S\), comparator \(C\) for the elements of \(S\)

Output sequence \(S\) sorted in increasing order according to \(C\)

\(\begin{align*}
P & \leftarrow \text{priority queue with comparator } C \\
\text{while } & \neg S.\text{isEmpty}() \\
& e \leftarrow S.\text{remove} (S.\text{first}()) \\
P.\text{insert}(e, e) \\
\text{while } & \neg P.\text{isEmpty}() \\
& e \leftarrow P.\text{removeMin}() \\
S.\text{insertLast}(e)
\end{align*}\)
Sequence-based Priority Queue

- Implementation with an unsorted sequence
  - Store the items of the priority queue in a list-based sequence, in arbitrary order
  - Performance:
    - insert takes $O(1)$ time since we can insert the item at the beginning or end of the sequence
    - removeMin, minKey and minElement take $O(n)$ time since we have to traverse the entire sequence to find the smallest key

- Implementation with a sorted sequence
  - Store the items of the priority queue in a sequence, sorted by key
  - Performance:
    - insert takes $O(n)$ time since we have to find the place where to insert the item
    - removeMin, minKey and minElement take $O(1)$ time since the smallest key is at the beginning of the sequence
Selection-Sort

Selection-sort is the variation of PQ-sort where the priority queue is implemented with an unsorted sequence.

Running time of Selection-sort:
1. Inserting the elements into the priority queue with \( n \) insert operations takes \( O(n) \) time.
2. Removing the elements in sorted order from the priority queue with \( n \) removeMin operations takes time proportional to

\[
n + (n - 1) + \ldots + 2 + 1
\]

Selection-sort runs in \( O(n^2) \) time.
**Insertion-Sort**

- Insertion-sort is the variation of PQ-sort where the priority queue is implemented with a sorted sequence

- **Running time of Insertion-sort:**
  1. Inserting the elements into the priority queue with $n$ insert operations takes time proportional to $1 + 2 + \ldots + n$
  2. Removing the elements in sorted order from the priority queue with a series of $n$ removeMin operations takes $O(n)$ time

- Insertion-sort runs in $O(n^2)$ time
In-place Insertion-sort

- Instead of using an external data structure, we can implement selection-sort and insertion-sort in-place.
- A portion of the input sequence itself serves as the priority queue.
- For in-place insertion-sort:
  - We keep sorted the initial portion of the sequence.
  - We can use `swapElements` instead of modifying the sequence.