Name: Mohamad Raafat Baki

Datum:09.05.2020

3.1. Selection Sort:

3.1.1 Selection Sort in absteigender Reihenfolge:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | A[1] Array A A[6] | | | | | |
| Runde | 3 | 9 | 6 | 1 | 5 | 4 |
| 1 | 9 | 3 | 6 | 1 | 5 | 4 |
| 2 | 9 | 6 | 3 | 1 | 5 | 4 |
| 3 | 9 | 6 | 5 | 1 | 3 | 4 |
| 4 | 9 | 6 | 5 | 4 | 3 | 1 |
| 5 | 9 | 6 | 5 | 4 | 3 | 1 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | A[1] Array A A[6] | | | | | |
| Runde | 3 | 9 | 6 | 1 | 5 | 4 |
| 1 | 1 | 9 | 6 | 3 | 5 | 4 |
| 2 | 1 | 3 | 6 | 9 | 5 | 4 |
| 3 | 1 | 3 | 4 | 9 | 5 | 6 |
| 4 | 1 | 3 | 4 | 5 | 9 | 6 |
| 5 | 1 | 3 | 4 | 5 | 6 | 9 |

3.1.2 Selection Sort in aufsteigender Reihenfolge:

3.1.3

var A = [8,6,1,7,2,3,5,4]

func printA(A[1..n])

for i = 1 to n

print A[i]

func selection\_sort( A[1..n] )

for k = 1 to n-1

var j = k

for i = k+1 to n

if A[i] < A[j]

j = i

var t = A[k]

A[k] = A[j]

A[j] = t

selection\_sort(A)

printA(A)

Wie vieleVergleicheführt der Algorithmus aus, wenn das Array:

a) die Länge n=1 hat :

Keine vergleiche.

b) die Länge n=2 hat und die Zahlen bei der Eingabe in der richtigen Reihenfolge enthält?

Ein vergleich.

c) die Länge n=3 hat und die Zahlen bei der Eingabe in der richtigen Reihenfolge enthält?

Drei vergleiche .

d) die Länge n=4 hat und die Zahlen bei der Eingabe in der richtigen Reihenfolge enthält ?

6 vergleiche.

e) die Länge n=3 hat und die Zahlen bei der Eingabe in der umgekehrten Reihenfolge enthält ?

Drei vergleiche .

f) die Länge n=4 hat und die Zahlen bei der Eingabe in der umgekehrten Reihenfolge enthält ?

6 vergleiche.

3.2. Insertion Sort:

3.2.1. Insertion Sort in absteigender Reihnfolge:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | A[1] Array A A[6] | | | | | |
| Runde | 3 | 9 | 6 | 1 | 5 | 4 |
| 1 | 9 | 3 | 6 | 1 | 5 | 4 |
| 2 | 9 | 6 | 3 | 1 | 5 | 4 |
| 3 | 9 | 6 | 3 | 1 | 5 | 4 |
| 4 | 9 | 6 | 5 | 3 | 1 | 4 |
| 5 | 9 | 6 | 5 | 4 | 3 | 1 |

3.2.2 Insertion Sort in aufsteigendender Reihenfolg:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | A[1] Array A A[6] | | | | | |
| Runde | 3 | 9 | 6 | 1 | 5 | 4 |
| 1 | 3 | 9 | 6 | 1 | 5 | 4 |
| 2 | 3 | 6 | 9 | 1 | 5 | 4 |
| 3 | 1 | 3 | 6 | 9 | 5 | 4 |
| 4 | 1 | 3 | 5 | 6 | 9 | 4 |
| 5 | 1 | 3 | 4 | 5 | 6 | 9 |

3.2.3

var A = [8,6,1,7,2,3,4,5]

func printA(A[1..n])

for i = 1 to n

print A[i]

func insertion\_sort(A[1..n])

for i = 2 to n

var x = A[i]

var j = i - 1

while (j >= 1) && (A[j] > x)

A[j+1] = A[j]

j = j - 1

A[j+1] = x

insertion\_sort(A)

printA(A)

Wie viele Vergleiche führt der Algorithmus aus, wenn das Array:

a) die Länge n=1 hat :

Keine vergleiche.

b) die Länge n=2 hat und die Zahlen bei der Eingabe in der richtigen Reihenfolge enthält?

1 vergleich.

c) die Länge n=3 hat und die Zahlen bei der Eingabe in der richtigen Reihenfolge enthält?

2 vergleiche .

d) die Länge n=4 hat und die Zahlen bei der Eingabe in der richtigen Reihenfolge enthält ?

3 vergleiche.

e) die Länge n=3 hat und die Zahlen bei der Eingabe in der umgekehrten Reihenfolge enthält?

4 vergleiche .

f) die Länge n=4 hat und die Zahlen bei der Eingabe in der umgekehrten Reihenfolge enthält?

9 vergleiche.

3.3. Sortieren

3.3.1

3.3.1. Wie viele elementare Schritte (1 Zeile == 1 Zeiteinheit) macht in Groß-𝒪 Notation:

a) Selection Sort im besten Fall (best-case): O(n\*2).

b) Selection Sort im schlechtesten Fall (worst-case) : O(n\*2).

c) Insertion Sort im besten Fall (best-case : O(n).

d) Insertion Sort im schlechtesten Fall (worst-case) : **O(n^2).**

3.3.2

Nennen Sie je ein Beispiel für ein Sortierverfahren aus der Vorlesung, bei der die Anzahl der elementaren Schritte :

a) im schlechtesten Fall quadratisch 𝒪(𝑛 2 ) ist : Insertion sort.

b) im schlechtesten Fall 𝒪(𝑛 ∙ log(𝑛)) ist : Mergesort.

c) von den Eingabedaten unabhängig ist : **Selection sort.**

d) von den Eingabedaten abhängig ist : Insertion Sort.

e) im besten Fall linear 𝒪(𝑛) ist : **Insertion sort.**

f) im besten Fall quadratisch 𝒪(𝑛 2 ) ist : **Selection  sort.**

3.3.3. Betrachten Sie den folgenden Algorithmus:

maybe4( A[1..4] ):

if A[1] > A[2]

tausche( A, 1, 2 )

if A[3] > A[4]

tausche( A, 3, 4 )

if A[1] > A[4]

tausche( A, 1, 4 )

if A[2] > A[3]

tausche( A, 2, 3 )

a) Nennen Sie eine Eingabe, die dieser Algorithmus korrekt in aufsteigender Reihenfolge sortiert.

A= [2,1,4,3].

b) Sortiert dieser Algorithmus jede Eingabe korrekt in aufsteigender Reihenfolge? Begründen Sie kurz ihre Entscheidung!

Nein, da dieser Algorithmus nur zahlen im Index von 1-4 durchgeht, kann eine Arraylänge >4 nicht weiter sortiert werden.

3.3.4. Schreiben Sie einen Algorithmus twosort( A[1..n] ) in Pseudocode, der jedes Array korrekt aufsteigend sortiert, das nur die Zahlen 0 und 1 enthält und der im schlechtesten Fall lineare Laufzeit 𝒪(𝑛) hat. Sie dürfen nur 𝒪(1) zusätzlichen Speicher benutzen.

var A = [0,1,0,0,1,1]

func printA(A[1..n])

for i=1 to n

print A[i]

func twosort(A[1..n])

var t=0

for i = 1 to n

t = t + A[i]

for j = 1 to n

if t > 0

A[n-j+1] = 1

t = t - 1

else

A[n-j+1] = 0

return A

twosort(A)

printA(A)

3.4. Suchen

3.4.1 Ein Array A[1..n] der Länge n=15 enthalte die folgenden Zahlen:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Indexi | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| A[i] | 12 | 28 | 96 | 13 | 43 | 29 | 54 | 68 | 93 | 17 | 23 | 39 | 42 | 88 | 72 |

a) Wie viele Vergleiche benötigt man für eine lineare Suche von links nach rechts nach 29?

6 vergleiche.

b) Wie viele Vergleiche benötigt man für eine lineare Suche von links nach rechts nach 88?

14 vergleiche .

c) Sortieren Sie das Array in aufsteigender Reihenfolge, tragen Sie das Ergebnis ein:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Indexi | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| A[i] | 12 | 13 | 17 | 23 | 28 | 29 | 39 | 42 | 43 | 54 | 68 | 72 | 88 | 93 | 96 |

d) Tragen Sie den Verlauf einer binären Suche nach der Zahl z=29 in folgende Tabelle ein, notieren Sie dabei für jeden Suchschritt die untere Grenze a, die obere Grenze b und die Mitte m. Unter Vergleich tragen Sie bitte die beiden zu vergleichenden Werte und ein, je nachdem wie der Vergleich in diesem Schritt ausgeht:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Schritt | a | b | c | Vergleich  (<, = oder > ) |
| 1 | 1 | 15 | 8 | m > z |
| 2 | 1 | 7 | 4 | m < z |
| 3 | 5 | 7 | 6 | M = z |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

e) Tragen Sie den Verlauf einer binären Suche nach der Zahl z=88 in folgende Tabelle ein, notieren Sie dabei für jeden Suchschritt die untere Grenze a, die obere Grenze b und die Mitte m. Unter Vergleich tragen Sie bitte die beiden zu vergleichenden Werte und ein, je nachdem wie der Vergleich in diesem Schritt ausgeht:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Schritt | a | b | c | Vergleich  (<, = oder > ) |
| 1 | 1 | 15 | 8 | m < z |
| 2 | 13 | 15 | 14 | m > z |
| 3 | 12 | 14 | 13 | M = z |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

3.5

3.5.1. Anzahl der Nullen in einem sortierten Array

a) Gesucht ist ein Algorithmus nullen( A(1..n] ) in Pseudocode der die Anzahl der in A enthaltenen Nullen berechnet. Dabei ist A ein bereits aufsteigend sortiertes Array natürlicher Zahlen ≥0. Beispiel: Für A = [0|0|2|7] ist das Ergebnis 2 und für A=[1|2|3|5|7] ist das Ergebnis 0. Die Laufzeit des Algorithmus soll deutlich besser als im allgemeinen Fall eines unsortierten Arrays sein. Welche Laufzeit hat der von Ihnen entwickelte Algorithmus größenordnungsmäßig?

b)\* Erweitern Sie den Algorithmus so, dass er eine zusätzliche Zahl x als Eingabe hat und die Anzahl der Vorkommen von x im aufsteigend sortierten Array A ermittelt.

a) var A = [0,0,1,2,3]

func nullen(A[1..n])

var x = 0

for i = 1 to n

if A[i] == 0

x = x + 1

return x

print nullen(A)

3.5.2.\*\* Gleichzeitiges Suchen von Minimum und Maximum in einem Array

Gegeben ist ein Array A[1..n] ganzer Zahlen. Gesucht sind das Minimum und das Maximum unter den Elementen von A. Vereinfachend sei n eine Zweierpotenz, also n = 2k für eine natürliche Zahl k. Der einfache Ansatz, die Algorithmen min(A[1..n]) und max(A[1..n]) hintereinander Aufzurufen funktioniert und benötigt insgesamt 2∙(n-1) Vergleiche. Finden Sie einen Algorithmus, der deutlich weniger Vergleiche benötigt! Hinweis: Eine optimale Lösung benötigt nur ca. 1,5∙n Vergleiche.

var A = [0,1,2,3]

func MinMax(A[1..n])

var max = A[1]

var min = A[1]

for i = 2 to n

if A[i] > max

max = A[i]

else

if A[i] < min

min = A[i]

print "max="+max

print "min="+min

MinMax(A)