Gravitational Waves with GStreamer Workshop

Writing a GStreamer element in Python

Leo Singer

LIGO Laboratory, California Institute of Technology

November 8, 2010

Introduction

In this tutorial, you will learn how to create your first GStreamer element in Python.

We will start by defining a simple problem. Then, we will walk through the steps needed to create an element from scratch, finally completing an element that solves this problem.

If, at the end, you have time to spare, you we encourage you to try one of the *Bonus* projects to make your first GStreamer element more powerful.

Some of these tasks will teach you more about GStreamer, and some of them will teach you more about designing algorithms and data structures for stream programming.

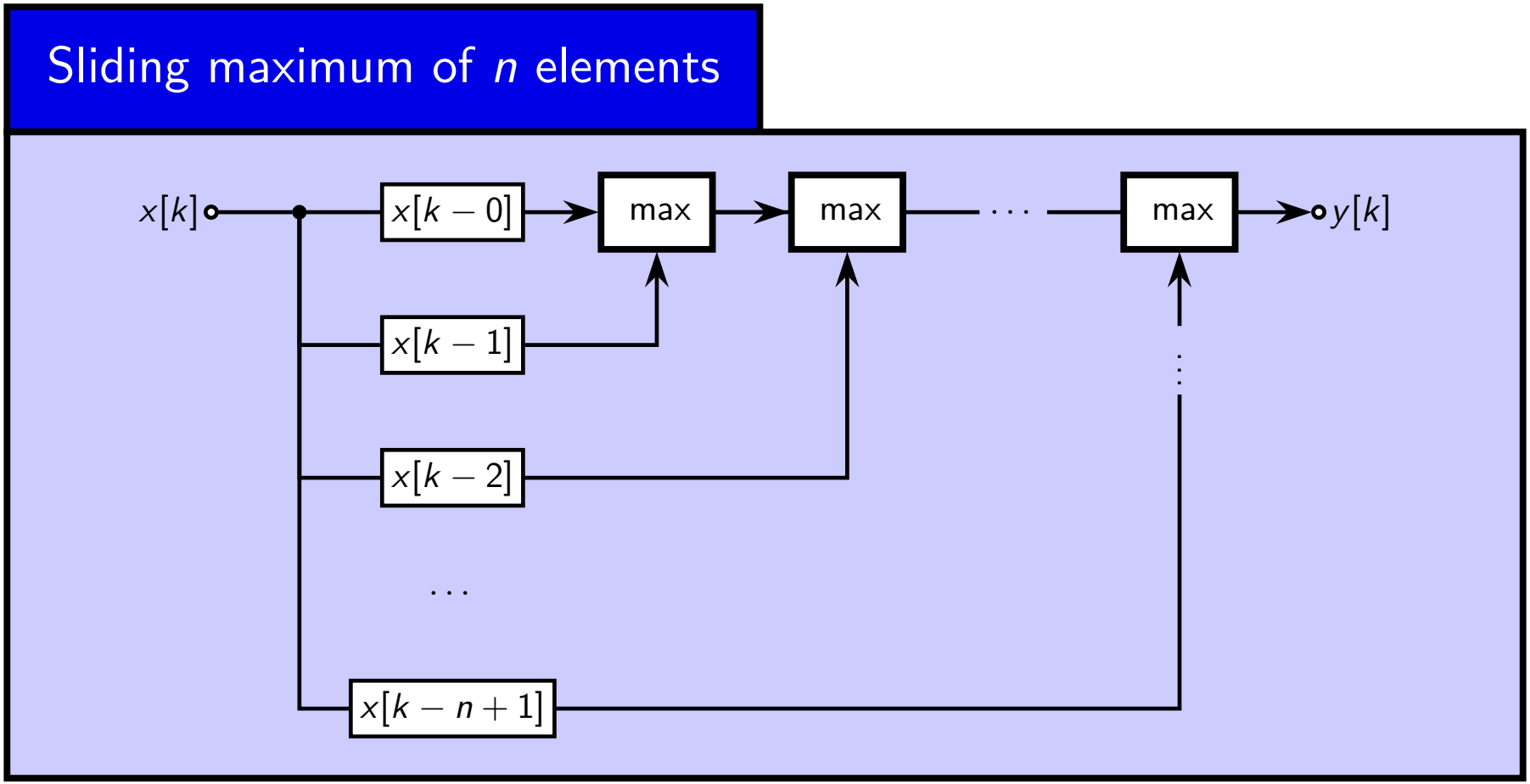
Who is this tutorial for?

This tutorial assumes that you have:

- `gstreamer`, `gst-plugins-base`, `gst-python`, and `gstlal` installed
- Understanding of core GStreamer concepts like caps, pads, and buffers
- Previous experience with Python
- Exposure to big-$\mathcal{O}$ notation

1. Problem Definition
   - Applications
2. Code Walkthrough
   - Element skeleton
   - Add pad templates and caps
   - Adding properties
   - Pause to admire your work
   - Implement desired behavior
   - Test!
3. Bonus
   - Circular FIFO
   - Heap
   - Arbitrary rank order
   - Gap support
   - Support more media types

Problem Definition



For this tutorial, we are going to write a GStreamer element that computes the maximum of the previous n elements.

Problem Definition

Applications

The sliding maximum is the simplest example of a *rank order filter*, a handy signal processing tool with a lot of applications:

- Welch spectral estimator and related methods
- Smoothing lossily decompressed signals (especially images)
- Robust parameter estimation
- Detrending
- Outlier rejection

Code Walkthrough

1. Problem Definition
   - Applications
2. Code Walkthrough
   - Element skeleton
   - Add pad templates and caps
   - Adding properties
   - Pause to admire your work
   - Implement desired behavior
   - Test!
3. Bonus
   - Circular FIFO
   - Heap
   - Arbitrary rank order
   - Gap support
   - Support more media types

Element skeleton

Choosing a base class

First, choose a GStreamer base class that matches your problem.

Base class	Uses	Examples
`GstElement`	Any number of pads	`tee`, `lal_skymap`
`GstBaseSrc`	One source pad	`audiotestsrc`, `lal_framesrc`
`GstBaseSink`	One sink pad	`alsasink`, `lal_gracedbsink`
`GstBaseTransform`	One sink, one source pad	`audioiirfilter`, `lal_whiten`
`GstBin`	Contains other elements	`playbin`, `lal_fakeligosrc`
`GstPipeline`	A special bin present in every GStreamer app	N/A

Element skeleton

Choosing a base class

For this project, `GstBaseTransform` is a good match. We have a single input and a single output, and there is exactly one output sample for every input sample.

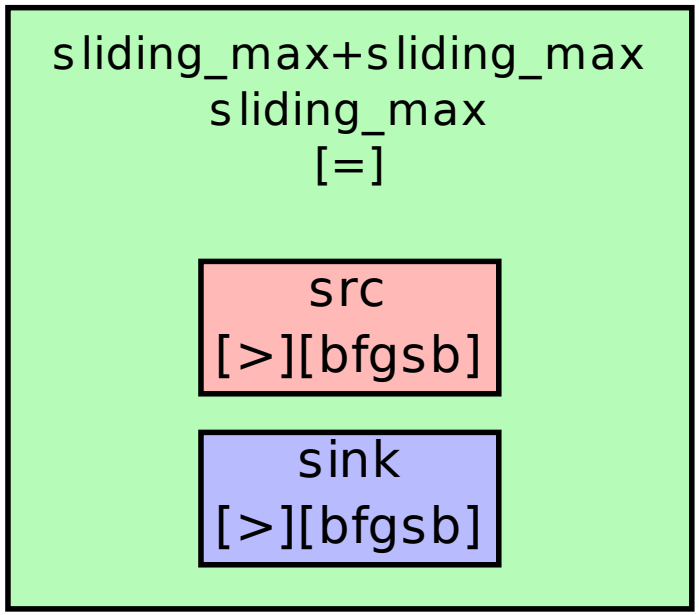


Figure: An instance of the new element class `sliding_max`, derived from `GstBaseTransform`

Element skeleton

Getting started

If you are writing an element class that will only be used in one application, then you can define the class in any Python module and instantiate it the same way as any other Python class.

If your element may be used again, you might want to put it into a `plugin` so that it can be used by any GStreamer application.

For example, putting an element in a plugin makes it so that you can use it with `gst-launch`:

```
$ gst-launch audiotestsrc freq=400 ! sliding_max n=8 ! autoaudiosink
```

This requires just a little bit of boilerplate code, much of which is streamlined by the `gstlal.pipeutil` module.

Element skeleton

Getting started

Start a file called `sliding_max.py` in the directory `~/.gstreamer-0.10/plugins/python/` . You may have to create this directory.

This is one of a few places that GStreamer automatically looks for Python elements.

(In the `gstlal` source code, Python elements are kept in `gst/python/`. These get installed to `${PREFIX}/lib/gstreamer-0.10/python/`.)

Element skeleton

Code listing

Start out with the following minimal subclass of `GstBaseTransform`:

```
from gstlal.pipeutil import *

class sliding_max(gst.BaseTransform):
        __gstdetails__ = (
                "Sliding maximum",
                "Filter",
                "Sliding maximum of past n samples",
                "Albert Einstein <albert.einstein@ligo.org>"
        )

# Register element class
gstlal_element_register(sliding_max)
```

Listing 1: Element skeleton: `sliding_max.py`

Add pad templates and caps

```
from gstlal.pipeutil import *

class sliding_max(gst.BaseTransform):
        __gstdetails__ = (...)
        __gsttemplates__ = (
                gst.PadTemplate("sink",
                        gst.PAD_SINK, gst.PAD_ALWAYS,
                        gst.caps_from_string("""
                                audio/x-raw-float,
                                endianness = (int) BYTE_ORDER,
                                width = (int) {32, 64},
                                channels = (int) 1
                        """)
                ),
                gst.PadTemplate("src",
                        gst.PAD_SRC, gst.PAD_ALWAYS,
                        gst.caps_from_string("""
                                audio/x-raw-float,
                                endianness = (int) BYTE_ORDER,
                                width = (int) {32, 64},
                                channels = (int) 1
                        """)
                )
        )

# Register element class
gstlal_element_register(sliding_max)
```

Listing 2: Adding caps

Adding properties

Code listing

```
from gstlal.pipeutil import *

class sliding_max(gst.BaseTransform):
        __gstdetails__ = (...)
        __gsttemplates__ = (...)
        __gproperties__ = {
                'n': (
                        gobject.TYPE_UINT,
                        'Window length',
                        'Number of samples in sliding window',
                        1, gobject.G_MAXUINT, 16, # min, max, default
                        gobject.PARAM_READWRITE | gobject.PARAM_CONSTRUCT
                ),
        }

        def do_set_property(self, prop, val):
                """gobject->set_property virtual method."""
                if prop.name == 'n':
                        self.n = val

        def do_get_property(self, prop):
                """gobject->get_property virtual method."""
                if prop.name == 'n':
                        return self.n

# Register element class
gstlal_element_register(sliding_max)
```

Listing 3: `sliding_max.py` with properties added

Pause to admire your work

Checking caps, properties with gst-inspect

```
$ gst-inspect sliding_max
Factory Details:
  Long name:    Sliding maximum
  Class:        Filter
  Description:  Sliding maximum of past n samples.
  Author(s):    Albert Einstein <albert.einstein@ligo.org>
  Rank:            none (0)

Plugin Details:
  Name:           python
  Description:    loader for plugins written in python
  Filename:       xxx/gstreamer-0.10/libgstpython.so
  Version:        0.10.19.1
  License:        LGPL
  Source module:  gst-python
  Binary package: GStreamer Python Bindings
  Origin URL:     http://gstreamer.freedesktop.org

GObject
 +----GstObject
       +----GstElement
             +----GstBaseTransform
                   +----sliding_max+sliding_max
```

```
Pad Templates:
  SRC template: 'src'
    Availability: Always
    Capabilities:
      audio/x-raw-float
             endianness: 1234
                  width: { 32, 64 }
               channels: 1

  SINK template: 'sink'
    Availability: Always
    Capabilities:
      audio/x-raw-float
             endianness: 1234
                  width: { 32, 64 }
               channels: 1

...

Element Properties:
  name                : The name of the object
                        flags: readable, writable
                        String. Default: null Current: "
  qos                 : Handle Quality-of-Service events
                        flags: readable, writable
                        Boolean. Default: false Current:
  n                   : Number of samples in sliding win
                        flags: readable, writable
                        Unsigned Integer. Range: 1 - 429
```

Implement desired behavior

Code listing

```
from gstlal.pipeutil import *
from gstlal import pipeio
import numpy

class sliding_max(gst.BaseTransform):
        ...

        def do_start(self):
                """GstBaseTransform->start virtual method."""
                self.history = []
                return True

        def do_transform(self, inbuf, outbuf):
                """GstBaseTransform->transform virtual method."""

                # Convert received buffer to Numpy array.
                x = pipeio.array_from_audio_buffer(inbuf)

                # Create output array.
                y = numpy.zeros(x.shape, dtype=x.dtype)

                # Do the dirty work.
                for i, xi in enumerate(x):
                        self.history.append(xi)
                        while len(self.history) > self.n: self.history.pop(0)
                        y[i] = max(self.history)

                # Copy output to buffer.
                outbuf[:len(y.data)] = y.data

                # Done!
                return gst.FLOW_OK
```

Test!

This simple pipeline will produce two ASCII data files, `x.txt` and `y.txt`.

```
$ gst-launch \
        audiotestsrc wave=pink-noise num-buffers=1 samplesperbuffer=128 \
        ! audio/x-raw-float,rate=16384,width=64 ! tee name=in_tee \
        \
        in_tee. ! queue ! lal_nxydump ! filesink location=x.txt sync=false \
        in_tee. ! queue ! sliding_max n=4 ! lal_nxydump ! filesink location=y.txt sync=false
```

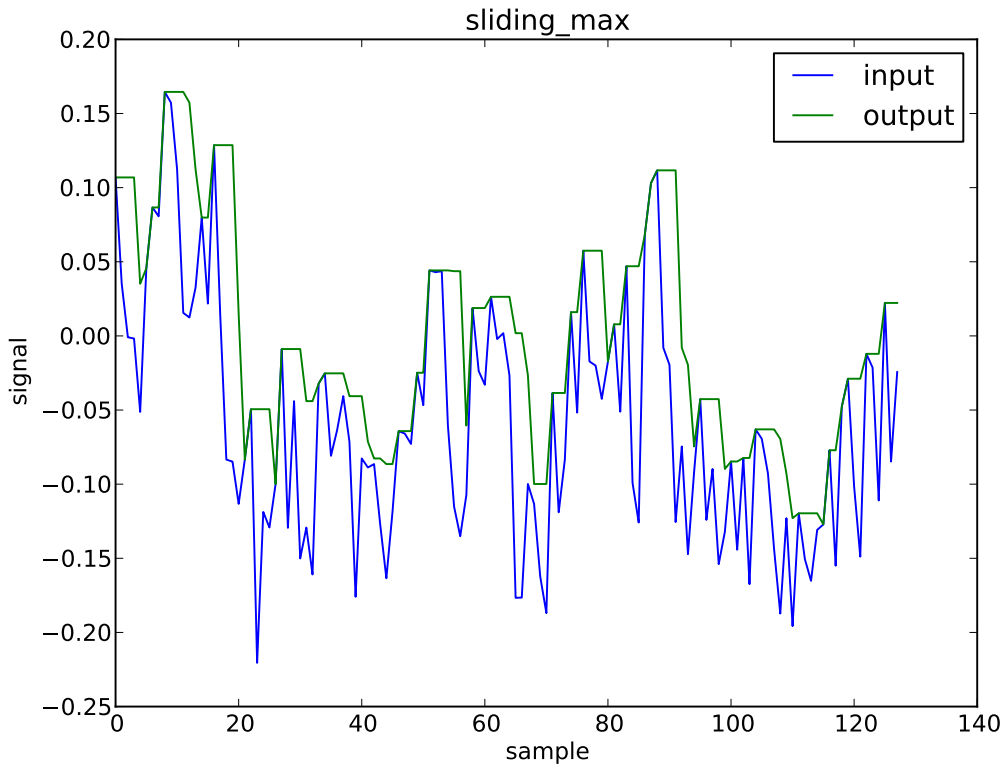


Figure: Filter input, `x.txt`, and output, `y.txt`, plotted with `matplotlib`.

Bonus

For the remainder of this tutorial, we encourage you to try one of the 'Bonus' projects.
You can work alone, or with a partner, or in small groups.
The first two are designed to illustrate the differences between conventional batch algorithms and algorithms that are well suited for stream processing or low-latency/realtime applications.
The last bonus three bonus projects are designed to teach you more about GStreamer.

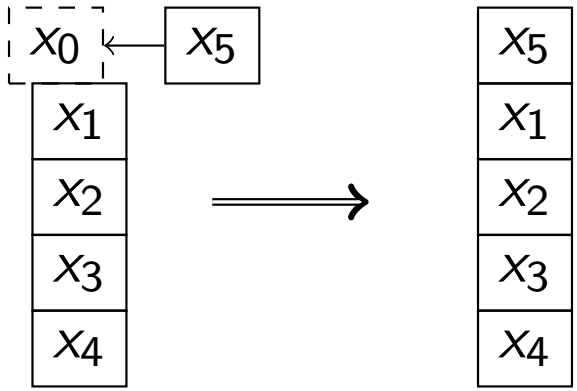
Bonus

Circular FIFO

If you are worried about the performance impact of storing the filter's history in a Python list, *good!*

With a Python list, it takes $\mathcal{O}(n)$ operations to replace the oldest sample in the history.

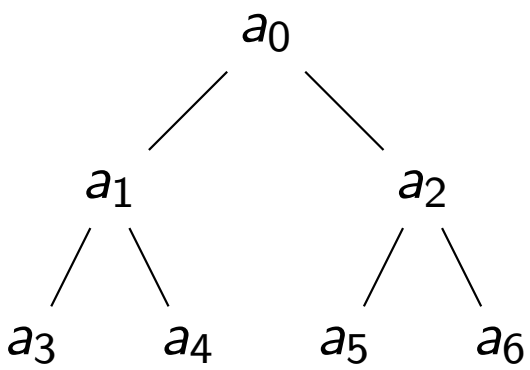
If we use a *circular FIFO* or *ring buffer* instead, updating the history takes $\mathcal{O}(1)$ operations. Circular FIFOs occur often in the Linux kernel and also in DSP applications.



The Python class `collections.deque` can be used as a circular FIFO.

Bonus

Heap



A heap is another data structure that is useful for this problem. A (max-)heap is a binary tree in which every node is greater than either of its children.
A heap is often stored in an array, so one can write this condition as:

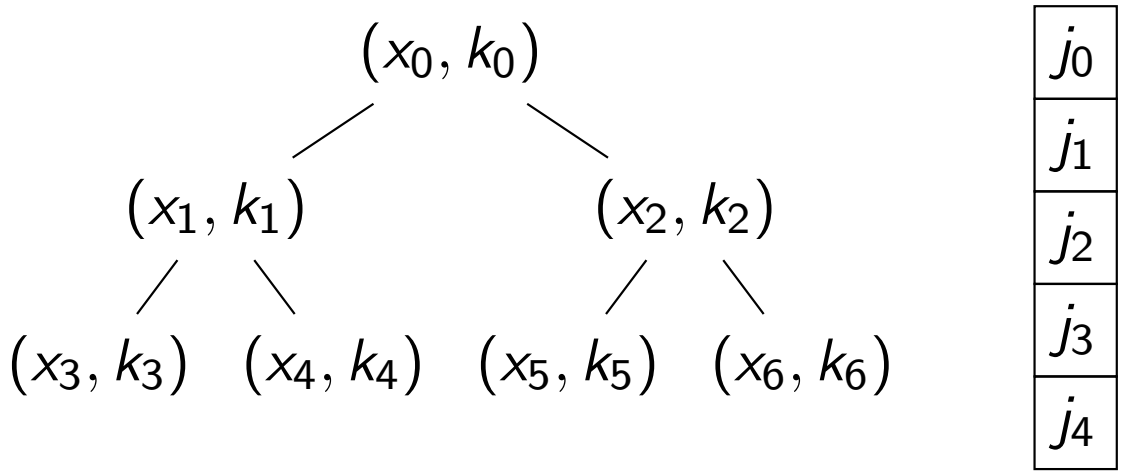
$$\forall k : a[k] > a[2k+1],\ a[k] > a[2k+2]$$

The maximum value in a heap is always the value at the root, a_0.

Bonus

Heap

What we want is a combination of a (max-)heap and a ring buffer.



Each entry in the heap is an ordered pair consisting of a value from the input stream, and its index in the input stream.

Each entry in the ring buffer refers to one of the past n samples of the input stream in chronological order. But the ring buffer does not store the input samples themselves — it stores their ranks in the heap.

Bonus

Heap

Using this combination of a heap and a circular FIFO, deletion of the oldest element is at worst $\mathcal{O}(\log n)$, and so is insertion of a new element.

However, on average, both operations are $\mathcal{O}(1)$. (Why? Hint: about half of the elements in a heap are leaves.)

For this project, write a version of `sliding_max` that uses this or a similar data structure to implement the sliding maximum in $\mathcal{O}(1)$ operations per sample.

Bonus

Heap

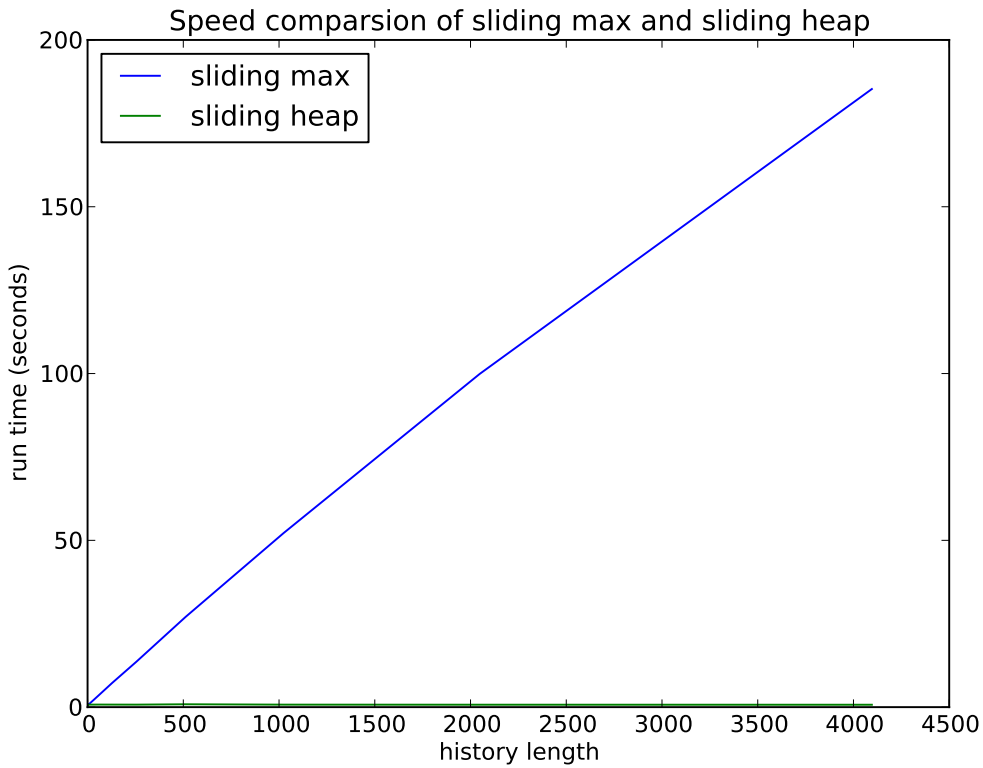


Figure: Running time comparison of sliding max and sliding heap for various history lengths n. Notice that the running time of the sliding max increases linearly with history length, and yet the sliding heap is unaffected.

Bonus

Arbitrary rank order

For this project, turn your element into a general rank-order filter.

Add a new property, k. For $k < n$, your element should return the kth largest number from the past n samples of history. ($k = 0 \Rightarrow$ minimum, $k = n - 1 \Rightarrow$ maximum)

Once you have done this, your element can now serve as median filter, or it can also compute the upper or lower quartile, ...

Bonus

Gap support

A *gap buffer* is a special buffer that is understood to contain neutral data, or 'silence'. Some elements, especially codecs, are able to process gap buffers differently in order to speed quickly over them.

For this project, add gap support to your element. Assume that a gap consists of zeros. Here's a skeleton to get you started:

```
def __init__(self):
        super(sliding_max, self).__init__()
        self.set_gap_aware(True)

def do_transform(self, inbuf, outbuf):
        ...
        if inbuf.flag_is_set(gst.BUFFER_FLAG_GAP):
                ... # Special case for gap buffers
        else:
                ... # Special case for normal buffers
        ...
```

Hint: what effect does a gap of n samples have on the filter's history?

Bonus

Support more media types

The original version of `sliding_max` only supports single-channel single or double precision floating point streams.

For this project, modify your element to accept more media types, including:

- integers of different widths (8, 16, 32, 64)
- signed or unsigned
- multiple channels

This will involve modifying the caps that the pads accept.

```
__gsttemplates__ = (
        gst.PadTemplate("sink",
                gst.PAD_SINK, gst.PAD_ALWAYS,
                gst.caps_from_string("""
                        audio/x-raw-float,
                        endianness = (int) BYTE_ORDER,
                        width = (int) {32, 64},
                        channels = (int) 1;
                        audio/x-raw-int, ...
```