

**CITS2200**  
**Data Structures and Algorithms**

Cara MacNish

School of Computer Science & Software Engineering  
University of Western Australia

## Topic 6

# Performance Analysis 1: Introduction

- Why analyse performance?
- Types of performance measurement
  - empirical
  - simulational
  - analytical
- An example of analytical analysis using Queue
- Introduction to growth rates

**Reading:** Lambert and Osborne, Section 4.1.

## 6.1 Educational Aims

The aims of this topic are to:

1. begin thinking about the **implications of the choices you make** for ADT performance
2. introduce simple metrics for assessing algorithm performance, which will later lead to mathematically-based techniques

## 6.2 Why performance analysis?

- Comparison
  - choice of ADT
  - choice of implementation
  - trade-offs — may be no clear winner/depend on calling program
- Improvement
  - identification of expensive operations, bottlenecks
  - improved implementations within ADTs
  - improved implementation of calling programs

## 6.3 Types of Performance Measurement

### Empirical measurement

We will see that the most efficient queue ADT to use depends on the program that uses it — which operations are used most often.

If we have access to the program(s), we may be able to measure the performance in those programs, on real data — called **evaluation in context**.

This is the “get yer hands dirty” approach. Run the system with real-world input and observe, or monitor (automatically), the results.

Can compare data structures on the same problems (same machine, same compiler, etc)

⇒ **benchmark** programs

- Useful if test input is close to expected input.
- Not much use if we are developing eg a library of modules for use in many different contexts

## Simulational Measurement

Construct a (computer) model of system and evaluate performance with simulated data.

eg. US nuclear weapons defence system

A computer program normally acts as its own model — run on simulated data (often generated using pseudo-random numbers)

However a simplified model may be built, or the program modified to fit the simulated data.

## **Advantages**

- nondestructive
- cheap (?)
- fast (?)

## **Disadvantages**

- only as good as the simulations
- can never be sure it matches reality

## Analytical Measurement

Construct a mathematical or theoretical model — use theoretical techniques to estimate system performance.

Usually

- coarse estimates
- growth rates, complexity classes rather than ‘actual’ time
- **worst case** or **average case**

But...!

- **fundamental view of behaviour** — less susceptible to
  - speed of hardware, number of other processes running, etc
  - choice of data sets
  - unrepresentative examples, spurious responses
- gives a better understanding of the problems
  - why is it slow?
  - could it be improved?

We will concentrate on analytical analyses.

## 6.4 Example: A Basic Analysis of the Queue ADTs

As an example of comparison of ADT performance we consider two representations of queues — block (without wraparound) and recursive — using a crude time estimate

Simplifying assumptions:

- each high-level operation (arithmetic operation, Boolean operation, subscripting, assignment) takes 1 time unit
- conditional statement takes 1 time unit + time to evaluate Boolean expression + time taken by most time consuming alternative (**worst-case** assumption)

- field lookup (“dot” operation) takes 1 time unit
- method takes 1 (for the call) plus 1 for each argument (since each is an assignment)
- creating a new object (from a different class) takes  $T_c$  time units

## 6.4.1 Block representation queues (without wraparound)

```
public QueueCharBlock (int size) { //2
    items = new char[size]; //1+Tc
    first = 0; //1
    last = -1; //1
}
```

5 +  $T_c$  time units

```
public boolean isEmpty () {return first == last + 1;}
```

4 time units

```
public boolean isFull () {return last == items.length - 1;}
```

5 time units

```
public void enqueue (char a) throws Overflow {           //2
    if (!isFull()) {                                     //7
        last++;                                         //1
        items[last] = a;                               //2
    }
    else throw new Overflow("enqueueing to full queue");
}
```

12 time units

## Exercise:

How many time units for each of the following...

```
public char examine () throws Underflow {
    if (!isEmpty()) return items[first];
    else throw new Underflow("examining empty queue");
}
```

```
public char dequeue() throws Underflow {
    if (!isEmpty()) {
        char a = items[first];
        first++;
        return a;
    }
    else throw new Underflow("dequeuing from empty queue");
}
```

## Summary for Block Implementation

`isEmpty`, `enqueue`, `examine` and `dequeue` are constant time operations

`Queue` is constant time if  $T_c$  is constant time

## 6.4.2 Recursive (linked) representation queues

```
public QueueCharLinked () {  
    first = null;  
    last = null;  
}
```

3 time units

```
public boolean isEmpty () return first == null;
```

3 time units

```

public void enqueue (char a) { //2
    if (isEmpty()) { //4
        first = new LinkChar(a,null); //1+Tc
        last = first; //1
    }
    else {
        last.successor = new LinkChar(a,null); //2+Tc
        last = last.successor; //2
    }
}

```

$10 + T_c$

```

public char examine () throws Underflow {
    if (!isEmpty()) return first.item;
    else throw new Underflow("examining empty queue");
}

```

8 units

```
public char dequeue () throws Underflow { //1
    if (!isEmpty()) { //5
        char c = first.item; //2
        first = first.successor; //2
        if (isEmpty()) last = null; //5
        return c; //1
    }
    else throw new Underflow("dequeuing from empty queue");
}
```

16 units

## Summary for Linked Implementation

Again all are constant time, assuming  $T_c$  is.

## Comparison...

	block	recursive
Queue	$5 + T_c$	3
isEmpty	4	3
enqueue	12	$10 + T_c$
examine		8
dequeue		16

... shows no clear winner, especially given

- estimates are very rough — many assumptions
- dependent on relative usage of operations in the programs calling the ADT — eg. is `isEmpty` used more or less than `dequeue`

We will generally not be interested in these “small” differences (eg 5 units vs 3 units) — given the assumptions made these are not very informative.

Rather we will be interested in **classifying** operations according to **rates of growth**...

## 6.5 Growth Rates

**Q:** If it takes 2 hours to roast a turkey, how long does it take to cook a mammoth?

**A:** Need to make some assumptions...

**Chef 1:** Turkey 10kg, Mammoth 1000kg

Temporal-calorific-multiplier  $c = \frac{2}{10} = 0.2$  hrs/kg

Cooking time  $t = 0.2 \times 1000 = 200$  hrs

## **Chef 2:** Turkey 10kg, Mammoth 1000kg

A little more distance for the heat to penetrate takes a lot more heat. In fact each centimetre of radius takes 6 times as long as the previous one. Radius increases with approx cube root of volume, so cook time increases with square of volume. Volume is proportional to mass. Therefore cook time increases with square of mass.

$$\text{Temporal-calorific-multiplier } c = \frac{2}{10^2} \text{ hrs/kg}^2$$

$$\text{Cooking time } t = \frac{2}{100} \times 1000^2 = 20\,000 \text{ hrs}$$

### **Chef 3:** Turkey 10kg, Mammoth 1000kg

Heat is hanging around in the rest of the unused oven space anyway, although for bigger animals you need a bigger oven, which is slower to heat up.

Doubling the mass only adds a constant amount to the cooking time.

Temporal-calorific-multiplier  $c = \frac{2}{\log 10}$

Cooking time  $t = \frac{2}{\log 10} \times \log(1000) = 6$  hrs

For comparative purposes exact numbers are pretty irrelevant! It is the **rate of growth** that is important.

We will abstract away from inessential detail...

- ignore specific values of input and just consider the number of items, or “size” of input
- ignore precise duration of operations and consider the number of (specific) operations as abstract measure of time
- ignore actual storage space occupied by data elements and consider number of items stored as abstract measure of space

## 6.6 Summary

Three types of performance measurement — empirical, simulational, analytical.

We will concentrate on analytical:

- fundamental view of behaviour
- abstracts away from machine, data sets, etc
- helps in understanding data structures and their implementations

Rather than attempting ‘fine grained’ analysis, comparing small differences, we will concentrate on a coarser (but more robust) analysis in terms of **rates of growth**.