Klausur Informatik II (05.07.2002)

Vorname	Klaus
Nachname	Wachtler
Matrikelnummer	12345678
Prüfer	Klaus Wachtler

Die Antworten meiner Musterlösung sind in blau gehalten, während Punktzahlen und Anmerkungen zur Bewertung in rot eingetragen sind.

Allgemeine Hinweise

Fragen mit freien Antworten

Die meisten Fragen sind durch freien (aber bitte lesbaren) Text im entsprechenden Feld zu beantworten. Der vorgesehene Platz ist **NICHT** bindend: Einerseits kann man manche Fragen auch sinnvoll beantworten, ohne den ganzen Platz zu benötigen; andererseits kann man auch auf einem zusätzlichen Blatt fortfahren, wenn man einen Roman schreiben will.

Jedes zusätzliche Blatt bitte mit Name und Matrikelnummer versehen sowie der Nummer der zugehörigen Frage, und spätestens bei der Abgabe mit den ausgeteilten Blättern zusammenheften.

Im Notfall fragen

Wenn eine Frage unklar oder falsch formuliert ist, dann lieber sofort fragen, anstatt mit meinen Fehlern Zeit zu verschwenden.

Wer eine Wissenslücke hat, die man schnell beheben könnte, und wegen dieser Lücke den Rest einer Aufgabe nicht lösen kann, kann sich einen Hinweis geben lassen, der mit dem Anschreiben angemessener Strafpunkte einhergeht. Damit kann er dann hoffentlich den Rest der Aufgabe lösen, und vermeidet dadurch Folgefehler.

Und jetzt bitte beruhigen, tief Luft holen, und viel Glück!


ENTSCHULDIGUNG, ABER HABEN SIE MEINE KATZE GESEHEN? ES IST EINE SCHWARZE KATZE. GANZ SCHWARZ. SCHWARZ.
MIT DIESER VERKRAMPFTEN EINSTELLUNG WERDEN SIE SIE NIE FINDEN.
KATZE?
WWW.NICHTLUSTIG.DE

Prüfungsfragen

1. Nach welchen Kriterien werden Algorithmen üblicherweise bewertet und untereinander verglichen?

   Die Kriterien bitte nennen und kurz erklären:

   (A) LAUFZEITEFFIZIENZ(1): wieviel Rechenzeit wird für die Problemlösung benötigt, bzw. wie hängt die benötigte Rechenzeit von der Problemgröße ab (Maschinenbefehle, I/O-Operationen)? (2)

   (b) Speichereffizienz(1): wieviel Speicher (Stack, Heap, statischer Speicher, Plattenplatz) wird für die Problemlösung benötigt (ggf. wieder in Abhängigkeit von der Problemgröße) (2)

   (c) kognitive Effizienz(1): wie verständlich oder nachvollziehbar ist der Algorithmus (wichtig zur Beurteilung der Korrektheit und zur sicheren Implementation) (2)

Summe: 9

2. Sortieren Sie bitte die Namen so in den Baum ein, daß sich ein alphabetisch korrekt sortierter *Red-Black-Tree* ergibt:

Albrecht

Fritz

Gustav

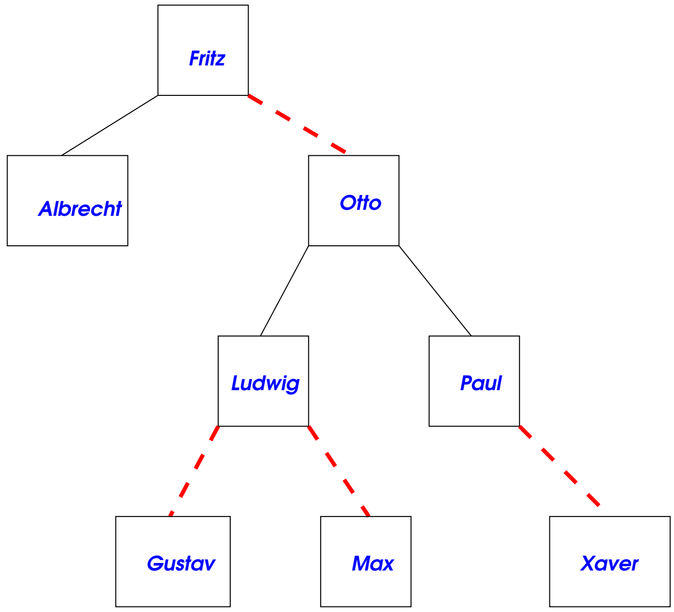
Ludwig

Max

Otto

Paul

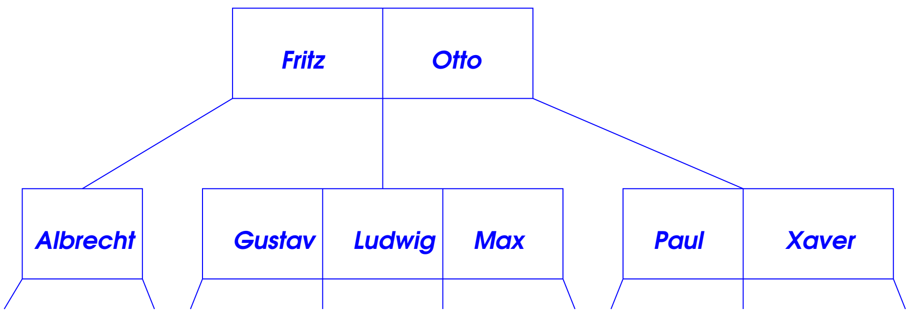
Xaver



Summe: 2

3. Aus welcher Datenstruktur entstand der obige *Red-Black-Tree*?
   Bitte den richtigen Namen angeben und die Struktur mit Inhalt zeichnen:

   AUS DIESEM 234-BAUM(2):



(5)

Summe: 7

4. Was versteht man unter *genetischen Algorithmen*, wofür können sie eingesetzt werden (eine Seite etwa sollte reichen!)?

Ein genetischer Algorithmus wird zur Optimierung verwendet. Dazu werden die variablen Größen des Problems für eine wählbare Anzahl von Lebewesen vorgegeben(1), und für jedes Lebewesen als dessen Erbinformation betrachtet(1) (analog zur Genetik)(1). Jedes der Lebewesen kann mithilfe einer Fitness-Funktion(1) als mehr oder weniger gut bewertet werden. Durch Mutation(1) und Rekombination(1) der Gene vorhandener Lebewesen werden neue geschaffen(1), während schlechte Lebewesen aussortiert werden und im weiteren nicht mehr teilhaben(1).
Einsetzbar sind genetische Algorithmen zur Optimierung aller Probleme, bei denen eine Zielfunktion (die Fitness-Funktion) aufgestellt werden kann(1). Außer der effektiven Berechenbarkeit der Zielfunktion gibt es keine weiteren Voraussetzungen(1). Allerdings erreicht man damit in der Regel nicht die optimale, sondern nur eine mehr oder weniger gute Lösung *(keine Punkte dafür, soweit in Aufgabe 7 beantwortet)*.

Summe: 10

5. Worin liegt der Unterschied zwischen *Evolutionsstrategien* und *genetischen Algorithmen*?

Bei genetischen Algorithmen werden die variablen Größen als Menge von Bits aufgefaßt, und auf dieser Ebene werden Rekombination und Mutation durchgeführt (Rekombination: Neukombination von Bitmustern, Mutation: Kippen von Bits); Erst bei der Bewertung eines Lebewesens in der fitness-Funktion wird wieder die dahinter stehende physikalische Bedeutung betrachtet. Bei Evolutionsstrategien dagegen werden die variablen Größen mit ihrer physikalischen Bedeutung beim Variieren betrachtet (Änderung: Addieren einer Differenz, Rekombination: beispielsweise Durchschnitt der Werte der Eltern bilden)

Summe: 3

6. Was bewirkt eine starke Mutation bei genetischen Algorithmen?

Zu Anfang schreitet die Evolution dadurch schneller fort; in späteren Schritten werden durch häufige Mutation auch bereits gefundene gute Lebewesen zerstört.

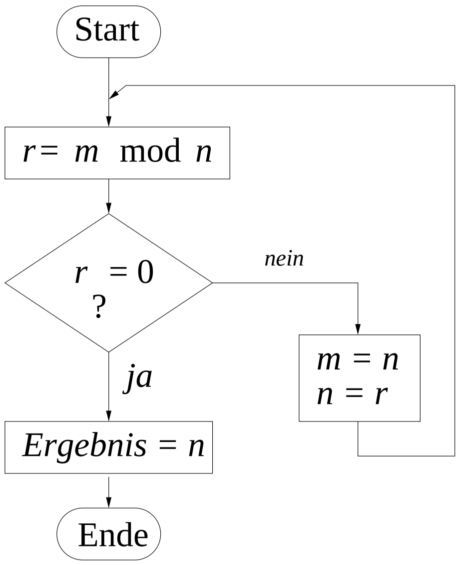
Summe: 4

7. Wann findet man bei genetischen Algorithmen normalerweise die optimale Lösung?

Nie, beziehungsweise nach unendlich langer Rechenzeit

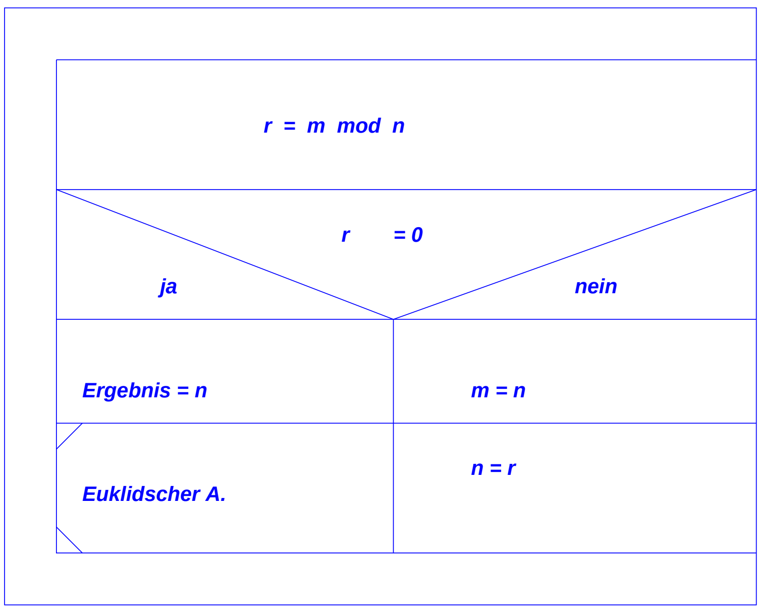
Summe: 2

8. Der im folgenden gegebene *Euklidsche Algorithmus* ist als *Nassi-Shneiderman-Diagramm* darzustellen:



Mein Nassi-Shneiderman-Diagramm:

Euklidscher Algorithmus:



Summe: 5

9. Es sei folgende Aufgabenstellung vorgegeben:

   (a) In einem Programm soll anfangs ein große Anzahl von Gemeinden G_i(mit $i = 1 \ldots N_G$) eines Landes eingelesen werden (jeweils durch zwei Koordinaten (x_i, y_i) eines Referenzpunktes der Gemeinde und einen Namen $\mathit{Ortsname}_i$ repräsentiert).

   (b) Dann soll im weiteren Programmlauf eine wahrscheinlich große Anzahl weiterer Punkte P_j(mit $j = 1 \ldots N_P$) (wieder jeweils durch ein Koordinatenpaar (x_j, y_j) repräsentiert) gelesen werden. Für jeden gelesenen Punkt P_j soll unmittelbar nach dem Einlesen herausgefunden werden, welcher der Gemeinden G_i er am nächsten liegt.
   Diese nächstgelegene Gemeinde G_i soll mit ihren Koordinaten und dem Ortsnamen ausgegeben werden.

   Um in dem obigen Abschnitt (b) zu einem gelesenen Punkt P_j die nächstgelegene Gemeinde zu finden, wird folgender Algorithmus **SucheGemeinde** angegeben:

   1. Setze `abstand_min` auf ∞
   2. Setze `i` auf 1
   3. Falls `i` größer als N_G ist: Springe zu Schritt 8.
   4. Berechne den Abstand `abstand_i` zwischen G_i und P_j:
      `abstand_i` $= \sqrt{(x_{G_i} - x_{P_j})^2 + (y_{G_i} - y_{P_j})^2}$
   5. Falls `abstand_i` kleiner ist als `abstand_min`:
      - Setze `abstand_min` auf `abstand_i`.
      - Setze `i_min` auf `i`.
   6. Erhöhe `i` um 1
   7. Gehe zu Schritt 3.
   8. Ergebnis ist $(x_{\texttt{i\_min}}, y_{\texttt{i\_min}})$ und $\mathit{Ortsname}_{\texttt{i\_min}}$

   Nun die Fragen dazu:

   (a) Erfüllt der angegebene Algorithmus die Forderung, tatsächlich die nächstgelegene Gemeinde zu finden?

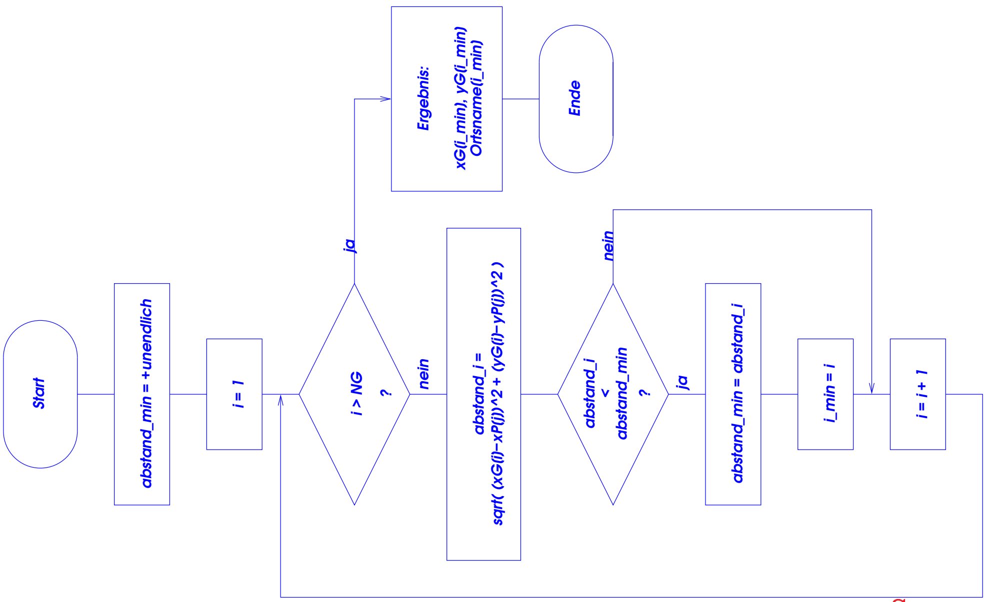
   ☒ ja(2)
   ◯ nein

Falls ja: Welche Laufzeitordnung hat der angegebene Algorithmus in Abhängigkeit von N_G (mit verständlicher Begründung)?
Falls nein: warum nicht?

$O(N_G)$(2), weil für jede Gemeinde ein Schleifendurchlauf nötig ist. Alle anderen Operationen sind von N_G unabhängig, und damit ist der dominierende Rechenaufwand proportional zu N_G(4)

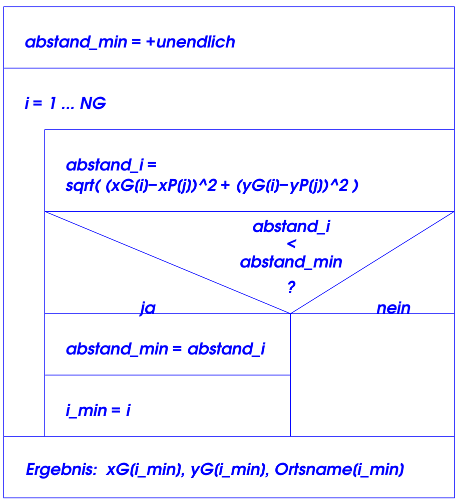
Summe: 8

(b) Stellen Sie den angegebenen Algorithmus als Flußplan dar:



Summe: 6

(c) Stellen Sie den angegebenen Algorithmus als Nassi-Shneiderman-Diagramm dar:



Summe: 6

(d) Geben Sie einen besseren Algorithmus an (bitte verständlich darstellen):

Nach dem Einlesen der Gemeinden (in Punkt (a) der Aufgabenstellung auf Seite 8) werden die Gemeinden bereits nach aufsteigenden x_i sortiert.
Zum Suchen nach der nächstgelegenen Gemeinde wird zuerst mit binärer Suche ermittelt, an welcher Stelle im Feld die in x-Richtung nächstgelegenen Orte liegen (je nach vorhandener Werte findet man so eine Stelle im Feld mit gleichem x, oder man landet zwischen zwei Einträgen mit jeweils größerem beziehungsweise kleinerem x).
Anschließend geht man ähnlich vor wie im vorgegebenen Algorithmus: man setzt ein abstand_min auf ∞, und berechnet nun für einen Ort nach dem anderen den Abstand, vergleicht ihn mit dem bisher gefundenen abstand_min und verringert letzteren, falls man einen Ort mit kleinerem Abstand gefunden hat.
Im Unterschied zum vorgegebenen Algorithmus werden die Gemeinden allerdings nicht in ihrer ursprünglichen Reihenfolge betrachtet, sondern von der gefundenen Stelle ausgehend erst alle darüber liegenden, dann alle darunterliegenden (natürlich nur, solange die betrachteten Elemente noch im Feld liegen).
Würde man diese Vergleiche fortsetzen, bis alle Elemente verglichen sind, hätte man -mit mehr Programmieraufwand- die selbe Laufzeitordnung (nämlich $O(N_G)$, weil jede Gemeinde genau einmal betrachtet werden muß).
Allerdings muß man nicht alle Gemeinden vergleichen: wenn beim Entfernen vom Startpunkt aus nach oben oder unten ein Ort erreicht wird, der in x-Richtung mehr als das bis dahin ermittelte abstand_min vom gegebenen Punkt P_j entfernt ist, können in dieser Richtung keine Gemeinden mehr im Feld stehen, deren Gesamtentfernung kleiner ist (weil ja der x-Abstand bereits größer ist als der bisher gefundene kleinste Gesamtabstand).

es werden natürlich auch andere brauchbare Ansätze akzeptiert!

Brauchbarer Ansatz: (5), verständliche Beschreibung des Algorithmus: (4)
Summe: 9

(e) Welche Laufzeitordnung hat Ihr Algorithmus in Abhängigkeit von N_G (mit verständlicher Begründung)?

Das Sortieren des Feldes geschieht nur einmal nach dem Einlesen aller Koordinaten und Ortsnamen, und geht daher in die Suche nach der nächstgelegenen Gemeinde nicht ein (weil laut Aufgabenstellung viele Punkte gelesen und der zugehörige Ort gesucht werden muß).
Auf jeden Fall muß vor dem eigentlichen Vergleich eine binäre Suche stattfinden mit $O(\log N_G)$.
Im schlechtesten Fall (der allerdings sehr unwahrscheinlich ist) müssen dann alle Gemeinden untersucht werden; dann hat man die ursprüngliche Laufzeitordnung $O(N_G)$, weil das gesamte Feld untersucht werden muß (und $O(N_G)$ gegenüber $O(\log N_G)$ dominiert).
Im besten Fall findet man bereits nach wenigen Schleifendurchläufen (in Relation zur Anzahl der Gemeinden) den kleinstmöglichen Abstand; dann hat man eine Laufzeitordnung $O(\log N_G)$.
Je nach den tatsächlichen Koordinaten liegt man also zwischen $O(\log N_G)$ und $O(N_G)$.

Anmerkung KW: Für großes N_G und nicht bewußt ungünstig konstruierte Koordinaten liegt man in der Tat recht nah bei $O(\log N_G)$

Summe: 4
(falls richtig und nachvollziehbar)

10. Gegeben ist das folgende Feld ganzer Zahlen:
25, 78, 1, 89, 14, 47, 42, 99, 8, 5, 13, 80, 70, 60, 50, 85, 35.

Sortieren Sie bitte das Feld mit *quick sort* (aufsteigend) und geben Sie dabei alle Einzelschritte des Algorithmus an.

In einem ersten Teilschritt wird das gesamte Feld betrachtet:
Zuerst wählt man ein Pivotelement, beispielsweise ein Element etwa aus der Feldmitte; das ist hier die 8. Dieses Pivotelement wird mit dem ersten Feldelement vertauscht; weiterhin merkt man sich wie weit man bisher das Feld umsortiert hat (hier mit einem /), anfangs steht diese Marke direkt nach dem an den Anfang gesetzten Pivotelement:
(8, / 78, 1, 89, 14, 47, 42, 99, 25, 5, 13, 80, 70, 60, 50, 85, 35)
Jetzt sortiert man das Feld so um, daß nach dem Pivotelement erst alle kleineren Elemente, dann alle größeren kommen. Dies erreicht man, indem ab dem zweiten Element (78) nacheinander alle Elemente mit dem Pivotelement verglichen werden; ist das Element kleiner, dann wird es gegen das Element hinter der Marke vertauscht, und die Marke eine Position nach rechts gesetzt.
Im Beispielfeld ist dies für das zweite Element 78 nicht der Fall, aber für das dritte (1). Also werden die 78 und die 1 vertauscht, und die Marke eine Position weitergesetzt:
(8, 1, / 78, 89, 14, 47, 42, 99, 25, 5, 13, 80, 70, 60, 50, 85, 35)
Die dann folgenden Elemente 89, 14, 47, 42, 99 und 25 sind alle größer als das Pivotelement; also muß nichts getauscht werden.
Dann kommt die 5, die wieder kleiner ist. Also die 5 mit dem Element hinter der Marke tauschen, und die Marke ein Element weitersetzen:
(8, 1, 5, / 89, 14, 47, 42, 99, 25, 78, 13, 80, 70, 60, 50, 85, 35)
Die restlichen Elemente (ab 13) sind wieder alle größer als das Pivotelement und es sind keine Vertauschungen nötig.
Dieser erste Teilschritt wird beendet, indem das Pivotelement am Anfang des Feldes mit dem letzten Element vor der Marke vertauscht wird:
(5, 1, 8, 89, 14, 47, 42, 99, 25, 78, 13, 80, 70, 60, 50, 85, 35)
Damit besteht das Feld aus lauter Elementen, die kleiner als das Pivotelement sind (5 bis 1), dann aus dem Pivotelement (8), dann lauter größeren Elementen (89 bis 35).
Dieser Algorithmus wird nun auf das Teilfeld links des Pivotelements und auf das Teilfeld rechts davon wiederholt angewandt, bis die Länge jedes Teilfelds 0 oder eins ist.
Also: Wählen eines Pivotelements, Vertauschen mit dem ersten Elementen und setzen der Marke (links und rechts jeweils gleichzeitig dargestellt):
(5, / 1) 8 (78, / 14, 47, 42, 99, 25, 89, 13, 80, 70, 60, 50, 85, 35)
Nach dem Durchlaufen der beiden Felder ergibt sich die Reihenfolge:

(5, 1 /) 8 (78, 14, 47, 42, 25, 13, 70, 60, 50, 35, / 99, 80, 85, 89)

Die Pivotelement werden wieder durch Vertauschen in das jeweilige Feld einsortiert (vor der Marke):

(1, 5) 8 (35, 14, 47, 42, 25, 13, 70, 60, 50, 78, 99, 80, 85, 89)

Nach dem oben beschriebenen Umsortieren ist das linke Teilfeld bereits fertig sortiert, die Länge des rechten ist nach wie vor größer als 1.

Auf jedes Teilfeld wird der Algorithmus rekursiv angewendet.

Wahl eines Pivotelements, Vertauschen mit dem ersten Element und Setzen der Marke:

1, 5, 8 (42, / 14, 47, 35, 25, 13, 70, 60, 50) 78 (80, / 99, 85, 89)

Umsortieren der Elemente:

1, 5, 8, (42, 14, 35, 25, 13, / 47, 70, 60, 50) 78 (80, / 99, 85, 89)

Einsortieren des Pivotelements:

1, 5 8 (13, 14, 35, 25) 42 (47, 70, 60, 50) 78, 80 (99, 85, 89)

Wahl eines Pivotelements, Vertauschen mit dem ersten Element und Setzen der Marke:

1, 5 8 (14, / 13, 35, 25) 42 (70, / 47, 60, 50) 78, 80 (85, / 99, 89)

Umsortieren der Elemente:

1, 5, 8 (14, 13, / 35, 25) 42 (70, 47, 60, 50 /) 78, 80 (85, / 99, 89)

Einsortieren des Pivotelements:

1, 5, 8, 13, 14 (35, 25) 42 (47, 60, 50) 70, 78, 80, 85 (99, 89)

Wahl eines Pivotelements, Vertauschen mit dem ersten Element und Setzen der Marke:

1, 5, 8, 13, 14 (35, / 25) 42 (60, / 47, 50) 70, 78, 80, 85 (99, / 89)

Umsortieren der Elemente:

1, 5, 8, 13, 14 (35, 25 /) 42 (60, 47, 50 /) 70, 78, 80, 85 (99, 89 /)

Einsortieren des Pivotelements ergibt nur noch ein Teilfeld mit mehr als einem Element:

1, 5, 8, 13, 14, 25, 35, 42 (50, 47) 60, 70, 78, 80, 85, 89, 99

Summe: **8** ***(falls vollständig und nachvollziehbar)***

Wählt man hier wieder ein Element (50) als Pivotelement und sortiert entsprechend um, sieht das Feld so aus:

1, 5, 8, 13, 14, 25, 35, 42 (50, 47 /) 60, 70, 78, 80, 85, 89, 99

Wie gewohnt wird das Pivotelement mit dem Element vor der MArke vertauscht, und das komplette Feld ist sortiert (es verbleiben keine Teilfelder mit mehr als einem Element):

1, 5, 8, 13, 14, 25, 35, 42 47, 50, 60, 70, 78, 80, 85, 89, 99

11. In dem Feld der vorigen Aufgabe soll mit *binärer Suche* nach dem Wert `12` gesucht werden; geben Sie dabei wieder alle Einzelschritte des Algorithmus an.

Binäre Suche kann nur auf einem sortierten Feld durchgeführt werden(2).
Dazu wird aus dem gesamten Feld ein Element nahe der Mitte gewählt (hier der Wert 47); der gesuchte Wert 12 ist kleiner, und müßte folglich im linken Teilfeld zu finden sein (1, 5, 8, 13, 14, 25, 35, 42). Hier wird wieder ein Element etwa aus der Mitte genommen (13), und mit dem gesuchten Wert verglichen. Weil 12 kleiner ist als 13, muß wieder im linken Teilfeld (1, 5, 8) gesucht werden. Das dortige mittlere Element 5 ist kleiner als die gesuchte 12, also wird die rechte Hälfte betrachtet (die nur noch aus einem Element 8 besteht). 8 ist kleiner als 12, also muß im Teilfeld rechts der 8 gesucht werden. Dieses Teilfeld ist leer, also ist die Suche beendet: der Wert 12 ist nicht im ursprünglichen Feld enthalten.

Summe:* 7 *(falls vollständig und nachvollziehbar)

Und bitte nicht vergessen, **alle** Blätter abzugeben (zusätzliche Blätter mit Name und Matrikelnummer)!