# Algorithms

COSC 1010

# Algorithms

- Algorithm - A clear specification of how to solve a problem
  - Effective method:
    - Measured in the resources solved in dependence of the size of the problem
      - Time
      - Memory
      - ...

# Algorithms

- Algorithms can be very different, yet still solve the same problem
  - Example:  Given an ordered list of numbers decide whether a number is in the list or not.
    - E.g. list is [3, 5, 7, 11, 13, 17, 19, 21, 23, 29, 31] (the first ten odd primes) and 25 is there

# Algorithms

- Algorithm Number 1:
  - Take the number (25) and compare it in turn with all elements of the list. If there is equality, remember it.
  - E.g. 25 in [3, 5, 7, 11, 13, 17, 19, 21, 23, 29, 31]?
    - Is 25 == 3?
    - Is 25 == 5?
    - Is 25 == 7?
    - …
    - Is 25 == 31?
  - Only got no-s, so 25 is not in the list.

# Algorithms

- Algorithm 2
  - Take the number (25) and compare it in turn with all elements of the list. If there is equality, report True immediately. Otherwise, report False after having compared all elements.
- E.g. 25 in [3, 5, 7, 11, 13, 17, 19, 21, 23, 29, 31]?
  - Is 25 == 3?
  - Is 25 == 5?
  - Is 25 == 7?
  - …
  - Is 25 == 31?
- Finished with the list, so 25 is not in the list.

# Algorithms

- Algorithm 3
  - Take the number (25) and compare it in turn with all elements of the list. If there is equality, report True immediately. Otherwise, if the list element is larger or we are at the end of the list, report False.
- E.g. 25 in [3, 5, 7, 11, 13, 17, 19, 21, 23, 29, 31]?
  - Is 25 == 3? No.
  - Is 25 == 5? No.
  - Is 25 == 7? No.
  - …
  - Is 25 == 29? No, and we can stop since 29 > 25.
- Report False

# Algorithms

- Algorithm 4:
  - If the list is empty, report False.
  - If the list has only one element, compare and decide directly.
  - Compare the number with the middle element of the list. If there is equality, report True. If the number is smaller, repeat the procedure for the first half of the list. If the number is greater, repeat the procedure for the second half of the list.

# Algorithms

- E.g. 25 in [3, 5, 7, 11, 13, 17, 19, 21, 23, 29, 31]?
  - Step 1: Compare 25 and 17. 25 is larger.
  - Step 2: Is 25 in [19, 21, 23, 29, 31]?
  - Step 3: Compare 25 with 23. 25 is larger.
  - Step 4: Is 25 in [29, 31]?
  - Step 5: Compare 25 with 29.  (There is a tie so we break it arbitrarily.) 25 is smaller, so
  - Step 6: Is 25 in [  ] (the empty list)?
  - Step 7: Report False

# Algorithms

- Set $n$ = length(list).
- Algorithm 1 always has $n$ comparisons.
- Algorithm 2 makes $n$ comparisons if the element is not in the list. If the element is in the list and **if** we assume that the elements are uniformly distributed between the min and the max of the list, then on average we make ~ $n/2$ comparisons.
- Algorithm 3 makes ~ $n/2$ comparisons, regardless of whether the number is in the list or not
- Algorithm 4 makes about $\log_2(n)$ comparisons

# Algorithms

- Does this mean that Algorithm 4 is always faster?
  - No, but since the log grows so much slower than *n*, for large lists, it will be better

# Algorithms

- Calculating the Variance
  - Average Squared Difference between the average and the numbers in a list
  - Average $\mu = \frac{x_1 + x_2 + \ldots + x_n}{n}$
  - Variance $\sigma^2 = \frac{(x_1 - \mu)^2 + (x_2 - \mu)^2 + \ldots + (x_n - \mu)^2}{n}$

# Algorithms

- Algorithm 1:
  - Add up all elements in the list and then divide by the number of elements to obtain $\mu$
  - Walk through the list again:
    - For each element, calculate the square of the difference of the element with $\mu$
    - Add up these values and divide by *n* in order to obtain $\sigma^2$

# Algorithms

- Algorithm 2
  - Walk through the list once, adding up two values:
    - The numbers themselves, obtaining $S(x^2)$
    - The square of the numbers, obtaining $S(x)$
  - Then calculate $$\sigma^2 = \frac{S(x^2)}{n} - \frac{S(x)^2}{n^2}$$

# Algorithms

- If the list is long, then going through the list twice is costly
  - Because the list does not fit in the processor cache, some time is spent fetching it to and from memory or - even worse - from SSD or disk
- Algorithm 2 seems therefore to be better:
  - It walks through the list only once.
  - However, the sum of squares can become quite big, so big that it overflows the CPU registers.
    - Besides, calculating the difference between two large numbers induces rounding errors.

# Algorithms ≠ Programs

- An algorithm is ***implemented*** by a program
  - The difference between program and algorithm is fuzzy, but several implementations of the same algorithms exist.
- Often, an algorithm is given in “pseudo-code”