CS100
Introduction to Programming

Tutorial 7: lists, vectors, and timing

Problem 1
std::chrono

Get the time and measure times with std::chrono

- Header:

```
#include <chrono>
```

- Use

```
std::chrono::high_resolution_clock
```

- Figure out how to
  - Get time-points
  - Extract durations
  - Express durations in microseconds, and cast them to an int

Get the time and measure times with std::chrono

- Test the time/duration functionality
- You may use a dummy function to consume time

```
// long operation to time
int dummyFunction(int n) {
  if (n < 2) {
    return n;
  } else {
    return fib(n-1) + fib(n-2);
  }
}
```

Problem 2

Implement a Timer class

Implement a Timer class

- Lap timer to accumulate iteration times!
- Interface:

```
class Timer {
public:
  Timer( bool start = false );
  virtual ~Timer();

  void start();
  void stop( bool restart = false );
  void stop( size_t iterations,
      bool restart = false );
  void reset();

  double averageTime();
  std::list<double>::iterator begin();
  std::list<double>::iterator end();
  …
};
```

Implement a Timer class

- What private variable to choose?
- What type should the lap time container be?

- begin():
  - Iterator to first element in lap-time container
- end():
  - Iterator to last element in lap-time container

Test the Timer class

- Use **`dummyFunction()`**

Problem 3
Filling lists and vectors

Define a large object

```
class LargeObject {
public:
  LargeObject();
  virtual ~LargeObject();
private:
  int m_data[1000000];
};

LargeObject::LargeObject() {};
LargeObject::~LargeObject() {};
```

Measure filling times

- Measure times for putting elements into a list and a vector!
  - Fill 500 **`LargeObject`**s into a list and a vector
  - Measure the time of each iteration with a **`Timer`**

- Print all times into the console
- What can you observe?

Problem 4

The mean and the median

The arithmetic mean

$$\begin{aligned}\Delta t_{\text{avg}} &= \frac{1}{n}\sum_i \Delta t_i \\ &= \underset{\Delta t_{\text{avg}}}{\operatorname{argmin}} \sum_i \|\Delta t_i - \Delta t_{\text{avg}}\|^2\end{aligned}$$

- Can you observe a difference in the average time it takes to put a LargeObject into a vector or list?

Getting a robust measure of time-complexity

- How to obtain a measure of the time complexity regardless of occasional copying?
- → Median!

$$\Delta t_{\mathrm{avg}} = \underset{\Delta t_{\mathrm{avg}}}{\operatorname{argmin}} \sum_i |\Delta t_i - \Delta t_{\mathrm{avg}}|$$

How to compute the median?

- Try to compute the median of the values {2,1,9}
- Try to compute the median of the values {1,3,2,1,50}

- → Median can be computed by sorting and taking the ”center” value
- → Sort the lap-times, and print the mean and median times into the console

Getting a robust measure of time-complexity

Final output:

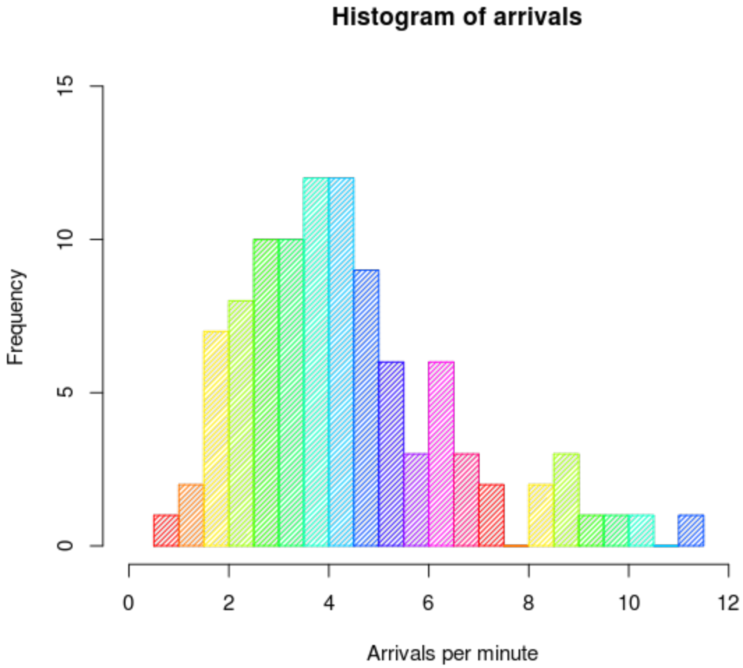
```
putting an element into a list takes in average 0.00167534 seconds.
putting an element into a vector takes in average 0.00380282 seconds.
the median time for putting an element into a list is 0.001603 seconds.
the median time for putting an element into a vector is 0.001601 seconds.
```

Problem 5

Histogram: Fast computation of the “center” value

What is a histogram?

- Estimates probability distribution of a continuous variable
- https://en.wikipedia.org/wiki/Histogram



Implement a histogram

- Implement a generic histogram class (template class)
- Include a function to approximate the median

Implement a histogram

- Interface:

```
template<class T>
class Histogram{
public:
  Histogram( T minVal = -1, T maxVal = 1, size_t bins = 20 );
  virtual ~Histogram();

  void insert( T val );
  size_t getRelevantBin( T val );
  void incrementRelevantBin( size_t bin, size_t increment = 1 );
  void clear();

  size_t binCount( size_t bin );
  size_t totalCount();
  T approximateMedian();
…
};
```

Test the histogram

- Generate 1000 (uniformly distributed) random numbers between -1 and 1:

```
double getRandomNumber() {
  return (((double) rand())/ ((double) RAND_MAX)-0.5)*2.0;
}
```

- Fill a histogram, and print the bin values and the median! Example output:

```
45 51 51 46 47 50 39 54 58 59 64 47 39 71 52 42 55 41 47 42

0.05
```