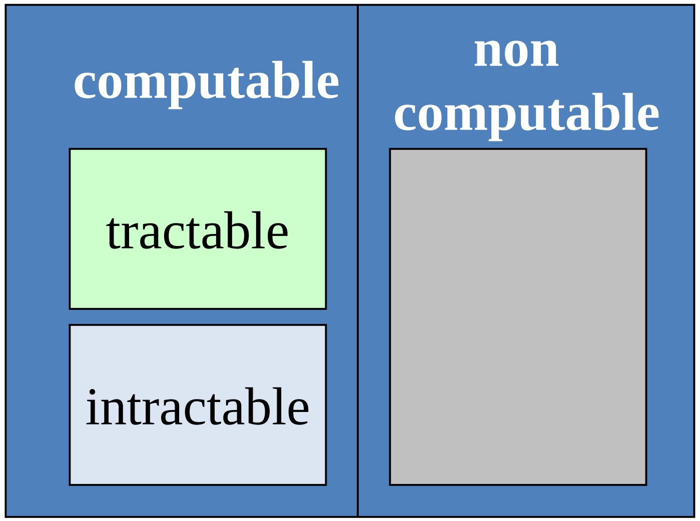
Time, Space & Complexity

- Problems are divided into classes
  - non-computable (icke beräkningsbara)
  - computable (beräkningsbara)
    - tractable (hanterliga)
      - **polynomial time**
      - **$\log_n$, n, $n\log_n$, n^2**
    - intractable (ohanterliga)
      - **non- polynomial time**
      - **2^n, $n!$, n^n**

- NP complete problems - class of intractable problems - use heuristics



Program running time

- Depends on
  - input e.g. # items in a sort
  - code quality generated by compiler
  - machine speed
  - time complexity of underlying algorithm
- Complexity functions may be defined
  - T(n) - running time
  - S(n) - space required
- Balance space / time
  - e.g. Hashing slots
- reduced T => higher S and vice versa

Best, average, worst case

- $T_{avg(n)}$, $T_{worst(n)}$, $T_{best(n)}$
  - what do these mean???
  - $T_{worst(n)}$ is usually given as a measure of T(n)
    - not always representative
- Complexity given as O(g(n)) “Big-Oh”
  - where T(n) <= c * g(n)
  - g(n) - growth rate
- For tractable algorithms
  - $O(\log_n)$, $O(n)$, $O(n*\log_n)$, $O(n^2)$, $O(n^3)$
- plot these values against n to compare algorithms
- E.g.

$T(n) = (n+1)^2 \Rightarrow \quad O(n^2)$

if $n_0 = 1$, $c=4$, $T(n) \le 4n^2$

Interpretation of n, T(n), O(n)

- **BE VERY CAREFUL HOW THESE ARE INTERPRETED !!!**
- **BE VERY CAREFUL THAT YOU UNDERSTAND HOW THESE HAVE BEEN ARRIVED AT !!!**
- **O(n) IS AN APPROXIMATION TO ALLOW COMPARISON OF ORDERS OF MAGNITUDE FOR DIFFERENT ALGORITHMS**
- **O(n) determines the size of a problem which can be solved using a computer**

  (everything is relative!)

Interpretation (Caveat emptor!)

- T(n) / O(n) - represent worst case
- one-off programs - choose simplest algorithm
- small input => O(n) may be less important than c (the constant) in T(n) <= c * g(n)
- beware of complex algorithms (KISS !!!)
- S(n) may also be a consideration (T(n) versus S(N))

- NB numerical accuracy may be more important than efficiency

Calculating T(n)

- For two program fragments P_1, P_2 with running times $T_1(n)$ and $T_2(n)$ respectively and $T_1(n)$ is $O(f(n))$ and $T_2(n)$ is $O(g(n))$ then

  the running time of P_1; P_2 (a sequence) is $O(\max(f(n), g(n)))$

hence a sequence of instructions may be calculated

General Rules for running times

- **Assignment** / read / write / usually O(1) (constant)
  exceptions: array assignment / assignment with function calls
- **sequence of statements** given by sum rule (i.e. max)
- **if-statement** cost of conditional evaluation -- O(1)
  + time of conditionally executed statements

  if-else-statement cost(conditional) + max(cost(true), cost(false))
- **loops** cost(termination condition evaluation)
  + sum(cost(loop body)) - often n*cost(loop body)
- **functions** sum of costs of each fn in calling sequence

  (non-recursive) (start with those without calls to other fns)

Recursive functions

- Associate T(n) (unknown) with each fn
- find a recurrence relationship for T(n)
- E.g.

```
int fact (n) {
  if  (n<=1) fact = 1;
  else fact = n * fact(n-1);
  }
```

- For some constants c & d

T(n) = c + T(n-1) if n>1
d if n<=1

T(1) = d
T(2) = c + d
T(3) = 2c + d
T(4) = 3c + d
T(i) = c(i-1) + d
T(n) = c(n-1) + d

hence conclude T(n) is O(n)