Data structures

Exercise session – Week 1

I. Introduction

Two kinds of types in Java

- Primitive types

  int, char, boolean, float, double, etc.

- Object types

  Integer, Character, String, etc.

Java generics

Before generics you had to write:

```
reverse(Integer[] arr)
reverse(String[] arr)
reverse(Float[] arr)
…
```

Now you can write:

```
reverse(E[] arr)
```

Gotcha!
E *must* be an object type!

Two ways to reverse

- *functional*

  ```
  <E> E[] reverse(E[] str)
  ```

- *in-place*

  ```
  <E> void reverse(E[] str)
  ```

Which way is the better way?

STAR
WARS

IsPalindrome?

Using functional reverse, it's *one* line

```
reverse (chars).equals(chars)
```

(where chars is a Character[])

Time Complexity

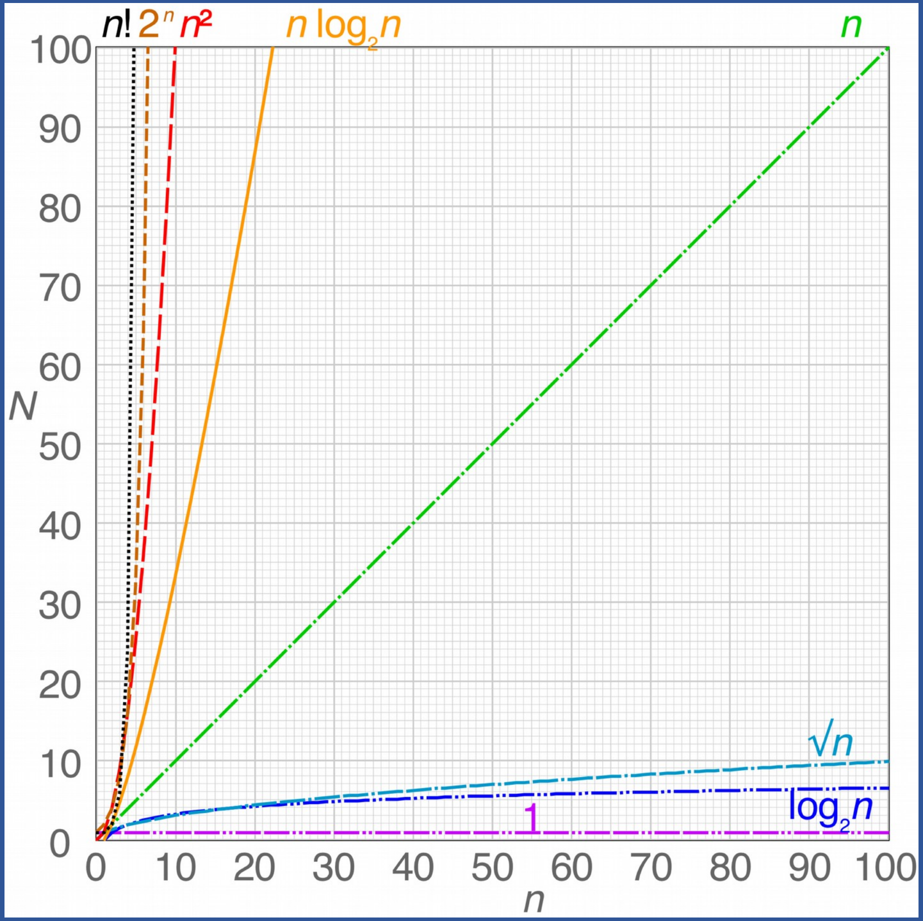
How many times does *result++* run?

```
for (int j = 1; j <= n; j++)
   result++;
```

How many times does *result++* run?

```
for (int i = 1; i <= n; i ++)
   for (int j = 1; j <= n; j++)
      result++;
```

Typical growth functions



”Big-O” time complexity

A function *t(n)* is classified as O(*f(n)*)

, for some function *f(n)*

, if there exists some +ve *c* and *n’*

, such that *t(n)* <= *c* * *f(n)* when n >= *n’*

Exercise!

Arrange the following functions in order of complexity:

n^4, $\log n$, $n \log n$, $4n$, $3n^3$, $5n^2 + n$.

How many times does *result++* run?

```
for (int i = 1; i <= n; i ++)
   for (int j = 1; j <= i; j++)
      result++;
```

How many times does *result++* run?

```
for (int j = 1; j <= n; j *= 2)
    result++;
```

How many times does *result++* run?

```
for (int i = 1; i <= n; i *= 2)
   for (int j = 1; j <= n; j++)
result++;
```

How many times does *result++* run?

```
for (int i = 1; i <= n; i *= 2)
   for (int j = ; j <= i; j++)
result++;
```

Run times for binary search?!

Input size n	Execution time
10	8.9
100	17.2
1000	49.5
10000	52.2
100000	60.1
1000000	70.5
10000000	199.0

Still O(log n)! Why?

10 * log(n),	**Execution time**
33.21	8.9
66.43	17.2
99.65	49.5
132.87	52.2
166.09	60.1
199.31	70.5
232.53	199.0

Reading

- Weiss, 1.5.8 - *Restrictions on Generics*
- Weiss, 2 – Algorithm Analysis