ALGORITHM DESIGN & ANALYSIS
DR NICOLAS WU
#
1. INTRODUCTION

"It is not only the violin that shapes the violinist, we are all shaped by the tools we train ourselves to use, and in this respect programming languages have a devious influence: they shape our thinking habits."

— E. W. Dijkstra, 2001

https://materials.doc.ic.ac.uk/resources/2122/50001

LECTURES:

On Zoom, see link in materials & your calendar

Mondays	12:00 – 13:00
Wednesdays	09:00 – 10:00
Thursdays	15:00 – 16:00

RECORDINGS

Lecture recordings will be on panopto 24 hrs after a lecture.

EXERCISES

There will be weekly exercise sheets.
Seminar sessions will go over material
These are part of the scheduled lectures.
All solutions will be released.

COURSEWORK

There are 2 courseworks:

CW1 - Week 6-7
CW2 - Week 9-10

These are auto-marked & individually assessed.
They account for 15% of your final grade

FURTHER READING

sorted list ✓ sorted list ✓

```
> insert :: Int → [Int] → [Int]
> insert x []     = [x]
> insert x (y:ys)
>     | x ≤ y     = x : y : ys
>     | otherwise = y : insert x ys
```

How expensive is this function?

(1) Accurate: reduction steps
(2) Approximate: recurrence relation

Method 1:

Accurate:

```
   insert 4 [1, 3, 6, 7, 9]
⇝
   1 : insert 4 [3, 6, 7, 9]
⇝
   1 : 3 : insert 4 [6, 7, 9]
⇝
   1 : 3 : 4 : 6 : [7, 9]
```

Method 2 : Approx:

sorted list ↓ sorted list ↓

```
> insert :: Int → [Int] → [Int]
> insert x []     = [x]
> insert x (y:ys)
>     | x ≤ y     = x : y : ys
>     | otherwise = y : insert x ys
```

The cost of insert x ys is:

$$T_{insert}(0) = 1$$
$$T_{insert}(n) = T_{insert}(n-1) + 1$$

where n = length ys

$$T_{insert}(n) = 1 + T_{insert}(n-1)$$
$$= 1 + (1 + T_{insert}(n-2))$$
$$= 1 + (1 + (1 + T_{insert}(n-3)))$$
$$= \underbrace{1 + \dots + 1}_{n} + \underbrace{T_{insert}(0)}_{1}$$

$$T_{insert}(n) = n + 1$$

```
isort :: [Int] → [Int]
isort  []   = []
isort  (x:xs) =  insert  x   (isort xs)
```

(1)

```
   isort [3, 1, 2]
⇝ { def isort }
   insert 3 (isort [1,2])
⇝ { def isort }
   insert 3 (insert 1 (isort [2]))
⇝ { def isort }
   insert 3 (insert 1 (insert 2 []))
⇝ { def insert }
   insert 3 (insert 1 ( [2] ))
⇝ { def insert }
   insert 3 ( 1 : 2 : [] )
⇝ { def insert }
   1 : insert 3 [2]
⇝ { def insert }
   1 : 2 : insert 3 []
⇝ { def insert }
   1 : 2 : [3]
```

8 reductions

(2)

```
isort :: [Int] → [Int]
isort []     = []
isort (x:xs) = insert x (isort xs)
```

The cost of isort xs where n = length xs is $T_{isort}(n)$ where:

$$T_{isort}(0) = 1$$

$$T_{isort}(n) = 1 + T_{insert}(n-1) + T_{isort}(n-1)$$

$$= \underbrace{(1+n) + (1+(n-1)) + (1+(n-2)) + \cdots}_{\frac{n(n+1)}{2}} \quad \underbrace{\uparrow \uparrow \cdots}_{n}$$

$$\frac{n^2}{2} + \frac{3n}{2} + 1$$