### Self-Check Questions

1.1

1. Eine durch die Nutzer:innen vorgegebene Folge reellwertiger Zahlen soll in aufsteigender Reihenfolge sortiert werden. Erstellen Sie eine vollständige Definition des entsprechenden Rechenproblems.

|  |
| --- |
| *Antwort:*  Eingabe (Input): Folge V aus n reellwertigen Zahlen, bezeichnet mit  Vorbedingungen: n ist eine positive natürliche Zahl und jeder Wert bzw. jedes Element von V ist eine reelle Zahl.  Ausgabe (Output): Folge W aus n reellwertigen Zahlen, bezeichnet mit  Nachbedingung:  Zusammenhänge zwischen Eingaben und Ausgaben: Für jeden in V enthaltenen Wert gilt, dass die Zahl seiner Instanzen in V mit der Zahl seiner Instanzen in W übereinstimmt. Umgekehrt gilt für jeden in W enthaltenen Wert, dass die Zahl seiner Instanzen in W mit der Zahl seiner Instanzen in V identisch ist. |

1.2

1. Erstellen Sie ein JavaScript-Programm mit einer JavaScript-Funktion, die einen Primzahltest für jede eingegebene natürliche Zahl größer eins durchführt. Testen Sie Ihr Programm nacheinander mit jeder der folgenden Zahlen: -4, 0, 1, 2, 3, 3.5, 5, 6, 6.2, 15, 17, 18, 21 und 23.

|  |
| --- |
| *Antwort:*  Einfacher Algorithmus für Primzahltests  **let** readline **=** require**(**'readline-sync'**);**  **let** isPrime **=** **function** **(**n1**)** **{**  **if** **(**n1**>**1**){**  **let** r **=** **true;** **let** i **=** 2**;**  **while** **((**r**===true)** **&&** **(**i**<**n1**)){**  r **=** **((**n1 **%** i**)** **!==** 0**);** i**++;**  **}**  **return** r**;**  **}**  **else** **if** **(**n1**===**1**)** **{return(false)}**  **}**  **let** strictlyPosInt **=** **function(**s**){**  **let** n **=** Number**(**s**);**  **return((**s**!==**''**)** **&&** **(**Number**.**isInteger**(**n**))** **&&** **(**n**>**0**))**  **}**  **while** **(true){**  **let** mess1 **=** '\nEnter a strictly positive integer '**;**  **let** mess2 **=** 'Only strictly positive integers please!'**;**  **let** ns **=** readline**.**question**(**mess1**);**  **if** **(**strictlyPosInt**(**ns**)===true){**  **let** n **=** **parseInt(**ns**);**  console**.**log**(**'\n'**+**n**+**' is a prime number: '**+**isPrime**(**n**));**  **}**  **else** **{**console**.**log**(**mess2**);}**  **}** |

1.3

1. Erstellen Sie ein JavaScript-Programm, das die Wurzel einer positiven natürlichen Zahl mit dem gewünschten Genauigkeitsgrad berechnet. Letzterer soll durch die Nutzer:innen über die Zahl der erwarteten Nachkommastellen festgelegt werden können. Außerdem soll das Programm das Quadrat des zuvor ausgegebenen Ergebnisses bzw. das Quadrat der approximierten Wurzel (mit der jeweils festgelegten Genauigkeit) berechnen. Erstellen Sie anschließend eine Übersicht der Ausgabe Ihres Programms bei der Berechnung der Wurzel von 3 mit 0, 1, 2, 3, 4, 5, 6, 7 und 8 Nachkommastellen.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Antwort:*  Algorithmus zur Wurzelberechnung  **let** readline **=** require**(**'readline-sync'**);**  **let** squRt **=** **function** **(**n1**,**d1**){**  **if** **((**n1**>=**0**)** **&&** **(**d1**>=**0**))** **{**  **return** **(**Math**.**sqrt**(**n1**)).**toFixed**(**d1**);**  **}**  **}**  **let** posInt **=** **function(**s**){**  **let** n **=** Number**(**s**);**  **return((**s**!==**''**)** **&&** **(**Number**.**isInteger**(**n**))** **&&** **(**n**>=**0**))**  **}**  **while** **(true){**  **let** mess1 **=** '\nEnter a positive integer '**;**  **let** mess2 **=** 'Required number of digits after dot '**;**  **let** mess3 **=** 'Only positive integers please!'**;**  **let** nt **=** readline**.**question**(**mess1**);**  **let** mt **=** readline**.**question**(**mess2**);**  **if** **((**posInt**(**nt**))** **&&** **(**posInt**(**mt**))){**  **let** n **=** **parseInt(**nt**);**  **let** d **=** **parseInt(**mt**);**  **let** r **=** squRt**(**n**,**d**);**  **let** ms **=** 'Square root of ' **+** n**;**  ms **=** ms **+** ' with ' **+** d **+** ' digits after dot: ' **+** r**;**  console**.**log**(**ms **+** ' whose square is ' **+** **(**r**\***r**)** **);**  **}**  **else{**  console**.**log**(**mess3**);**  **}**  **}**  Ausgabe der errechneten Wurzel von 3 und des dazugehörigen Quadrats mit den gewünschten Genauigkeitsgraden (Nachkommastellen):   |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | Zahl d. Nach­kommastellen | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | | Wurzel | 2 | 1,7 | 1,73 | 1,732 | 1,7321 | 1,73205 | 1,732051 | 1,7320508 | | Quadrat | 4 | 2,88999 | 2,9929 | 2,999824 | 3,00017 | 2,9999972 | 3,0000006 | 2,99999997 | |

1.4

1. Welchem zentralen ethischen Prinzip sollte die Datenverarbeitung durch Algorithmen unterworfen sein?

*Algorithmen sollten Daten nur dann verarbeiten, wenn dies dem Wohl der Gesellschaft dient.*

1. Erstellen Sie ein Programm zur Implementierung des oben beschriebenen Algorithmus für die Lauflängenkodierung und testen Sie es mit der Binärzahl 11111111111111111000011111111111. Dabei können Sie die folgende Funktion zur Konversion einer Dezimalzahl in eine Binärzahl verwenden.

***function*** binarise**(**n1**)** **{**

***return*** **(**n1 **>>>** 0**).**toString**(**2**);**

**}**

|  |
| --- |
| *Antwort:*  Algorithmus zur Lauflängenkodierung (Start)  **let** readline **=** require**(**'readline-sync'**);**  **let** cutBlocks **=** **function** **(**s**){**  **let** bl **=** **new** Array**();**  **let** be **=** 0**;**  **let** ct **=** 1**;**  **for** **(let** i**=**1**;** i**<**s**.**length**;** i**++){**  **if** **(**s**[**i**]** **===** s**[**be**]){**  ct**++;**  **}**  **else{**  bl**.**push**(**s**[**be**]);**  bl**.**push**(**ct**);**  be **=** i**;**  ct **=** 1**;**  **}**  **}**  bl**.**push**(**s**[**be**]);**  bl**.**push**(**ct**);**  **return(**bl**);**  **}**  **let** codeLength **=** **function** **(**bls**){**  **let** r **=** 0**;**  **let** i **=** 1**;**  **while(**i**<**bls**.**length**){**  **if** **(**bls**[**i**]** **>** r**){**  r **=** bls**[**i**];**  **}**  i**=**i**+**2**;**  **}**  **return((**binarise**(**r**)).**length**);**  **}**  **let** completeZeros **=** **function** **(**s1**,**n1**){**  **let** r **=** ""**;**  **for** **(let** i**=**0**;** i**<**n1**-(**s1**.**length**);** i**++){**  r **=** '0' **+** r**;**  **}**  **return** **(**r**+**s1**);**  **}**  **let** binarise **=** **function** **(**n1**){**  **return** **(**n1 **>>>** 0**).**toString**(**2**);**  **}** |
| *Antwort (Fortsetzung):*  Algorithmus zur Lauflängenkodierung (Ende)  **let** codeBlocks **=** **function** **(**bls**){**  **let** l **=** codeLength**(**bls**);**  **let** r **=** ""**;**  **let** i **=** 0**;**  **while(**i**<**bls**.**length**){**  **let** s **=** binarise**(**bls**[**i**+**1**]);**  **let** z **=** completeZeros**(**s**,**l**);**  r **=** r **+** bls**[**i**]** **+** z**;**  i**=**i**+**2**;**  **}**  **return(**r**);**  **}**  **let** isBinary **=** **function** **(**s**){**  **let** i **=** 0**;**  **while** **((**i**<**s**.**length**)&&((**s**.**charAt**(**i**)===**'0'**)||(**s**.**charAt**(**i**)===**'1'**)))**  **{**i**++;}**  **return** **(**i**===**s**.**length**);**  **}**  **while** **(true){**  **let** mess **=** '\nEnter a binary number '**;**  **let** b **=** readline**.**question**(**mess**);**  **if** **((**b**.**length**!==**0**)** **&&** **(**isBinary**(**b**)===true)){**  **let** blcs **=** **new** Array**();**  blcs **=** cutBlocks**(**b**);**  **let** cl **=** codeLength**(**blcs**);**  **let** rle **=** codeBlocks**(**blcs**);**  console**.**log**(**'The initial binary is: ' **+** b**);**  console**.**log**(**'The blocks of are: ' **+** blcs**);**  console**.**log**(**'Length of the binary of longest block is: '**+**cl**);**  console**.**log**(**'The RLE code is: ' **+** rle**);**  **}**  **}** |

2.1

1. Erstellen Sie eine vollständige JavaScript-Klasse, die einen Array-basierten Stapel implementiert. Testen Sie die Klasse im Hauptprogramm, indem Sie die Zahlen 10, 35, 63, 12 und 29 nacheinander mit jeweils einem Push-Befehl in einen Stapel einfügen. Anschließend soll Ihr Hauptprogramm den Stapel anzeigen, das oberste Element per Pop-Befehl entfernen und den verbleibenden Stapel erneut anzeigen.

|  |
| --- |
| *Antwort:*  Stack-Implementierung mit einem Array  **let** readline **=** require**(**'readline-sync'**);**  **class** ArrStack **{**  **constructor()** **{this.**myArray **=** **new** Array**();}**  putMeOnTopOf**(**t**)** **{(this.**myArray**).**push**(**t**);}**  removeMyTop**()** **{this.**myArray**.**pop**();}**  displayMe**(){**  **let** txt **=** 'Displaying the stack'**;**  txt **=** txt **+** '\n' **+** 'Top of the stack'**;**  **let** tp **=** **((this.**myArray**).**length**)-**1**;**  **while** **(**tp **>=** 0**){**  txt **=** txt **+** '\n' **+** **this.**myArray**[**tp**];** tp**--;**  **}**  **return(** txt **+** '\n' **+** 'Bottom of the stack'**);**  **}**  getTopElement**()** **{**  **let** tp **=** **((this.**myArray**).**length**)-**1**;**  **return** **this.**myArray**[**tp**];**  **}**  **}**  **let** captureElements **=** **function** **(**s1**,**n1**)** **{**  **let** mess **=** 'Please enter your next element '**;**  **for** **(let** i **=** 0**;** i **<** n1**;** i**++)** **{**  **let** e **=** readline**.**question**(**mess**);**  s1**.**putMeOnTopOf**(**e**);**  **}**  **}**  **while** **(true){**  **let** mess **=** '\nPlease how many elements do you have? '**;**  **let** n **=** **parseInt(**readline**.**question**(**mess**));**  **let** s **=** **new** ArrStack**();**  captureElements**(**s**,**n**);** console**.**log**(**s**.**displayMe**());**  s**.**removeMyTop**();**  console**.**log**(**'One element removed from the stack'**);**  console**.**log**(**s**.**displayMe**());**  **}** |

2.2

1. Erstellen Sie eine rekursive Version der nachstehenden iterativen JavaScript-Funktion, die jedes Zeichen eines gegebenen Strings in Großbuchstaben umwandelt. Erweitern Sie dann Ihr Hauptprogramm um Ausgabeanweisungen, die zweifelsfrei belegen, dass beide Varianten zum selben Ergebnis führen. Stellen Sie den Output Ihres Programms für den Input-String „Advanced Algorithmics“ dar.

Iterativer Algorithmus zur Umwandlung in Großbuchstaben

|  |
| --- |
| **l****et** readline **=** require**(**'readline-sync'**);**  **let** capIter **=** **function** **(**s1**)** **{**  **if** **(**s1**.**length**>**0**)** **{**  **let** r **=** ""**;**  **for** **(let** i**=**0**;** i**<**s1**.**length**;** i**++){**  r **=** r **+** s1**[**i**].**toUpperCase**();**  **}**  **return** r**;**  **}**  **}**  **while** **(true){**  **let** s **=** readline**.**question**(**'\nEnter a string '**);**  **let** ci **=** capIter**(**s**);**  **let** ms **=** 'Capitalization of '**+**s**+**' iterative is '**+**ci**;**  console**.**log**(**ms**);**  **}** |

|  |
| --- |
| *Antwort:*  Iterative und rekursive Variante des Algorithmus zur Umwandlung in Großbuchstaben (Start)  **let** readline **=** require**(**'readline-sync'**);**  **let** cpI **=** **function** **(**s1**)** **{**  **if** **(**s1**.**length**>**0**)** **{let** r **=** ""**;**  **for** **(let** i**=**0**;** i**<**s1**.**length**;** i**++){**r **=** r**+**s1**[**i**].**toUpperCase**();}**  **return** r**;**  **}**  **}** |

|  |
| --- |
| *Antwort:*  Iterative und rekursive Variante des Algorithmus zur Umwandlung in Großbuchstaben (Ende)  **let** cpR **=** **function** **(**s1**){**  **if** **(**s1**.**length**>**0**){**  **if** **(**s1**.**length**===**1**)** **{return** s1**[**0**].**toUpperCase**();}**  **else** **{**  **return((**s1**[**0**].**toUpperCase**())+**cpR**(**s1**.**substring**(**1**,**s1**.**length**)));**  **}**  **}**  **}**  **while** **(true){**  **let** s **=** readline**.**question**(**'\nEnter a string '**);**  **let** i **=** cpI**(**s**);** **let** r **=** cpR**(**s**);**  **let** o **=** 'Capitalizing '**+**s**+**'. Iterative: '**+**i**+**'; Recursive: '**+**r**;**  console**.**log**(**o**);**  **}**  *Output des Programms für den Input-String „Advanced Algorithmics“:*  Capitalizing Advanced Algorithmics. Iterativ: ADVANCED ALGORITHMICS; Rekursiv: ADVANCED ALGORITHMICS |

2.3

1. Wahr oder falsch: Die Aufteilung der Lösung eines Problems in verschiedene Funktionen und Abläufe ist ein Beispiel für die Nutzung des Teile-und-herrsche-Ansatzes durch einen Algorithmus.

* Wahr
* *Falsch*

2.4

1. Fügen Sie den Wert 100 in den unten abgebildeten binären Suchbaum ein und stellen sie den daraus resultierenden balancierten Baum dar.

Zu erweiternder AVL-Baum

|  |
| --- |
| *Antwort:*  Erweiterung des AVL-Baums um den Wert 100 |

3.1

1. Identifizieren Sie die am besten geeignete Option (A, B, C, D oder E) für jeder der Zellen der folgenden Tabelle in Bezug auf die folgende Aussage: Um eine Sequenz SE in aufsteigender Wertefolge zu sortieren, sucht und tauscht der Sortieralgorithmus SA hauptsächlich und kontinuierlich die Position des ... Elements in verschiedenen Unterfolgen von SE.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Option | | | | |
|  | A  Kleinsten | B  Mittleren | C  Größten | D  A, B und C | E  Weder A, noch B, noch C, noch D |
| SA |
| Bubblesort |  |  |  |  |  |
| Insertionsort |  |  |  |  |  |
| Bucketsort |  |  |  |  |  |
| Quicksort |  |  |  |  |  |
| Mergesort |  |  |  |  |  |

*Antwort*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Option | | | | |
|  | A  Kleinsten | B  Mittleren | C  Größten | D  A, B und C | E  Weder A, noch B, noch C, noch D |
| SA |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| Bubblesort |  |  | C |  |  |
| Insertionsort | A |  |  |  |  |
| Bucketsort |  |  |  |  | E |
| Quicksort |  | E |  |  |  |
| Mergesort | A |  |  |  |  |

3.2

1. Sehen Sie sich das nachfolgend abgebildete Json-Objekt genau an. Formulieren Sie dann zwei jq-test-Befehle, die die korrekte Formatierung ähnlicher Json-Objekte überprüfen. Gehen Sie dabei davon aus, dass die Liste der erlaubten Anreden die Werte Ms, Mrs, Mr, Dr und Prof umfasst und Initialen aus einzelnen Großbuchstaben gefolgt von einem Punktzeichen bestehen. Die Zahl der Initialen soll zwischen 1 und 8 liegen. Außerdem ist davon auszugehen, dass die im Array „Expenses“ vorfindlichen Elemente Geldbeträge bezeichnen, die ein Punktzeichen mit beliebig vielen vorausgehenden und nachfolgenden Ziffern enthalten.

Drittes Json-Beispielobjekt

{

"Nickname": "Ms S.D.E.O.",

"Expenses": ["300.07", "278.45", "67.34"]

}

Wie stellt sich der Output ihrer beiden jq-Befehle für das oben aufgeführte und das nachfolgende Objekt dar?

Viertes Json-Beispielobjekt

{

"Nickname": "Mr ALGO.O.E.",

"Expenses": ["csvsj278.45", "300.07", "67.hhs34"]

}

*Antwort:*

test-Befehl zur Prüfung der Nickname-Einträge:

.Nickname | test("^(Ms|Mrs|Mr|Dr|Prof)\\s+(([A-Z]\\.){1,8})$")

Output für das Objekt „Ms S.D.E.O“: true

Output für das Objekt „Mr ALGO.O.E.“ false

test-Befehl zur Prüfung der Elemente im Array Expenses:

.Expenses[] | test("^([0-9]+)\\.([0-9]+)$")

Output für das Objekt „Prof S.D.E.O“: true, true, true

Output für das Objekt „Ms ALGO.O.E.“ false, true, false

3.3

1. Berechnen Sie die Werte von e und d für den (oben abgebildeten) RSA-Algorithmus. Gehen Sie dabei von den folgenden Annahmen aus: p1 =5; p2 =13; e ist größer als p1 und das kleinste gemeinsame natürliche Vielfache von p1, das unter den gegebenen Umständen möglich ist; d ist die kleinste natürliche Zahl, die größer als e und unter den gegebenen Umständen passend ist.

*Antwort: e=25 and d=73*

3.4

1. Nutzen Sie den k-means-Algorithmus, um den folgende Datensatz in drei Cluster zu unterteilen: 3, 5, 35, 25, 7, 43 und 15. Wählen Sie dabei 7, 35 und 43 als Ursprungswerte. Ihr Endergebnis muss sowohl eine Liste aller Elemente jedes Clusters als auch deren jeweilige Mittelwerte umfassen. Außerdem ist eine grafische Darstellung der ermittelten Cluster erforderlich.

*Antwort:*

*Es ergeben sich die Cluster {3, 5, 7, 15}, {25, 35} und {43} mit den Mittelwerten 7,5; 30; 43. Die grafische Darstellung der ermittelten Cluster sieht dann folgendermaßen aus:*

Ergebnis des k-means-Algorithmus

4.1

1. Die nachfolgend dargestellte Funktion isPrime soll ermitteln, ob es sich bei dem übergebenen Parameter m um eine Primzahl handelt. Erstellen Sie einen Beweis für die partielle Korrektheit dieser Funktion.

Primzahltestfunktion

**let** isPrime **=** **function** **(**m**){**

**let** d **=** **true;**

**if** **(**m**===**2**){**d **=** **false;}**

**if** **(**m**>=**3**){**

**let** i **=** 2**;** d **=** **((**m **%** i**)** **===** 0**);**

**while** **((**d**===false)** **&&** **(**i**<**m**-**1**)){**

i**++;** d **=** **((**m **%** i**)** **===** 0**);**

**}**

**}**

**return** **!**d**;**

**}**

*Antwort:*

*Wir gehen davon aus, dass die Vorbedingung der Primzahltest-Funktion* isPrime *die zulässigen Eingabewerte auf positive ganze Zahlen begrenzt. Die Schleifeninvariante fordert: Für jeden Wert* i<m-1 *ist der Eingabewert* m *der Primzahltest-Funktion durch keine Ganzzahl* j *zwischen 2 und* i-1 *teilbar. Die Nachbedingung besagt, dass die Primzahltest-Funktion genau dann den Booleschen Wert „true“ ausgibt, wenn es sich bei dem eingegebenen Parameter um eine Primzahl handelt.*

*Da der Algorithmus offensichtlich korrekt ist, wenn m den Wert 0, 1 oder 2 hat, interessieren im Folgenden nur Eingaben von* m>=3*: In diesem Fall gilt vor dem Eintritt in die Schleife* i=2*, sodass es keine ganze Zahl im Intervall zwischen 2 und 1 (d. h.* i-1*) gibt. Folglich ist jeder Eingabewert* m *nicht durch ein Element aus dem Intervall zwischen 2 und 1 teilbar. Anschließend nehmen wir an, dass die Schleifeninvariante für jeden Wert i kleiner oder gleich einem gegebenen Wert k zwischen 2 und m-2 wahr ist. Unter dieser Voraussetzung müssen wir nun zeigen, dass die Invariante auch für* i=k+1 *wahr ist. Dies gelingt unter Verweis auf die folgenden Feststellungen: Die Schleife kann die Anweisung* i=k+1 *nur erreichen, weil k kein Teiler von m ist. Außerdem können wir aus der Induktionshypothese folgern, dass eine Teilung von m durch einen Wert kleiner (oder gleich) k ebenfalls ausgeschlossen ist. Das beweist: Wenn* i=k+1 *ist, ist m durch keine ganze Zahl von 2 bis k teilbar. Was nun den Austritt aus der Schleife angeht, so ist hier die folgenden Bedingung relevant:* ((d===false) && (i<m-1))*. Die Schleife erreicht also ihr Ende, wenn* (d===true) *oder* (i===m-1)*. Im ersten Fall* (d===true) *erfolgt dies, weil* m *keine Primzahl ist. Im zweiten Fall* (i===m-1) *stoppt die Schleife,* weil *keine der infrage kommenden Zahlen einschließlich* m-1 *ein Teiler von* m *ist, d. h. die Bedingung*(d===true) *erfüllt.*

4.2

1. Worin besteht der Zweck der folgenden sgP-Funktion, die auf der Primzahltest-Funktion isPrime basiert?

Beispielprogramm

**let** sgP **=** **function** **(**m**)** **{**

**if** **(**m**>=**0**)** **{**

**let** p **=** m**;**

**while** **((**isPrime**(**p**)===false)** **||** **(**isPrime**((**2**\***p**)+**1**)===false)){**

p**++;**

**}**

**return** p**;**

**}**

**}**

*Antwort:*

Die sgP-Funktion soll für einen gegebenen Wert m die kleinste Zahl p ermitteln, die die folgenden drei Bedingungen erfüllt: p ist größer oder gleich m; p ist eine Primzahl; 2p+1 ist ebenfalls eine Primzahl.

4.3

1. Modifizieren Sie das letzte Programm dieses Abschnitts so, dass es die oben beschriebene SQL-Injektion verhindert.

*Antwort: Der letzte Teil des Programms sollte wie folgt modifiziert werden (während die anderen Programmteile in unveränderter Form übernommen werden können):*

Code zur Verhinderung eines SQL-Injection-Angriffs

con**.**connect**(function(**err**)** **{**

emad**.**replace**(/'/g,**'\_'**);** **if** **(**err**)** **throw** err**;**

**const** sql1 **=** "SELECT \* FROM login WHERE email=" **+** "'" **+** emad **+** "'"**;**

con**.**query**(**sql1**,** **function** **(**err**,** result**)** **{**

**if** **(**err**)** **throw** err**;** console**.**log**(**result**);**

**});**

**});**

4.4

1. Definieren Sie das Mittelintervall für das im entsprechenden Abschnitt beschriebene Mittelgliederproblem mit n=100 und epsilon=1/64. Berechnen Sie die geschätzte Erfolgsquote von midRange0 und midRangeV für c=80.

*Antwort: Mittelintervall [493, 507]; geschätzte Genauigkeit 1,56 % and 71,63 %*

5.1

1. Identifizieren Sie die einzige Turingmaschine, die in diesem Abschnitt abgebildet ist, und ermitteln Sie das Ergebnis ihrer Verarbeitung des Eingabewerts abcabc.

*Antwort:*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Zustand | Band | | | | | | |
| SaZ | ↓a | B | c | a | b | c | ꓕ |
| Sb | N | ↓b | c | a | b | c | ꓕ |
| Sc | N | Z | ↓c | a | b | c | ꓕ |
| SN | N | ↓Z | P | a | b | c | ꓕ |
| SN | ↓N | Z | P | a | b | c | ꓕ |
| SaZ | N | ↓Z | P | a | b | c | ꓕ |
| SZ | N | Z | ↓P | a | b | c | ꓕ |
| SP | N | Z | P | ↓a | b | c | ꓕ |

Es gibt keinen möglichen Folgezustand für Sp, wenn der Lese-Schreib-Kopf auf das Zeichen a zeigt. Die Turingmaschine erkennt also das Wort abca nicht an, weil dieses nicht Teil seiner Sprache anbncn (mit n als positiver natürlicher Zahl) ist.

5.2

1. Bitte vervollständigen Sie den folgenden Satz:

Aufgrund der Unentscheidbarkeit des Halteproblems gibt es keinen Algorithmus, der im Vorhinein entscheiden kann, ob eine zufällig gewählte Turing...

*Antwort: ...maschine für einen zufällig gewählten Input terminiert.*

5.3

1. Suchen Sie im Internet nach einer Beschreibung von Hilberts zehntem Problem und treffen Sie ein Urteil über seine Entscheidbarkeit. Vergessen Sie dabei nicht, ihre Quelle(n) zu zitieren.

*Antwort:* *Bei Hilberts zehntem Problem geht es um die Frage, ob es einen Algorithmus gibt, der für eine willkürliche gewählte polynomische Gleichung mit ganzzahligen Koeffizienten die Existenz einer ganzzahligen Lösung ermitteln kann. Dies ist ein bekanntermaßen unentscheidbares Problem (Richardson, 2020).*

6.1

1. Erstellen Sie ein JavaScript-Programm in NodeJs, das ohne Berücksichtigung der Groß- und Kleinschreibung überprüft, ob es sich bei einem Inputstring um ein Palindrom handelt. Geben Sie außerdem die Approximationsfunktion seiner Zeitkomplexität in O-Notation an. Hinweis: Ein Palindrom ist eine Zeichenkette, die von links nach rechts gelesen die gleiche Reihenfolge aufweist wie von rechts nach links (beispielsweise das Wort „Ebbe“).

*Antwort: Es handelt sich um einen O(n)-Algorithmus.*

Einfacher JavaScript-Algorithmus zur Identifizierung eines Palindroms

**let** readlineSync **=** require**(**'readline-sync'**);**

**let** palin **=** **function** **(**ss**){**

**let** s**=**ss**.**toUpperCase**();** **let** n**=**s**.**length**;** **let** pl**=true;** **let** i**=**0**;**

**while** **((**i**<**n**/**2**)** **&&** **(**pl **===** **true))** **{**pl **=** **(**s**[**i**]===**s**[(**n**-**1**)-**i**]);** i**++;}**

**return(**pl**);**

**}**

**while(true){**

**let** t **=** readlineSync**.**question**(**'\nPlease type something : '**);**

console**.**log**(**'It is ' **+** palin**(**t**)+**' that ' **+** t **+** ' is a palindrome'**);**

**}**

1. Geben Sie unter Bezugnahme auf relevante Belege an, ob die Aussage NLOGTIME ⊆ NL wahr oder falsch ist.

*Antwort: Aussage* NLOGTIME ⊆ NL *ist wahr. Dies lässt sich unter Bezugnahme auf die unten aufgeführten Eigenschaften belegen:*

*Da* NTIME(f(n)) ⊆ DSPACE(f(n)) *gilt* NTIME(log2(n))) ⊆ DSPACE(log2(n)))*. Mit anderen Worten:* NLOGTIME ⊆ L*, weil* NTIME(log2(n))) = NLOGTIME *und* L = DSPACE(log2(n)))*.*

An dieser Stelle vernachlässigen wir für einen Moment die Tatsache, dass NLOGTIME ⊆ L*.*

*Es gilt* DSPACE(f(n)) ⊆ NSPACE(f(n))*, sodass,* DSPACE(log2(n))) ⊆ NSPACE(log2(n)))*. Mit anderen Worten:* L ⊆ NL*, da* L = DSPACE(log2(n))) *und* NL = NSPACE(log2(n)))*.*

*Damit haben wir* L ⊆ NL *belegt. Wenn wir nun wieder berücksichtigen, dass* NLOGTIME ⊆ L *bereit bewiesen ist, können wir folgern, dass* NLOGTIME ⊆ L ⊆ NL*. Dies zeigt, dass NLOGTIME* ⊆ NL *wahr ist.*

6.2

1. Erläutern Sie die Optimierungsversion des oben erwähnten Färbungsproblems aus der Graphentheorie.

*Antwort:* *Fragestellung der Optimierungsversion des Färbungsproblems der Graphentheorie: Ermittelt werden soll die kleinste Zahl verschiedener Farben, mit denen alle gegebenen Knoten eines Graphen so belegt werden können, dass keine zwei direkt verbundenen Knoten die gleiche Farbe haben.*

6.3

1. Bitte vervollständigen Sie den folgenden Satz:

Ein Problem gilt genau dann als praktisch unlösbar, wenn es nicht *mit einer physikalisch realisierbaren Rechenmaschine* in *polynomieller* Zeit gelöst werden kann.

7.1

1. Wie viele parallel laufende Prozesse werden in ParallelJs durch die map-Operation für die auf dem obigen Screenshot abgebildeten Matrizen erstellt?

*Antwort: Es werden zwei (2) Prozesse erstellt – einer pro Spalte der rechten Matrix (die insgesamt zwei Spalten umfasst).*

7.2

1. Erstellen Sie ein WebPPL-Programm, das zunächst ein Array mit zehn binären Zufallswerten (null oder eins) erzeugt und dann sowohl den Inhalt des Arrays als auch die Zahl der darin enthaltenen Einsen anzeigt.

*Antwort:*

Lösungsbeispiel

**let** b **=** Bernoulli**({**p**:** 0.5**})**

**let** binaryRandomSample **=** **function(){**

**return** **(**sample**(**b**)?** 1 **:** 0**)**

**};**

**let** random10binaries **=** **function(){**

**return** repeat**(**10**,**binaryRandomSample**)**

**};**

**let** a **=** random10binaries**()**

**let** ms1 **=** '\nDrawn random sample of ten 0 or 1 values : ' **+** a

**let** ms2 **=** '\nNumber of drawn ones : ' **+** sum**(**a**)**

**let** ms3 **=** ms1 **+** ms2

ms3

Möglicher Output des WebPPL-Beispielprogramms

7.3

1. Bestimmen Sie die Adjunkte der Adjazenzmatrix U aus dem letzten Beispiel und zeigen Sie, dass U eine unitäre Matrix ist. Berechnen Sie außerdem den Betrag der Zustände des Ausgangs-Qubits aus demselben Beispiel und ermitteln Sie die Wahrscheinlichkeit der Beobachtung von Zustand 0.

*Antwort: polynomisch*

*Betrag der Zustände des Ausgangs-Qubits: Quadratwurzel aus (((1+9)/14) + (4/14)) = 1*

*Die Wahrscheinlichkeit der Beobachtung des Systems in Zustand 0 beträgt 10/14 oder 5/7 (auf der Grundlage der obigen Berechnungen).*