7.1 Background

We have discussed Abstract Lists with explicit linear orders

- Arrays, linked lists, strings

We saw three cases which restricted the operations:

- Stacks, queues, deques

Following this, we looked at search trees for storing implicit linear orders: Abstract Sorted Lists

- Run times were generally $\Theta(\ln(n))$

We will now look at a restriction on an implicit linear ordering:

- Priority queues

7.1.1 Definition

With queues

- The order may be summarized by *first in, first out*

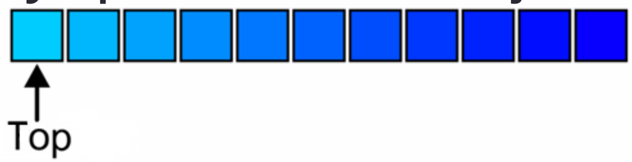
If each object is associated with a priority, we may wish to pop that object which has highest priority

With each pushed object, we will associate a nonnegative integer (0, 1, 2, ...) where:

- The value 0 has the *highest* priority, and
- The higher the number, the lower the priority

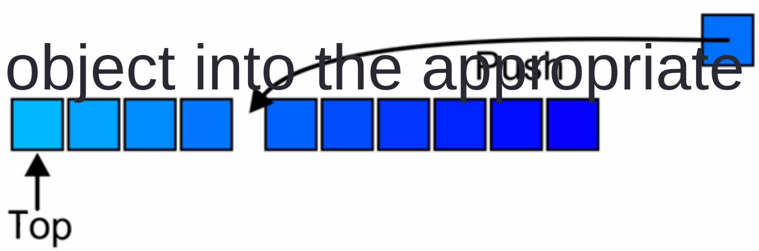
7.1.2 Operations

The top of a priority queue is the object with highest priority



Popping from a priority queue removes the current highest priority object:

Push places a new object into the appropriate place



7.1.3 Lexicographical Priority

Priority may also depend on multiple variables:

- Two values specify a priority: (a, b)
- A pair (a, b) has higher priority than (c, d) if:
  - $a < c$, or
  - $a = c$ and $b < d$

For example,

- (5, 19), (13, 1), (13, 24), and (15, 0) all have *higher* priority than (15, 7)

7.1.4

Process Priority in Unix

This is the scheme used by Unix, *e.g.*,

```
% nice +15 ./a.out
```

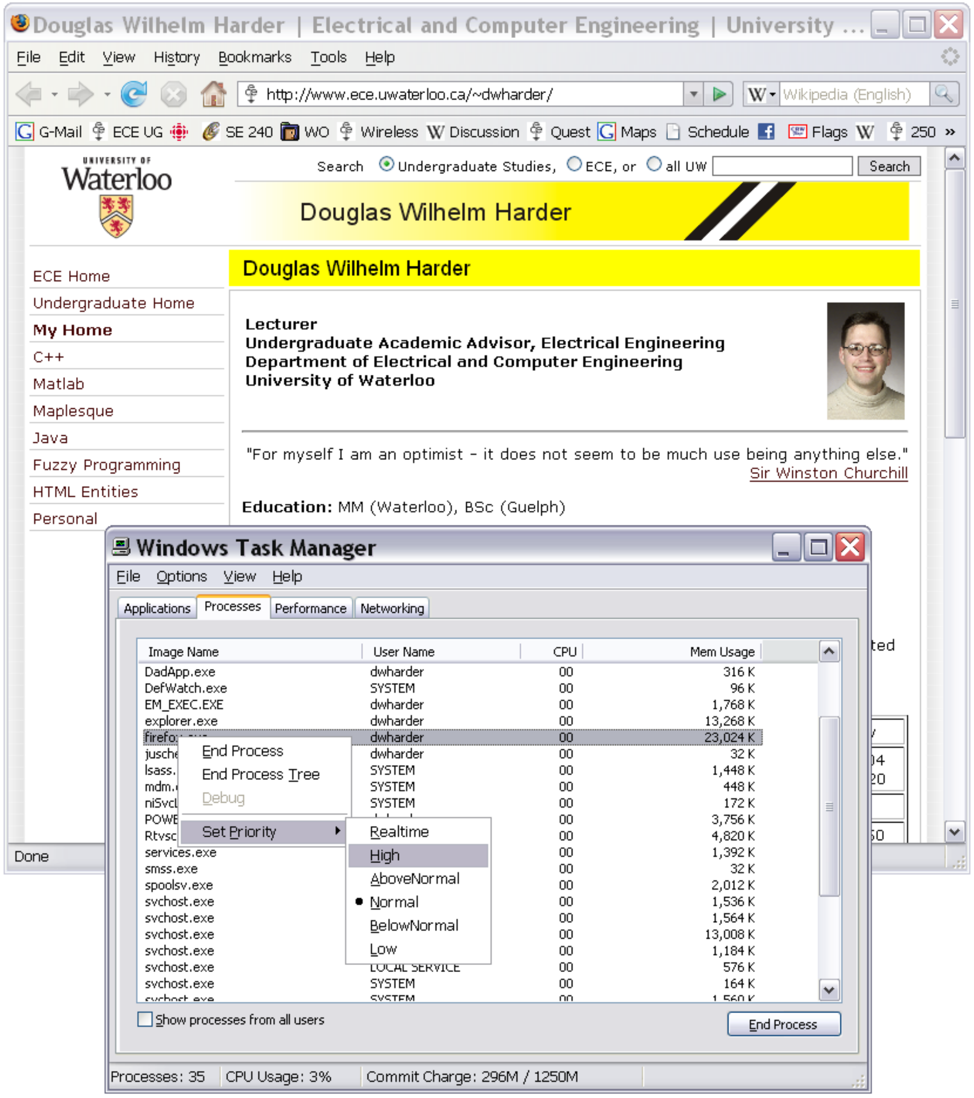
reduces the priority of the execution of the routine `a.out` by 15

This allows the processor to be used by interactive programs

- This does not significantly affect the run-time of CPU-bound processes

7.1.4 Process Priority in Windows

The priority of processes in Windows may be set in the *Windows Task Manager*



7.1.5 Implementations

Our goal is to make the run time of each operation as close to $\Theta(1)$ as possible

We will look at two implementations using data structures we already know:

- Multiple queues—one for each priority
- An AVL tree

The next topic will be a more appropriate data structure: the heap

7.1.5.1 Multiple Queues

Assume there is a fixed number of priorities, say M

- Create an array of M queues
- Push a new object onto the queue corresponding to the priority
- Top and pop find the first empty queue with highest priority

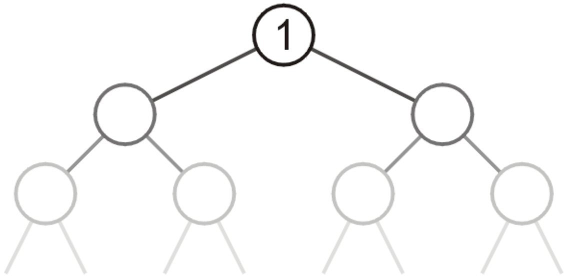
7.1.5.3 Heaps

Can we do better?

- That is, can we reduce some (or all) of the operations down to $\Theta(1)$?

The next topic defines a *heap*

- A tree with the top object at the root
- We will look at binary heaps
- Numerous other heaps exists:
  - d-ary heaps
  - Leftist heaps
  - Skew heaps
  - Binomial heaps
  - Fibonacci heaps
  - Bi-parental heaps



References

Cormen, Leiserson, Rivest and Stein,
Introduction to Algorithms, The MIT Press, 2001, §6.5, pp.138-44.

Mark A. Weiss,
Data Structures and Algorithm Analysis in C++, 3rd Ed., Addison Wesley, 2006, Ch.6, p.213.

Joh Kleinberg and Eva Tardos,
Algorithm Design, Pearson, 2006, §2.5, pp.57-65.

Elliot B. Koffman and Paul A.T. Wolfgang,
Objects, Abstractions, Data Structures and Design using C++, Wiley, 2006, §8.5, pp.489-96