Aufgabe 1:

Element „57“ in randomisierte Höhe 4

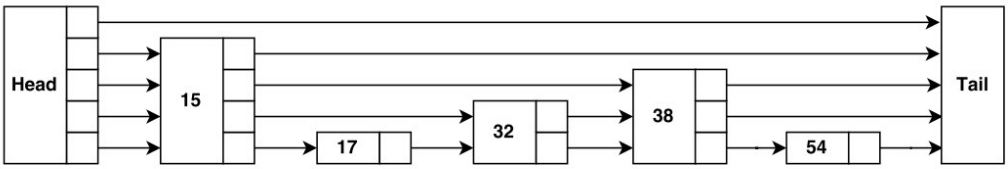
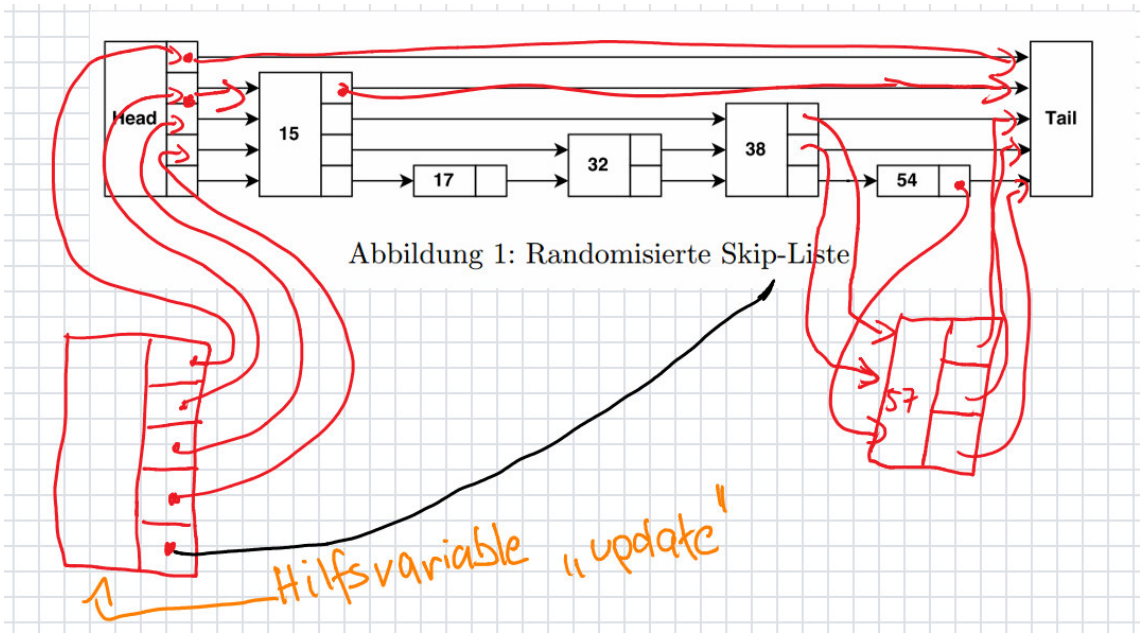
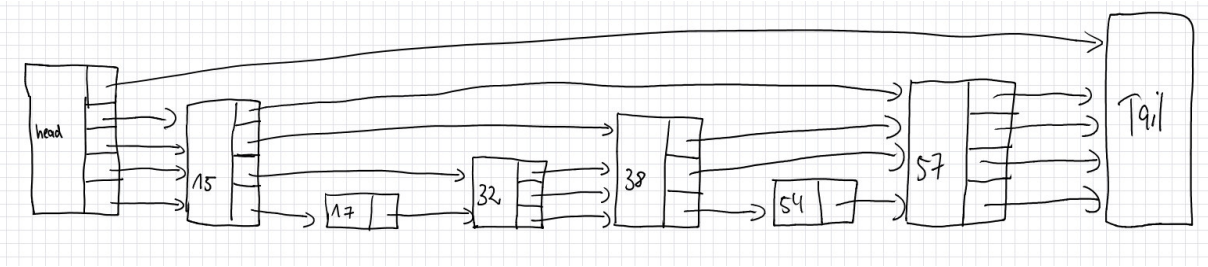


Abbildung 1: Randomisierte Skip-Liste

Mit der Hilfsvariable " update"



Neue randomisierte Skip-Liste mit dem Element „57“



Das Element „57“ wird zwischen dem Element „54“ und dem Tail eingefügt. Die randomisierte Höhe von „57“ ist 4, die Höhen der anderen verändern sich nicht. Es wird eine Hilfsvariable „Update“ benötigt, da bei der Suche die alten Referenzen gespeichert werden. Diese müssen dann am Ende rückwärts ausgelesen und neu zugewiesen.
Beim ausgehenden Knoten „54“ geht die Zuweisung an das neue Element „57“ hin und nicht mehr zu Tail. Bei dem Element „38“ müssen die zwei oberen Ebenen nicht mehr auf Tail zuweisen, sondern auf das neue Element „57“.
Die ausgehenden Knoten von „57“ verweisen alle auf Tail.

X = 57
SkipNode update = new SkipNode[5]

neuehoehe = 4 *(schon gegeben)*
SkipNode p = head

For-Schleife:

- i = 4:
  - p.next[4].elem = tail > x → *schon fertig*
  - update[4] = head
- i = 3:
  - p.next[3].elem = 15 < x → p = p.next[3] = 15
  - p.next[3].elem = tail > x → *fertig*
  - update[3] = 15
- i = 2
  - p.next[2].elem = 38 < x → p = p.next[2] = 38
  - p.next[2].elem = tail > x → *fertig*
  - update[2] = 38
- i = 1
  - p.next[1].elem = tail > x → *schon fertig*
  - update[1] = 38
- i = 0
  - p.next[0].elem = 54 < x → p = p.next[0] = 54
  - p.next[0].elem = tail > x → *fertig*
  - update[0] = 54
  -

p = p.next[0] = tail
p.elem =! X → passt

SkipNode neu = new SkipNode(5)
Neu.elem = x

For-Schleife:

- i = 0
  - neu.next[0] = update[0].next[0] = tail
  - update[0].next[0] = neu *(update[0] = 54)*
- i = 1
  - neu.next[1] = update[1].next[1] = tail
  - update[1].next[1] = neu *(update[1] = 38)*
- i = 2
  - neu.next[2] = update[2].next[2] = tail
  - update[2].next[2] = neu *(update[2] = 38)*
- i = 3
  - neu.next[3] = update[3].next[3] = tail
  - update[3].next[3] = neu *(update[3] = 15)*
- i = 4
  - neu.next[4] = update[4].next[4] = tail
  - update[4].next[4] = neu *(update[4] = head)*
-

<u>Aufgabe 2</u>

$$[“Hund“, “Katze“, “Maus“, “Tiger“, “Hirsch“, “Hase“, “Fisch“]$$

Ges.:

$$\text{“}Fisch\text{“}, \text{“}Katze\text{“}, \text{“}Tiger\text{“}, \text{“}Maus\text{“}, \text{“}Katze\text{“}$$

a) MF-Regel:

Fisch – 7 Zugriffe
Fisch, Hunde, Katze, Maus, Tiger, Hirsch, Hase
Katze – 3 Zugriffe
Katze, Fisch, Hund, Maus, Tiger, Hirsch, Hase
Tiger – 5 Zugriffe
Tiger, Katze, Fisch, Hund, Maus, Hirsch, Hase
Maus – 4 Zugriffe
Maus, Tiger, Katze, Fisch, Hund, Hirsch, Hase
Katze – 3 Zugriffe
Katze, Maus, Tiger, Fisch, Hund, Hirsch, Hase
Gesamt: 7+3+5+4+3 = 22

b) T-Regel

Fisch – 7 Zugriffe
Hund, Katze, Maus, Tiger, Hirsch, Fisch, Hase
Katze – 2 Zugriffe
Katze, Hund, Maus, Tiger, Hirsch, Fisch, Hase
Tiger – 4 Zugriffe
Katze, Hund, Tiger, Maus, Hirsch, Fisch, Hase
Maus – 4 Zugriffe
Katze, Hund, Maus, Tiger, Hirsch, Fisch, Hase
Katze – 1 Zugriff, Liste bleibt bestehen
Gesamt: 7+2+4+4+1 = 19

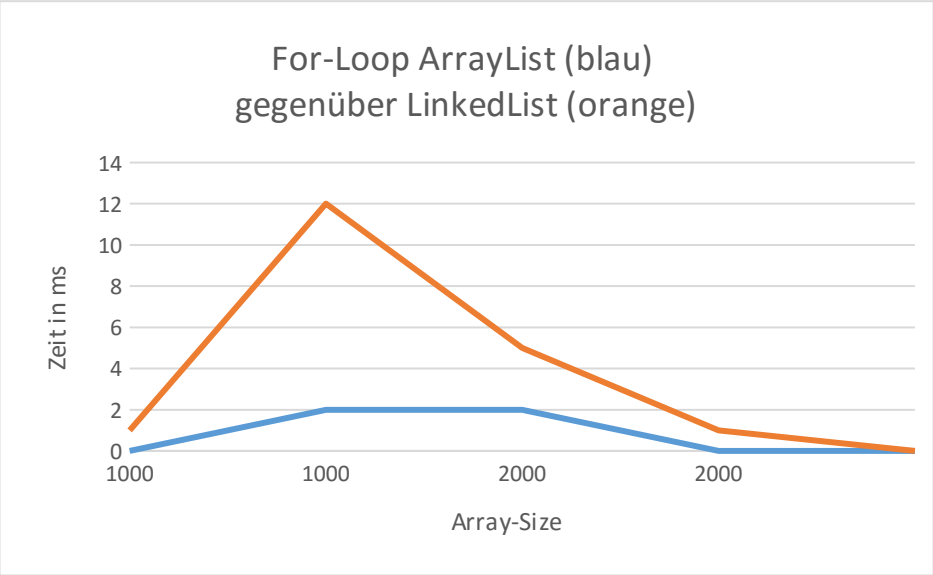
c) FC-Regel:

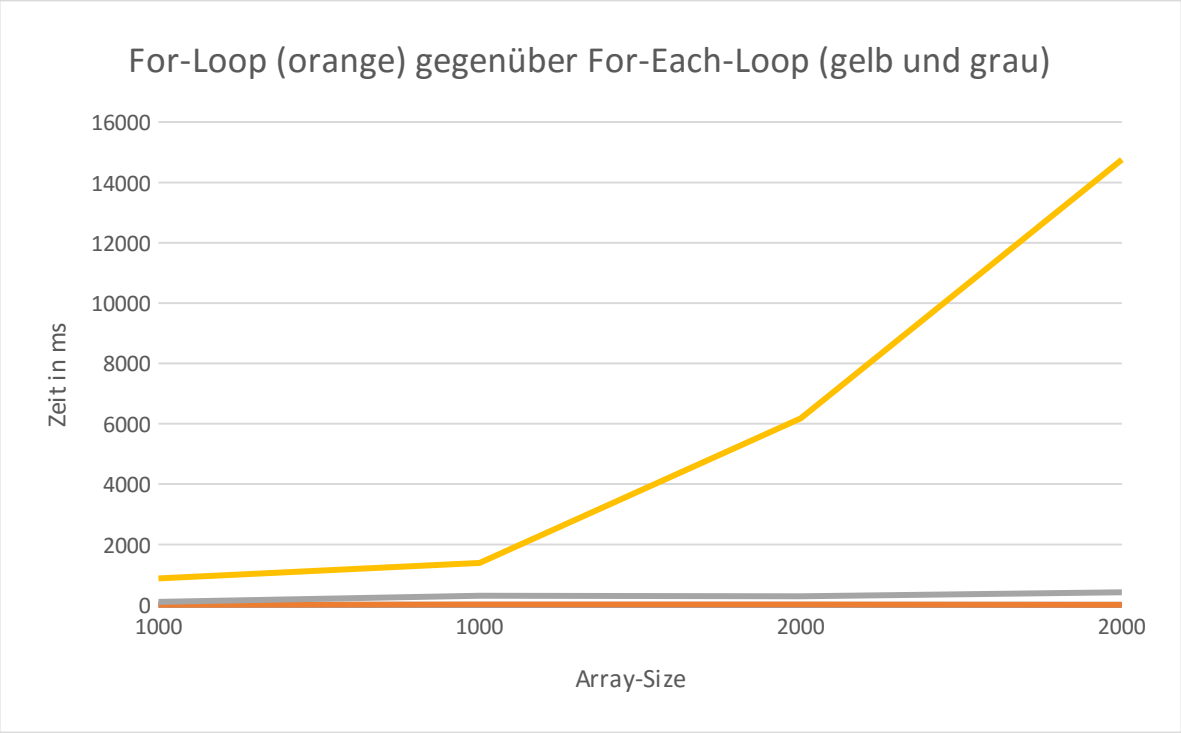
Element	“Hund“	“Katze“	“Maus“	“Tiger“	“Hirsch“	“Hase“	“Fisch“
Häufigkeitszähler	5	4	3	3	2	0	0

Fisch – 7 Zugriffe
Hund 5, Katze 4, Maus 3, Tiger 3, Hirsch 2, Fisch 1, Hase 0
Katze – 2 Zugriffe
Hund 5, Katze 5, Maus 3, Tiger 3, Hirsch 2, Fisch 1, Hase 0
Tiger – 4 Zugriffe
Hund 5, Katze 5, Tiger 4, Maus 3, Hirsch 2, Fisch 1, Hase 0
Maus – 4 Zugriffe
Hund 5, Katze 5, Tiger 4, Maus 4, Hirsch 2, Fisch 1, Hase 0
Katze – 2 Zugriffe
Katze 6, Hund 5, Tiger 4, Maus 3, Hirsch 2, Fisch 1, Hase 0
Gesamt: 7+2+4+4+2=19

		For-Loop		For-Each-Loop	
Arraysize	**Iterations**	**Array-List**	**Linked-List**	**Array-List**	**Linked-List**
1000	500	0	1	100	770
1000	1000	2	10	292	1088
2000	1000	2	3	274	5904
2000	2000	0	1	417	14342

Allgemein ist die For-Loop viel effizienter für beide Listenarten.
Die ArrayList ist je größer das Array ist effizienter als die LinkedList.





Array-List- For-Each: 417.0 ms

Array-List- For: 0.0 ms

Linked-List For-Each: 14342.0 ms

Linked-List For: 1.0 ms

Aufgabe4 - Benchmark

	n	runs	time in ms
linear	10000	100000	205
binary			9
linear	100000	100000	2285
binary			12
linear	1000000	100000	26517
binary			15
linear	100000	1000000	23546
binary			72
linear	1000000	1000000	256588
binary			101