Übungsblatt 6

Thomas Graf
EF / WF Informatik 2018-2019
Algorithmen

22. Oktober 2018

1 Insertion Sort ausführen ($\star$)

Führe den Insertion Sort Algorithmus (Schritt für Schritt) auf der Eingabe $A = [31, 41, 59, 26, 41, 58]$ durch.

2 Grosse Unterschiede bei der Laufzeit ($\star\star$)

Auf der Webseite https://www.top500.org wird eine aktuelle Liste der Top 500 Supercomputer der Welt geführt.
Die aktuellste Version findet sich auf https://www.top500.org/lists/2018/06/.
Informiere Dich dort darüber, wie viele *floating point operations per second* (**FLOPS**) die Nummer 1 auf der aktuellen Liste maximal (*peak performance*) durchführen kann (*Rpeak*).
Ein Algorithmus A hat die Laufzeit $T_A(n) = n^2$ und ein Algorithmus B hat die Laufzeit $T_B(n) = n!$.
Welche Problemgrösse n kann der Supercomputer in einer Minute für Algorithmus A berechnen?
Wie viele Sekunden benötigt der Supercomputer für die Problemgrössen $n = 20$ bei Algorithmus B? Wie viele Sekunden bei $n = 32$ (auch für Algorithmus B)? Vergleiche den letzten Wert mit dem Alter des Universums.

3 Untersuchung eines Algorithmus ($\star\star$)

Betrachte folgenden Algorithmus in Pseudocode:

Data: Eine Liste $A\,[1...n]$.
Result: ???
for *i = 1 to A.length-1* **do**
    for *j = A.length to i+1* **do**
        if $A\,[j] < A\,[j-1]$ **then**
            exchange $A\,[j]$ with $A\,[j-1]$;
        end
    end
end

Algorithm 1: unbekannter Algorithmus

Beschreibe genau, was der Algorithmus macht.
Implementiere den Algorithmus in Python und führe ihn für verschiedene Eingaben aus.
Ist $T_{\text{worst-case}}(n)$ für diesen Algorithmus eine quadratische oder eine lineare Funktion in n?

4 Perfekte Zahlen (schnell) (⚛)

In der Challenge Aufgabe auf Übungsblatt 2 haben wir Python-Code geschrieben, welcher uns alle perfekten Zahlen in $[n_1, n_2]$ ausgegeben hat.

```python
def is_perfect_slow(n):
    """ decides if a given integer n is a perfect number or not
        this is the straightforward implementation
    """
    if n <= 0:
        return(False)
    sum = 0
    for i in range(1,n):
        if n % i == 0:
            sum += i
    return(sum == n)

def perfect_numbers(n1, n2, method = 'slow'):
    """ prints all the perfect numbers in [n1,n2] (n1 < n2) in
        increasing fashion, using either a fast or a slow method
        (default method is 'slow')
    """
    if method == 'fast':
        fun = is_perfect_fast
    else:
        fun = is_perfect_slow

    for k in range(n1,n2+1):
        if fun(k) == True:
            print(k)
```

Dieses Vorgehen ist recht langsam und der Computer benötigt einige Zeit um alle perfekten Zahlen bis z.B. 100000 zu finden.
Schreibe eine Funktion `is_perfect_fast`, welche perfekte Zahlen schneller findet.
Tipp:
Zeige, dass es zur Entscheidung, ob eine natürliche Zahl $n \geq 2$ perfekt ist oder nicht, genügt die Teiler von n in der Menge $\{1, ..., \lfloor\sqrt{n}\rfloor\}$ zu Suchen.
Abgabe: bis spätestens Montag, 8. Oktober 2018, um 08:00.