Prof. Dr. Stefan Göller
Dr. Da-Jung Cho

Einführung in die Informatik

WS 2019/2020

Übungsblatt 10
17.01.2020 - 23.01.2020

Abgabe: Bis zum 23.01.20 18:00 Uhr über moodle.

Aufgabe 1 (Laufzeit von Programmen) (5+5+5+5=20 Punkte):

Geben Sie diese Aufgabe als `aufgabe_1.txt` ab. Kennzeichen Sie die Teilaufgaben bitte eindeutig.

Geben Sie jeweils die Best-Case-Laufzeit (also die am schwächsten wachsende Funktion, f, so dass das Programm in $O(f)$ läuft) sowie die Worst-Case-Laufzeit der folgenden Programme in Abhängigkeit von der Eingabe $n \in \mathbb{N}$, bzw. der Länge $n \in \mathbb{N}$ der Eingabe an. Geben Sie auch an, welche Funktion das Programm genau berechnet. Begründen Sie Ihre Angaben wie im folgenden Beispiel:

```
def programm0(n):
        if n % 2 == 0:
                return -1
        j = 0
        for i in range(n):
                j += i
        return j
```

Der Best-Case für dieses Programm sind Eingaben gerader Zahlen. In diesem Fall benötigt das Programm eine konstante Anzahl von Operationen, denn es werden nur die ersten beiden Zeilen je einmal ausgeführt. Der Best-Case ist also in $O(1)$. Der Worst-Case für das Programm sind ungerade Zahlen, die Laufzeit ist dann in $O(n)$. In diesem Fall werden die beiden ersten Zeilen einmal, und die Zeile in der Schleife n mal ausgeführt. Das Programm

berechnet die Funktion

$$f(n) = \begin{cases} -1 & \text{falls } n \text{ gerade} \\ \sum_{i=0}^{n-1} i = \frac{1}{2}(n-1)n & \text{sonst} \end{cases}$$

Gehen Sie davon aus, dass eine einzelne Ausführung jeder Zeile eine konstante Anzahl von Operationen benötigt.

a)
```
def programm1(n):  # Eingabegroesse: n
        if n < 2:
                return False
        divisor = 2
        while divisor * 2 <= n:
                if n % divisor == 0:
                        return False
                divisor += 1
        return True
```

b)
```
def programm2(n):  # Eingabegroesse: n
        i = 0
        while i <= n:
                i += 1
                if i ** 2 > 0:
                         break
        return i
```

c)
```
def programm3(L):  # Eingabegroesse: len(L)
      for i in range(len(L)):
            sortiert = True
            for j in range(1,len(L)-i):
                    if L[j] < L[j-1]:
                            L[j], L[j-1] = L[j-1], L[j]
                            sortiert = False
            if sortiert == True:
                  break
      return L
```

d)
```
def programm4(n):  # Eingabegroesse: n
        i = 1
        for j in range(n):
                i *= 2
        j = 1
        for k in range(i):
                j *= 2
        return j
```

Aufgabe 2 (Programmoptimierung) (5+5+10=20 Punkte):

Geben Sie für diese Aufgabe zwei Dateien ab, nämlich `aufgabe_2.txt` für Teilaufgaben a) und b) (kennzeichen Sie die Teilaufgaben bitte eindeutig) und eine Datei `aufgabe_2.py` für Teilaufgabe c).

Gegeben ist das folgende Programm, wobei `maxmult(L)` die Hauptfunktion ist:

```
def max(L):
        if len(L) == 0:
                return 0

        a  = max(L[1:])
        return a if a > L[0] else L[0]


def maxmult(L):
        if len(L) < 2:
                return 0

        a = maxmult(L[1:])
        b = max(L[1:]) * L[0]
        return a if a > b else b
```

Bei Eingabe einer Liste aus positiven Integern L gibt `maxmult`(L) die größte Zahl zurück, welche sich durch Multiplikation zweier Zahlen die an verschiedenen Positionen in L stehen, ergeben kann, falls die Liste mindestens Länge 2 hat. Ansonsten wird 0 zurückgegeben.

Beispiel: Bei Eingabe `[2,6,4]` gibt `maxmult` die Zahl `24` zurück und bei Eingabe `[2,6,8,1,8]` gibt `maxmult` die Zahl `64` zurück.

a) Beschreiben Sie kurz, wie das Programm diese Zahl rekursiv berechnet.

b) Geben Sie die Laufzeit des Programms mit Begründung an.

c) Schreiben Sie eine Funktion `maxmult_lin(L)`, welche die gleiche Funktion wie `maxmult(L)` berechnet, aber nur lineare Laufzeit benötigt.

Aufgabe 3 (Rekurrenzgleichungen) (8+8+4=20 Punkte):

Geben Sie diese Aufgabe als `aufgabe_3.txt` ab. Kennzeichen Sie die Teilaufgaben bitte eindeutig.

Für den Kurs "Einführung in die Informatik" soll der Meister im Kirschkernweitspucken ermittelt werden. Eine Partie läuft dabei folgendermaßen ab: Zwei Kontrahenden stellen sich nebeneinander und spucken einen Kirschkern möglichst weit. Wer weiter gespuckt hat, hat gewonnen. Wir nehmen an, dass so eine Partie, inklusive Aufstellung usw. genau eine Minute dauert, und niemals unentschieden endet.

Zur Ermittlung des Meisters sind nun verschiedene Formate möglich:

i) (*King of the Hill*) Gibt es nur einen Teilnehmer, hat dieser gewonnen. Ansonsten wird zu Beginn zufällig eine vorläufige Reihung der Spieler ausgelost. Der schlechteste tritt nun gegen den zweitschlechtesten an. Der Verlierer dieser Partie ist ausgeschieden, der Sieger tritt gegen den drittschlechtesten an, usw., bis der vorletzte noch teilnehmende Spieler gegen den Besten antritt. Der Gewinner dieser Partie ist der Meister.

ii) (*Single Elimination*) Gibt es nur einen Teilnehmer, hat dieser gewonnen. Ansonsten wird das Teilnehmerfeld zufällig in zwei gleichgroße Hälften aufgeteilt. Jede dieser Hälften ermittelt nach der Single-Elimination-Methode ihren besten Teilnehmer. Der Sieger der Partie zwischen diesen beiden besten ist der Meister.

iii) (*Round Robin*) Bei diesem Modus wird der Sieger nach Punkten ermittelt. Gibt es nur einen Teilnehmer, hat dieser mit 0 Punkten gewonnen. Ansonsten wird zufällig ein Teilnehmer A ausgelost. Die übrigen Teilnehmer spielen jetzt untereinander ihren Besten B aus, der dabei eine Punktzahl erhält. Dann spielt A gegen jeden der übrigen Teilnehmer, außer B. Für einen Sieg erhält A je einen Punkt, für einen Niederlage keinen. Dann spielt A gegen B. Der Sieger der Partie erhält einen zusätzlichen Punkt. Derjenige von A und B, der mehr Punkte hat, hat gewonnen. Bei Gleichstand entscheidet das Los.

iv) (*Kasseler Methode*) Gibt es nur einen Teilnehmer, hat dieser gewonnen. Bei zwei Teilnehmern entscheidet eine einzelne Partie über den Meister. Bei mehr als zwei Teilnehmern wird ein Teilnehmer A zufällig per Los bestimmt. Die übrigen Teilnehmer spielen nun nach der Kasseler Methode ihren Besten B aus. Danach spielen alle

Teilnehmer außer B, aber inklusive A, nach der Kasseler Methode ihren Besten C aus. Schließlich spielt B gegen C. Der Gewinner dieser Partie ist der Meister.

Wir gehen im Folgenden davon aus, dass die Anzahl der Teilnehmer eine Zweierpotenz ist. Ihre Aufgabe ist nun:

a) Stellen Sie für alle vorgeschlagenen Formate Rekurrenzgleichungen auf, die die Anzahl der benötigten Partien in Abhängigkeit von der Teilnehmergröße auf entsprechende Partiezahlen für kleinere Turniere zurückführen.

b) Geben Sie für jede Rekurrenzgleichung je eine geschlossene Form an. Begründen Sie kurz, warum dies die korrekte Form ist. Sie brauchen keinen Induktionsbeweis zu führen.

c) Berechnen Sie anschließend, wie lange es bei jedem Modus dauern würde, den Meister für die "Einführung in die Informatik" zu bestimmen. Gehen Sie von 32 Teilnehmern aus. Eine Partie dauert genau eine Minute. Da es nur eine Arena gibt, kann immer nur eine Partie gleichzeitig stattfinden. *Bonus*: Rechnen Sie die Turnierdauer ggf. in Stunden, Tage, Jahre um. Ein Jahr hat genau 365 Tage.

Hinweis: Für die Aufgabenteile a) und b) kann es für das Verständnis hilfreich sein, jeweils für einige kleine Eingaben das entsprechende Turnier einmal voll aufzubauen.