Aufgabe 1

Bearbeitungszeit: 20 Minuten

(a)

Eine Template-Klasse kann in C++ dazu benutzt werden, Algorithmen und Datenstrukturen so unabhängig wie möglich von einem Typ zu implementieren. Ein Beispiel dafür ist die Klasse `vector<T>`, die Daten des Typs `T` speichert, so z.B. `int`-Variablen in einem `vector<int>` oder Variablen vom Typ `string` in einem `vector<string>`.

(b)

Sei $F := \{f | f : \mathbb{N} \to \mathbb{R}_0^+\}$. Dann ist für $g \in F$:

$$\Theta(g) := \{f \in F | \exists c_1, c_2 > 0 \ \wedge \ \exists n_0 \in \mathbb{N} : \forall n \geq n_0 : c_1 g(n) \leq f(n) \leq c_2 g(n)\}$$

(c)

Doppelt verkettete Liste:

- *Vorne anfügen* geht bei Beibehaltung der Sortierung in $\mathcal{O}(n)$, da zuerst der richtige Einfügeort gesucht werden muss ($\mathcal{O}(n)$, wobei n =Länge der Liste). Das Einfügen geht in $\mathcal{O}(1)$.
  Muss man die Sortierung nicht beibehalten, so geht das Einfügen offensichtlich in $\mathcal{O}(1)$.

- *Suche* geht bei doppelt verketteten Listen nur in $\mathcal{O}(n)$, da man von vorne bis nach hinten durch die Liste gehen muss, bis das Element gefunden wurde. Die Ordnungseigenschaft kann dabei nicht ausgenutzt werden.

unsortiertes Array:

- *Vorne anfügen* braucht Zeit $\mathcal{O}(n)$, da das gesammte Array nach hinten verschoben werden muss, um vorne für das neue Element Platz zu schaffen.

- Wir haben keine Informationen über die gespeicherten Einträge, sodass man bei der *Suche* ggf. alle Elemente überprüfen muss, bis das gesuchte gefunden wurde. Somit haben wir $\mathcal{O}(n)$.

(d)

(i) *linear probing:* wenn eine Kollision auftritt wird solange ein neuer Hashwert mit einer Linearkombination des alten Hashwertes und einem Laufindex i bestimmt, bis man auf ein freies Feld stößt:

$$h(k,i) = (h_{alt}(k,i) + i) \mod m$$

(ii) *quadratic probing:* funktioniert wie *linear probing*, jedoch wird ein Polynom zweiten Grades verwendet:

$$h(k,i) = (h_{alt}(k) + c_1 i + c_2 i^2) \mod m$$

(iii) *double hashing:* hier wird mit Hilfe einer zweiten Hashfunktion solange ein neuer Hashwert berechnet, bis man auf ein freies Feld stößt:

$$h(k,i) = (h_{alt}(k) + i h_2(k)) \mod m$$

(e)

Eine endliche Menge $\mathcal{H}$ von Hashfunktionen $h : \mathcal{U} \to \{0, 1, ..., m-1\}$ heißt universelle Klasse von Hashfunktionen wenn für jedes Paar k_i, k_j von verschiedenen Schlüsseln gilt:
die Zahl der Funktionen $h \in \mathcal{H}$ mit $h(k_i) = h(k_j)$ ist höchstens $\frac{|\mathcal{H}|}{m}$.

(f)

Ein Circuit (= Rundgang) in einem Graphen ist ein Pfad, bei dem der erste und der letzte Knoten gleich sind. Ein Euler-Circuit ist ein Rundgang, bei dem jede Kante des Graphen genau einmal enthalten ist.

Aufgabe 2

Bearbeitungszeit: 30 Minuten

(a)

```
template<class T>
class liste {
  public:
    liste();
    liste(const liste&);
    ~liste();
    liste& operator=(const liste&);

    int elementanzahl() const {return anzahl;}

    void links_einfuegen(const T&);

    void suche(const T& i);

    void loesche(const T& i);

  private:
    class listenelement {
      friend class liste<T>;
      T knoten;
      listenelement *nachfolger, *vorgaenger;
      listenelement(const T& dat)
        : knoten(dat), nachfolger(NULL), vorgaenger(NULL) {}
    };

    listenelement *ende, *anfang;
    int anzahl;
}
```

(b)

```
template<class T>
void liste<T>::loesche(const T& i) {
  listenelement *current = anfang;
  while(current) {
    if (i == current->knoten) {
      if (current->vorgaenger != NULL) {
        current->vorgaenger->nachfolger = current->nachfolger;
      }
      else {
        anfang = current->nachfolger;
      }
      if (current->nachfolger != NULL) {
        current->nachfolger->vorgaenger = current->vorgaenger;
      }
      else {
        ende = current->vorgaenger;
      }
      delete current;
      anzahl--;
      }
    current = current->nachfolger;
  }
}
```

Aufgabe 4

Bearbeitungszeit: 15 Minuten
Anordnung: d,c,b,a
Begründung:
$(*)$: L'Hôpital

$$\begin{aligned}\lim_{n\to\infty}\frac{|n\cdot\ln(n)|}{|4^{\log_2(n+1)}|} &= \lim_{n\to\infty}\frac{|n\cdot\ln(n)|}{|(n+1)^2|}\\ &\overset{(*)}{=} \lim_{n\to\infty}\frac{|\ln(n)+1|}{|2n+2|}\\ &\overset{(*)}{=} \lim_{n\to\infty}\frac{|\frac{1}{n}|}{|2|}\\ &= \lim_{n\to\infty}\frac{1}{|2n|}=0\end{aligned}$$

$$\begin{aligned}\lim_{n\to\infty}\frac{|(n+1)^2|}{|e^n|} &\overset{(*)}{=} \lim_{n\to\infty}\frac{|2n+2|}{|e^n|}\\ &\overset{(*)}{=} \lim_{n\to\infty}\frac{2}{|e^n|}=0\end{aligned}$$

$$\begin{aligned}\lim_{n\to\infty}\frac{|e^n|}{|n!|} &= \lim_{n\to\infty}\frac{|\prod_{i=0}^{n-1}e|}{|\prod_{i=0}^{n-1}(n-i)|}\\ &= \lim_{n\to\infty}\prod_{i=0}^{n-1}\frac{e}{n-i}=0\end{aligned}$$

Aufgabe 5

Bearbeitungszeit: 15 Minuten
Nach 4 weiteren Runden ist das Spiel auf jeden Fall entschieden. Da die Wahrscheinlichkeit, dass ein Spieler eine Runde gewinnt, $\frac{1}{2}$ ist, ergeben sich folgende Möglichkeiten für die Sieger dieser 4 Runden, wobei jede Kombination gleich wahrscheinlich ist:

A-A		
A-B-A	B-A-A	
A-B-B-A	B-A-B-A	B-B-A-A
B-B-B		
B-B-A-B	B-A-B-B	A-B-B-B

$$
\begin{aligned}
P(A \text{ gewinnt zuerst 10 Spiele}) &:= P(A) \\
P(A) &= \left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^3 + \left(\frac{1}{2}\right)^4 + \left(\frac{1}{2}\right)^3 + \left(\frac{1}{2}\right)^4 + \left(\frac{1}{2}\right)^4 \\
&= \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \frac{1}{8} + \frac{1}{16} + \frac{1}{16} \\
&= \frac{4+2+1+2+1+1}{16} \\
&= \frac{11}{16}
\end{aligned}
$$

$$
\begin{aligned}
P(B \text{ gewinnt zuerst 10 Spiele}) &:= P(B) \\
P(B) &= 1 - P(A) \\
&= 1 - \frac{11}{16} \\
&= \frac{5}{16}
\end{aligned}
$$

Spieler A gewinnt in 11 von 16 Fällen, Spieler B gewinnt in 5 von 16 Fällen. Der Gewinn-Betrag geht also zu $\frac{11}{16}$ an Spieler A und zu $\frac{5}{16}$ an Spieler B.

Aufgabe 6

Bearbeitungszeit: 10 Minuten

$$T(n) = 2T\left(\frac{n}{3}\right) + n^3$$

Vermutung: $T(n) \in \mathcal{O}(n^3)$
Behauptung: $T(n) \leq c \cdot n^3$, wobei $c > 1$ beliebig
Beweis:
$n = 1$: $T(1) = 1 \leq c$
Induktionsschritt:

$$\begin{aligned} T(n) &= 2T\left(\frac{n}{3}\right) + n^3 \\ &\leq 2c\left(\frac{n}{3}\right)^3 + n^3 \\ &= 2 \cdot \frac{c}{27}n^3 + n^3 \\ &= 3n^3 \quad \text{für } c = 27 \end{aligned}$$

Aufgabe 7

Bearbeitungszeit: 15 Minuten

- $T(n) = 64T\left(\frac{n}{8}\right) + 1042n^2 + 1000000n$

$$\begin{aligned} a &= 64 \\ b &= 8 \\ f(n) &= 1042n^2 + 1000000n \\ \log_8(64) &= 2 \\ f(n) &\in \Theta(n^{\log_b(a)}) = \Theta(n^2) \end{aligned}$$

Dann gilt nach dem 2. Fall des Mastertheorems

$$T(n) = \Theta(n^2 \log(n))$$

- $T(n) = 27T\left(\frac{n}{3}\right) + n^3 \ln(n) + 42$

$$\begin{aligned} a &= 27 \\ b &= 3 \\ f(n) &= n^3 \ln(n) + 42 \\ \log_3(27) &= 3 \Rightarrow n^{\log_3(27)} = n^3 \end{aligned}$$

Es gilt:

$$f(n) \in \Omega(n^3)$$

Wir können jedoch kein $\varepsilon > 0$ finden, sodass $f(n) \in \Omega(n^{3+\varepsilon})$ ist, da der Logarithmus langsamer wächst als jede Potenz. Somit ist das Mastertheorem hier nicht anwendbar.