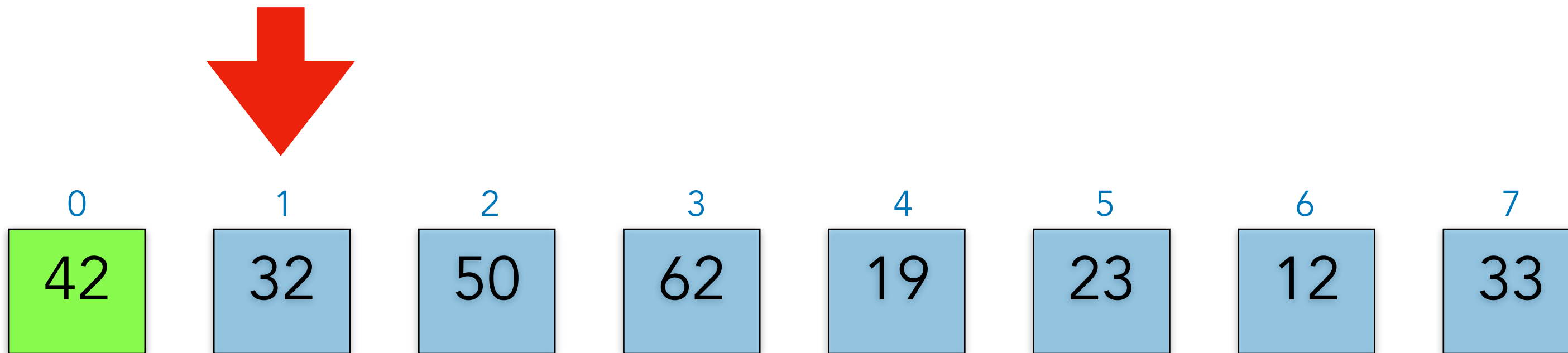


5. Sortieren und Suchen

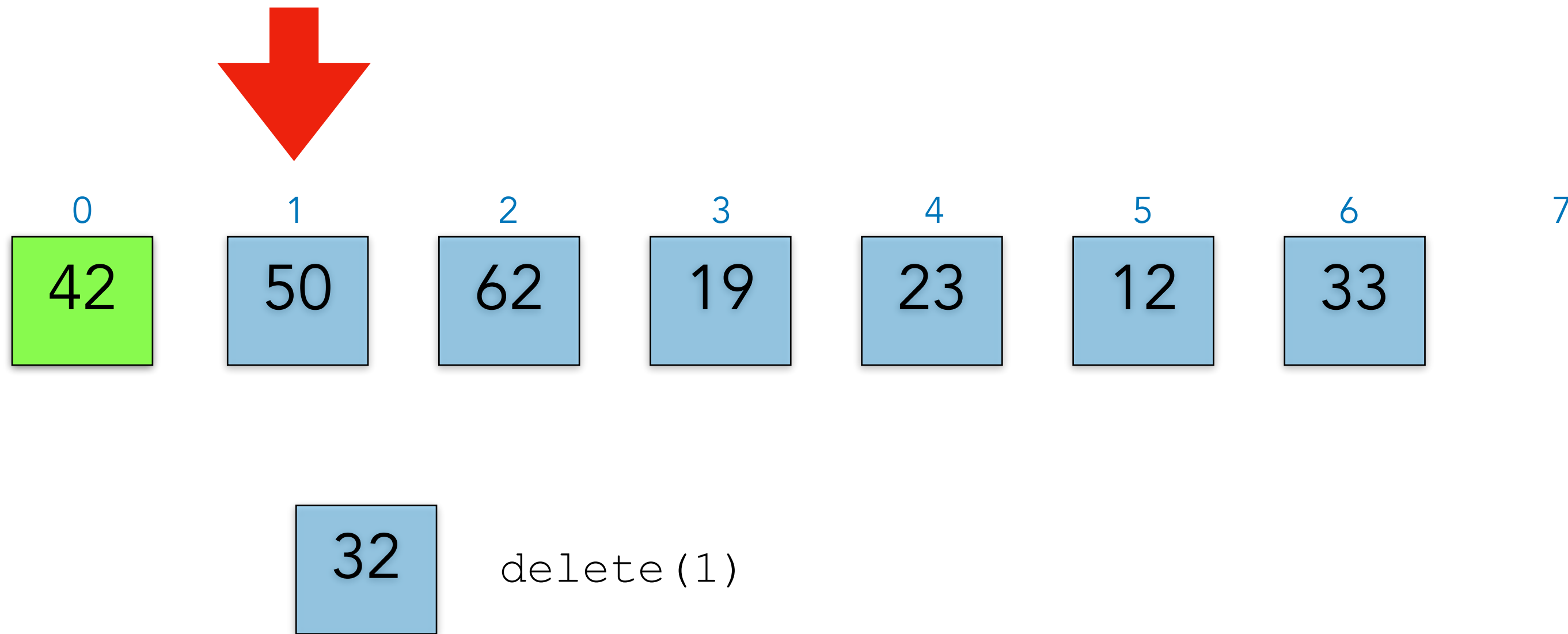
5.4 Der Insertionsort

Erster Ansatz, noch nicht optimal

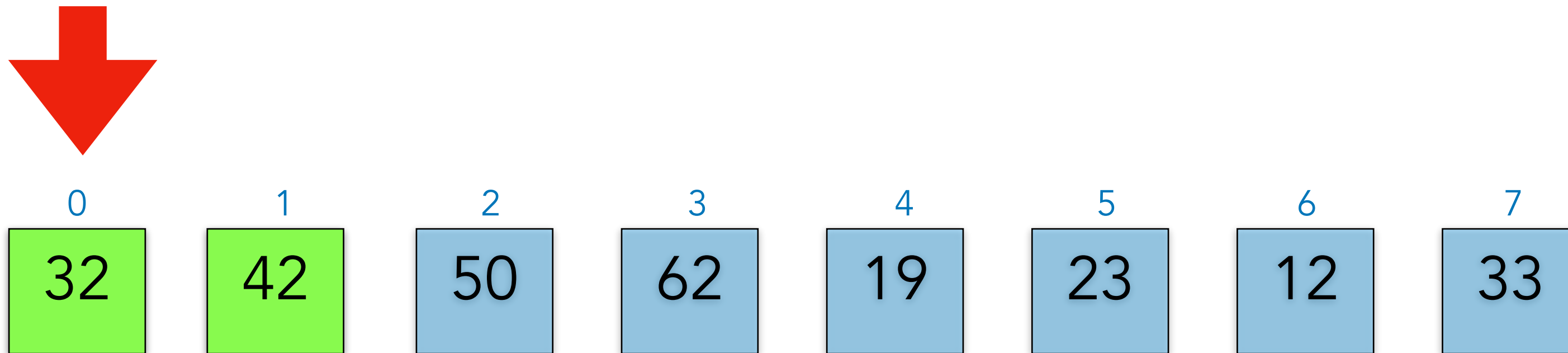
5.4 Insertionsort



5.4 Insertionsort

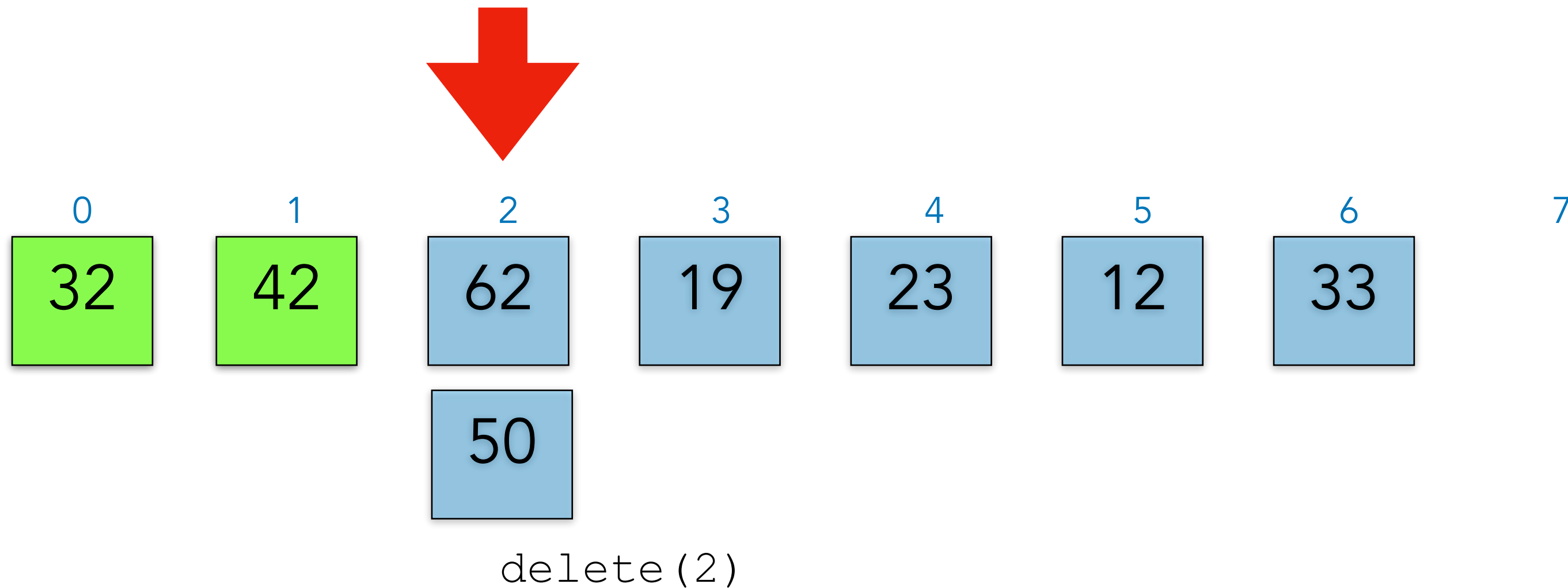


5.4 Insertionsort

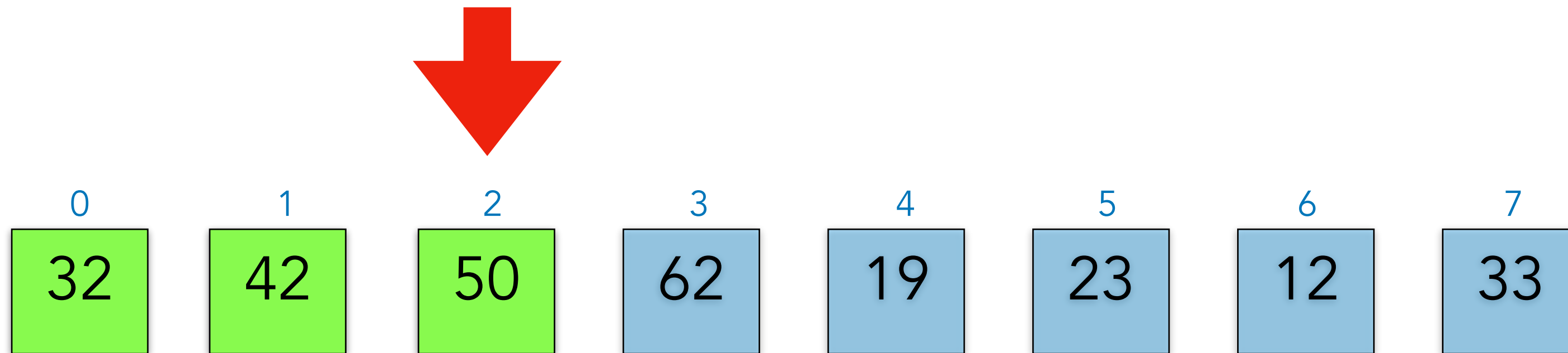


`insert(0)`

5.4 Insertionsort

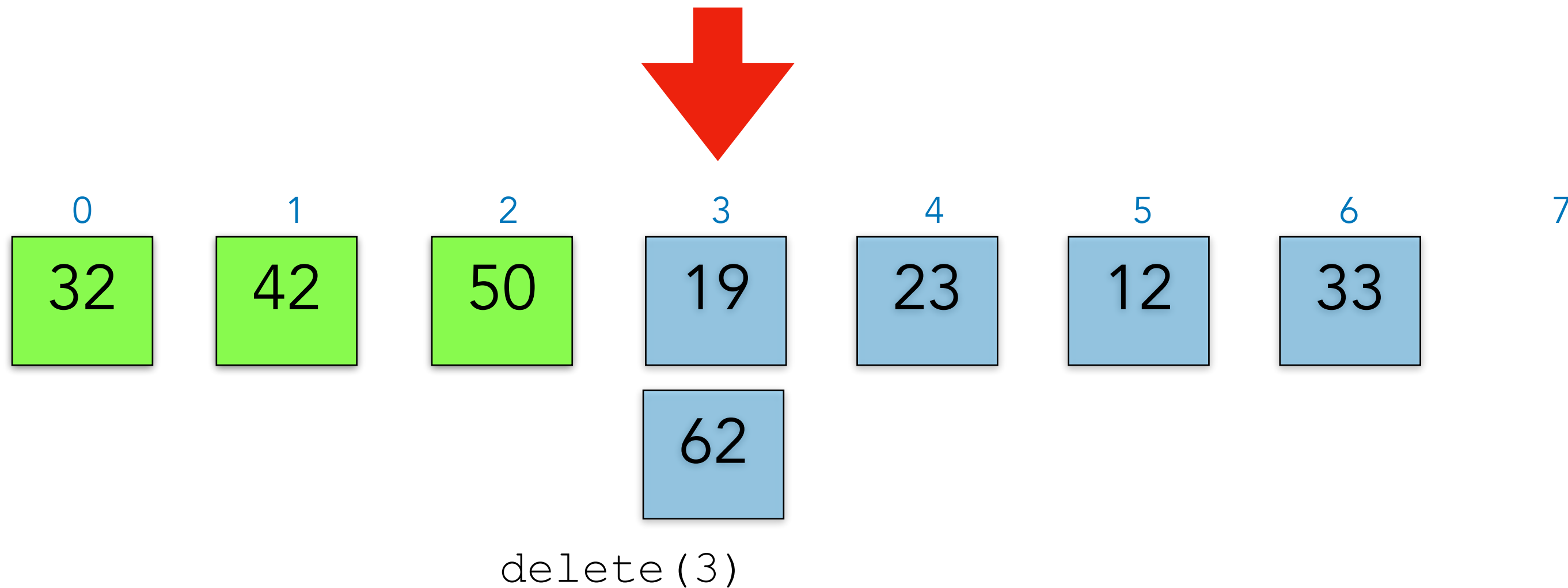


5.4 Insertionsort

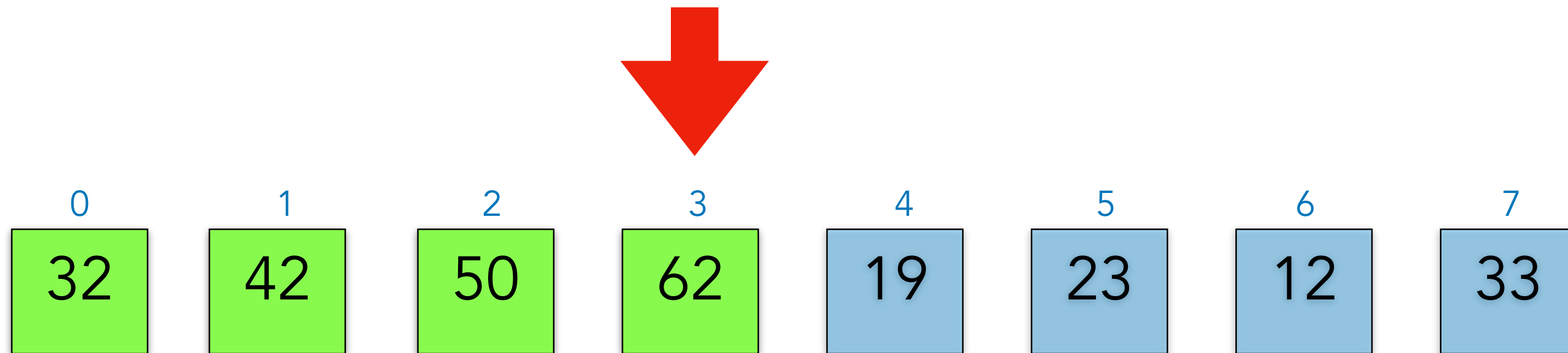


insert (2)

5.4 Insertionsort

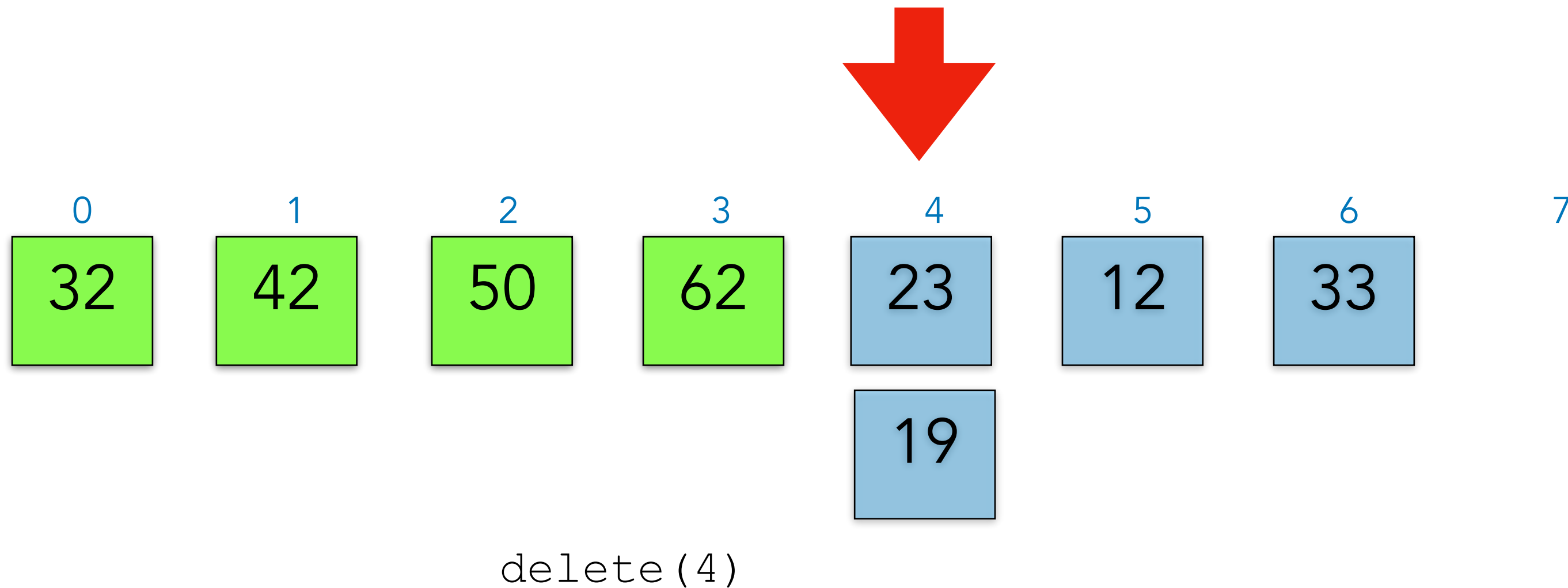


5.4 Insertionsort

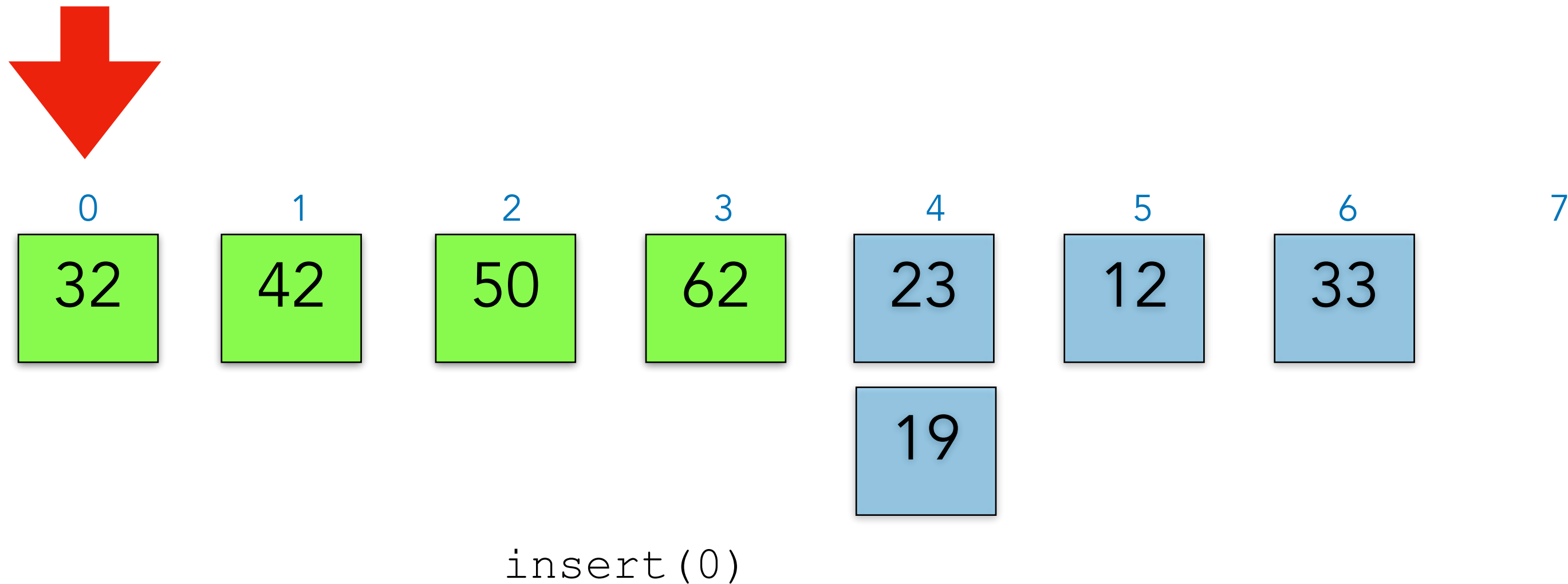


`insert(3)`

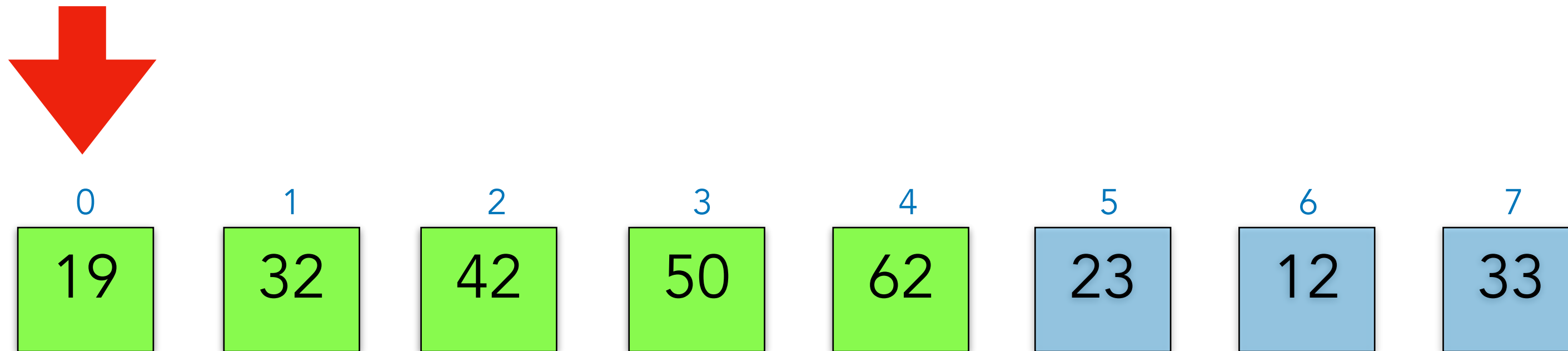
5.4 Insertionsort



5.4 Insertionsort



5.4 Insertionsort

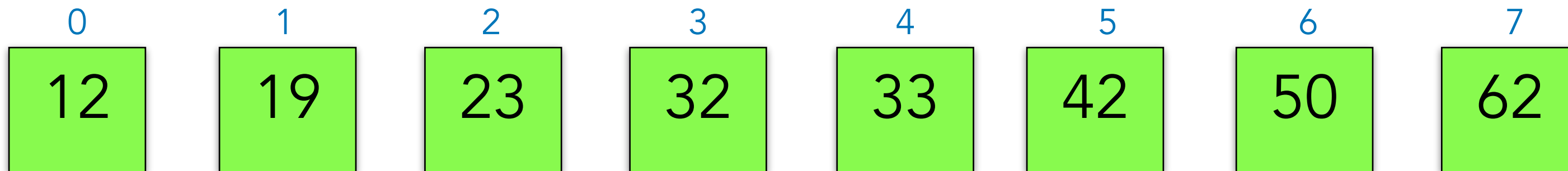


`insert(0)`

und so weiter ...

5.4 Insertionsort

Dieses Vorgehen ist alles andere als optimal!

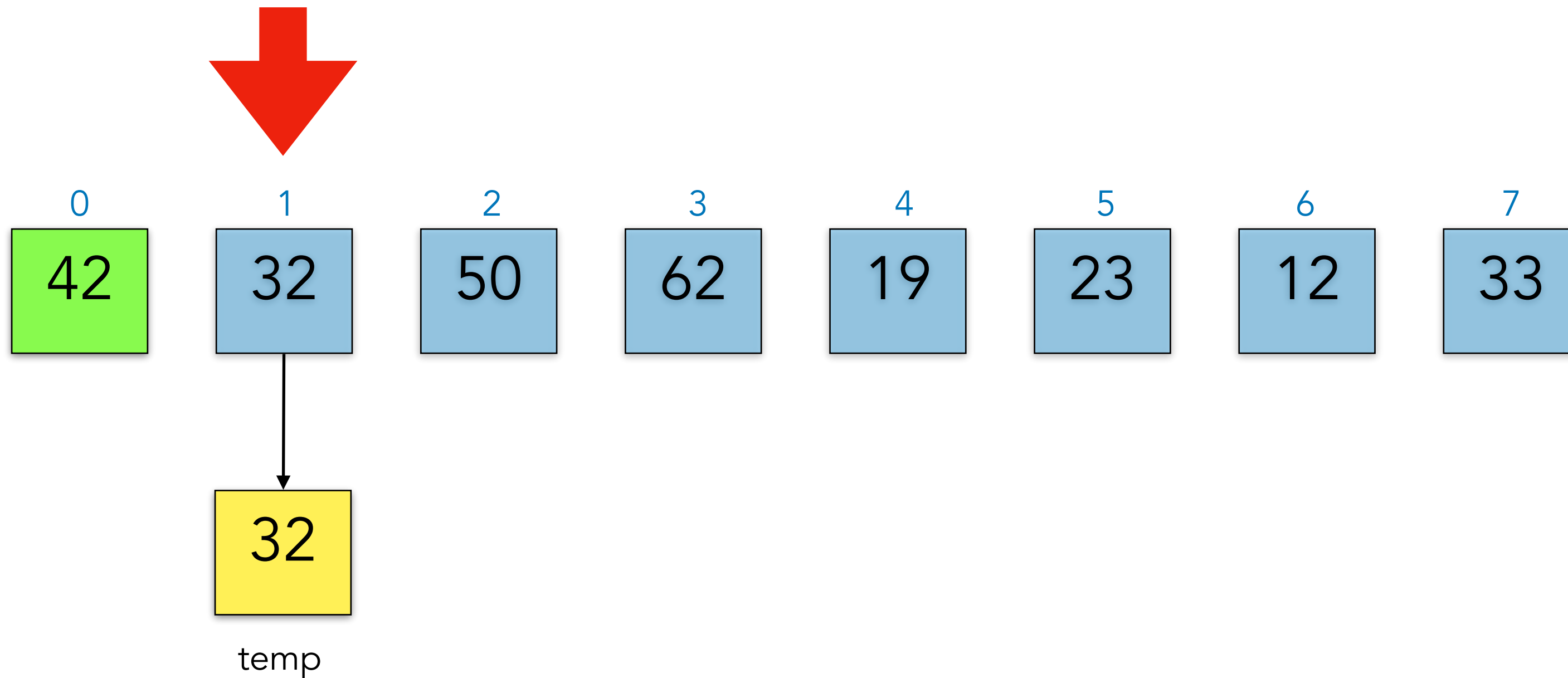


5. Sortieren und Suchen

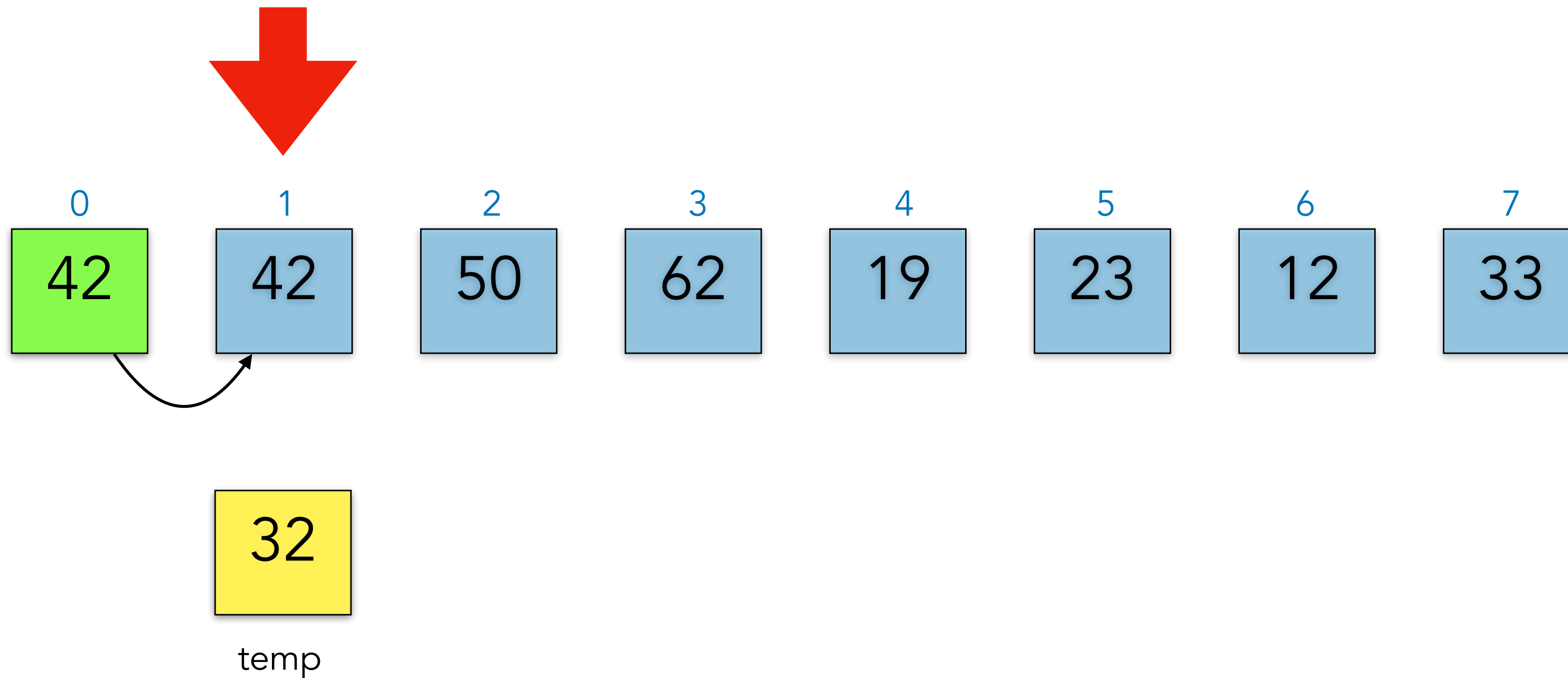
5.4 Der Insertionsort

Zweiter Ansatz, besser!

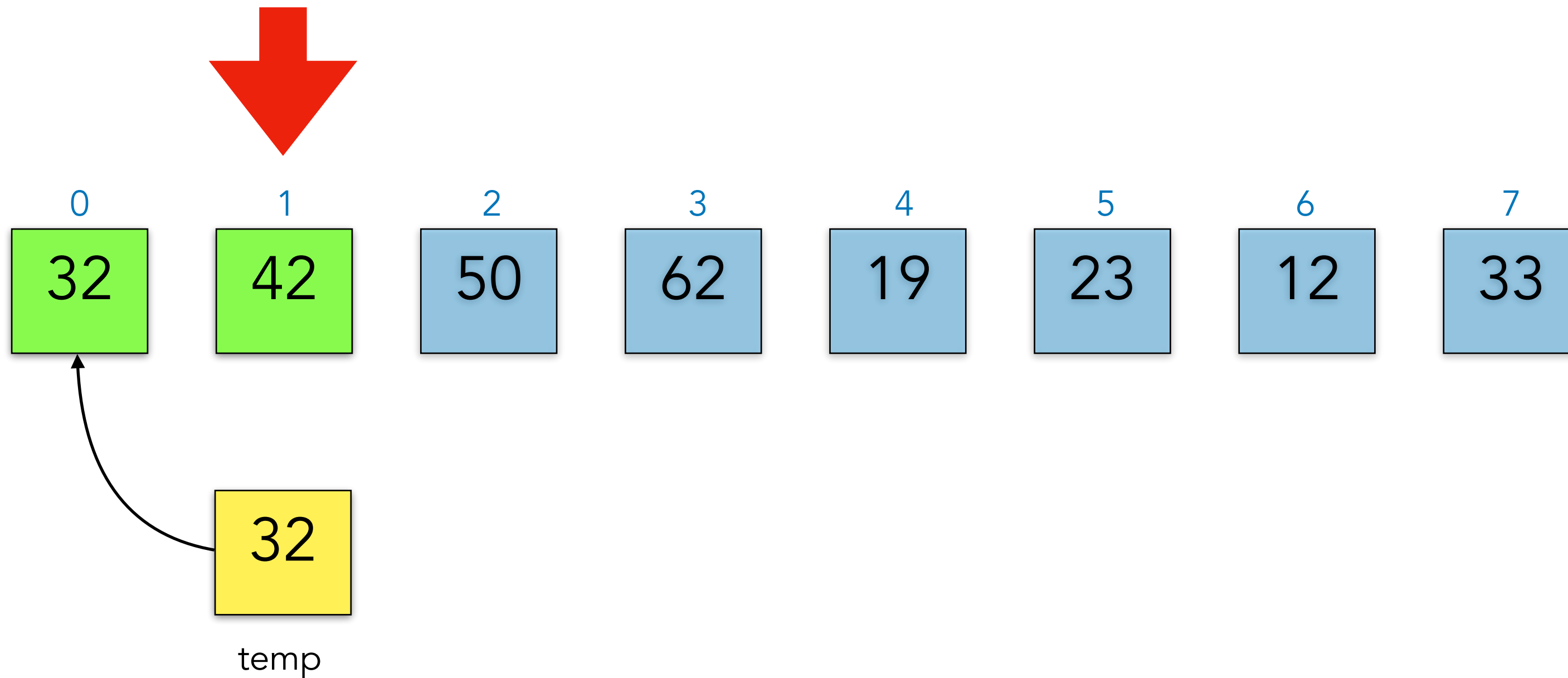
5.4 Insertionsort



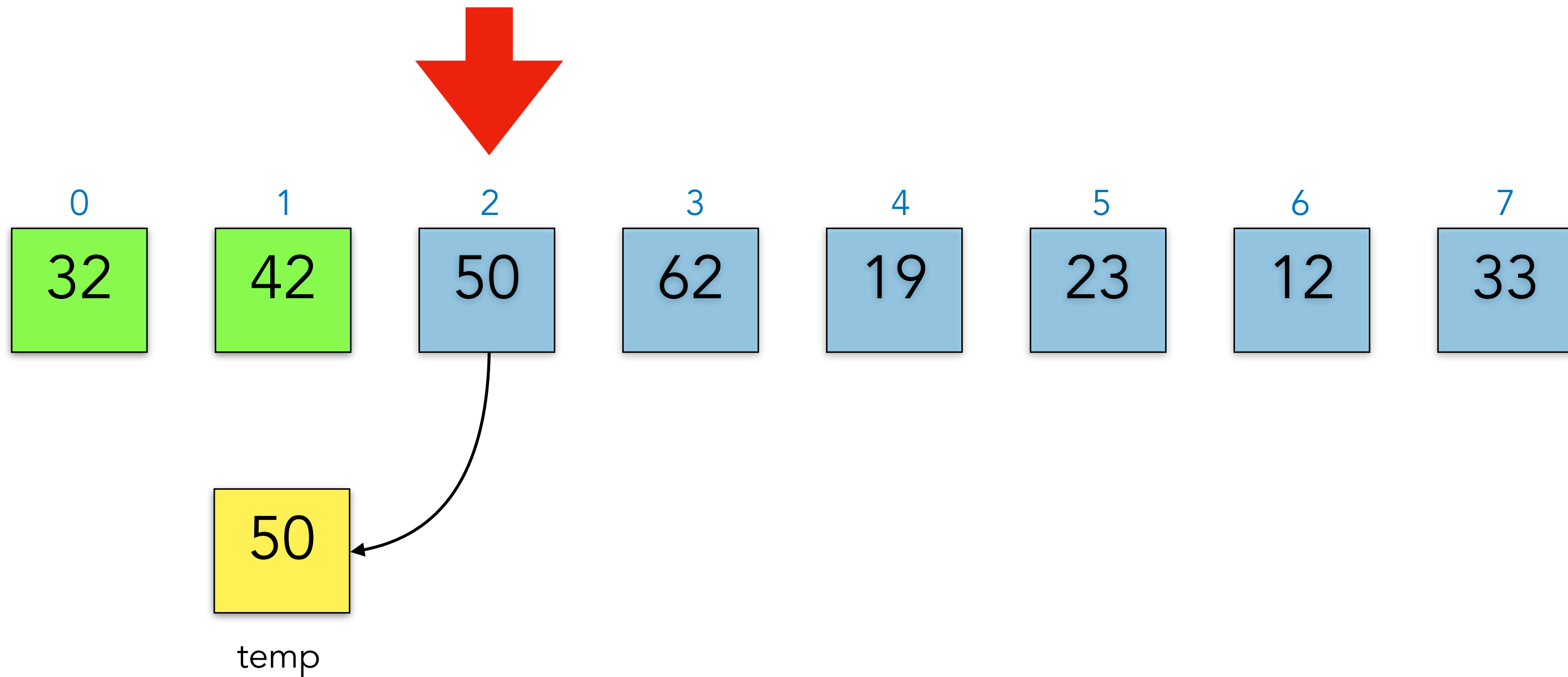
5.4 Insertionsort



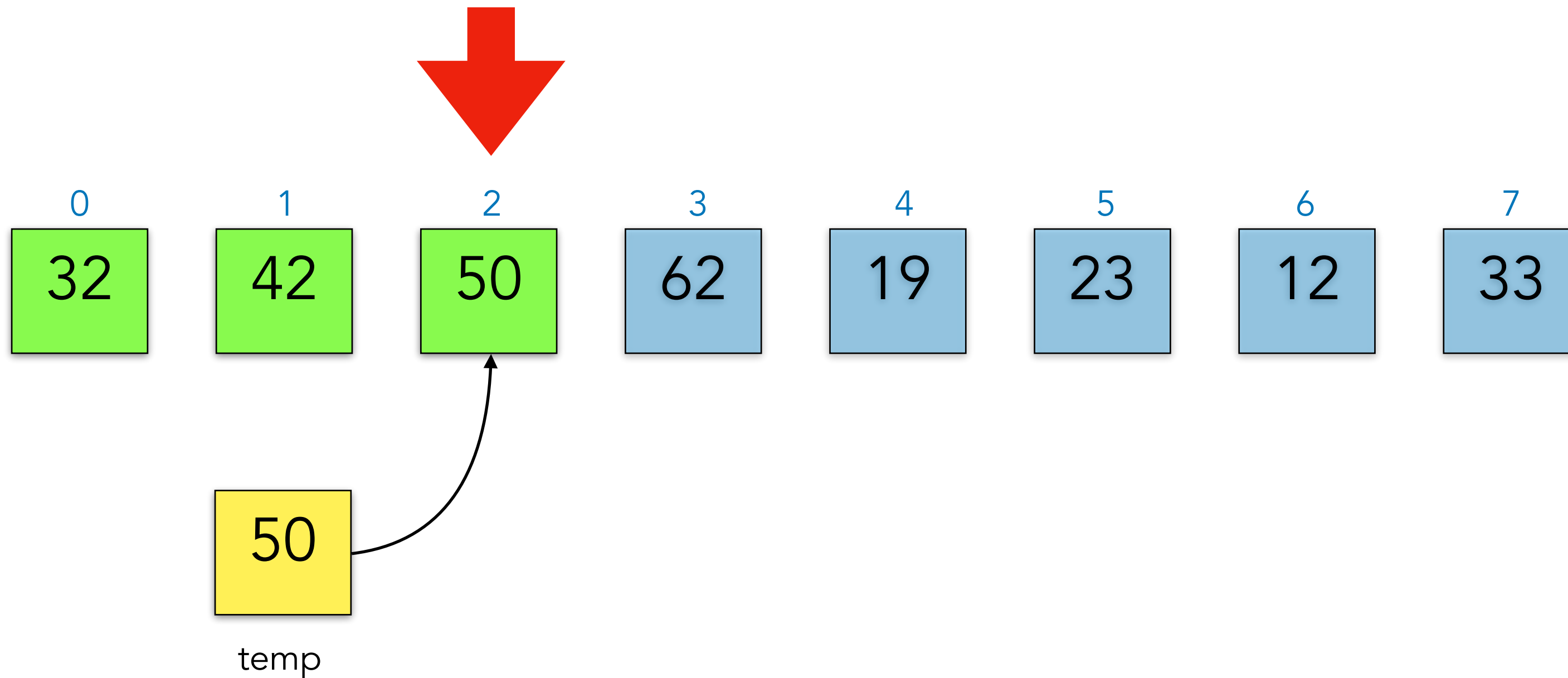
5.4 Insertionsort



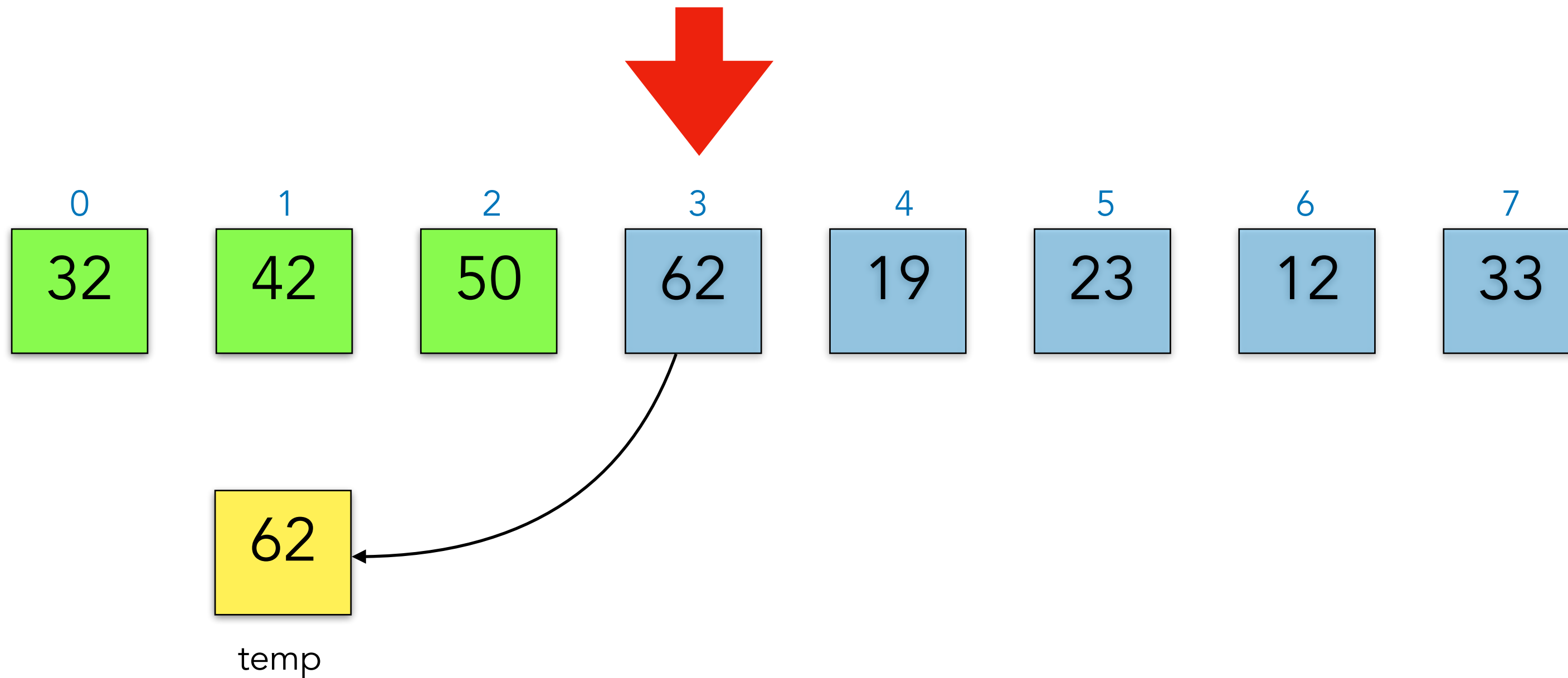
5.4 Insertionsort



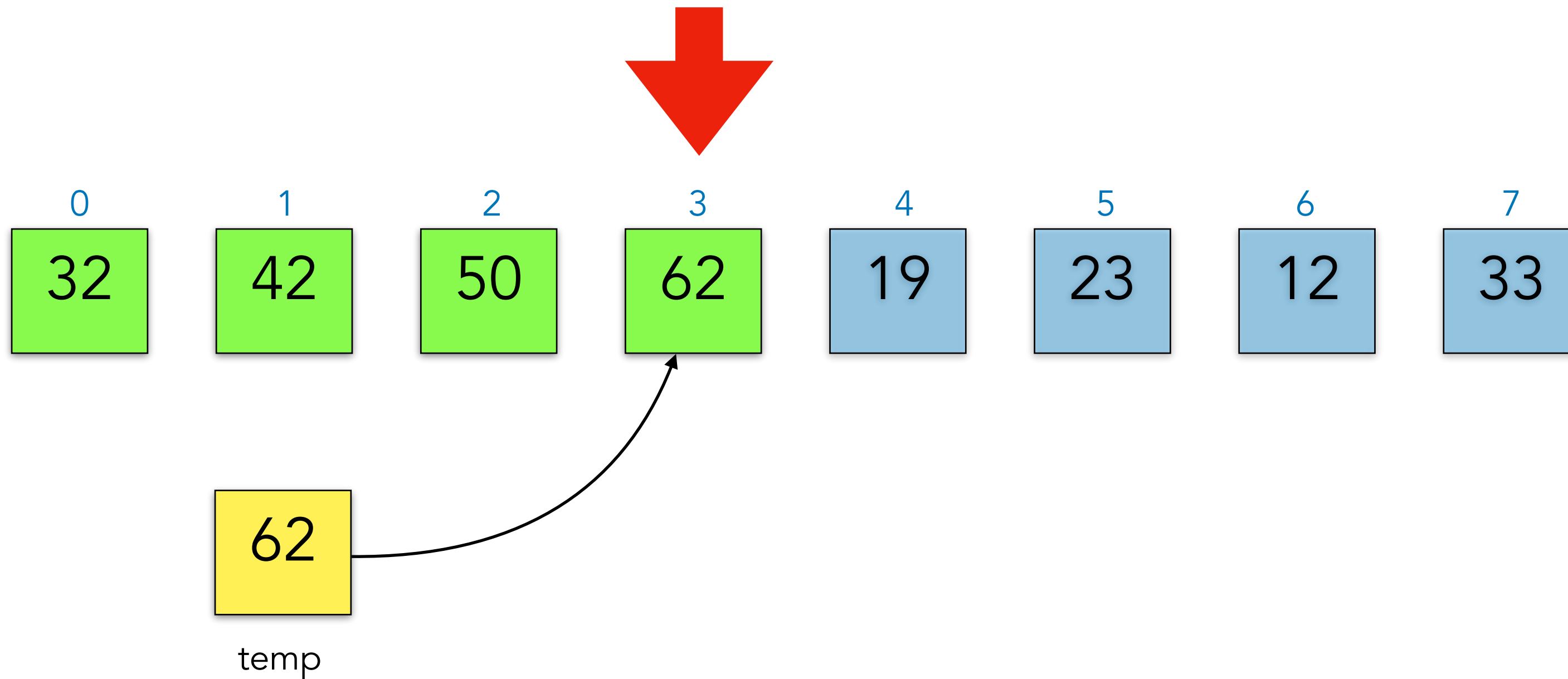
5.4 Insertionsort



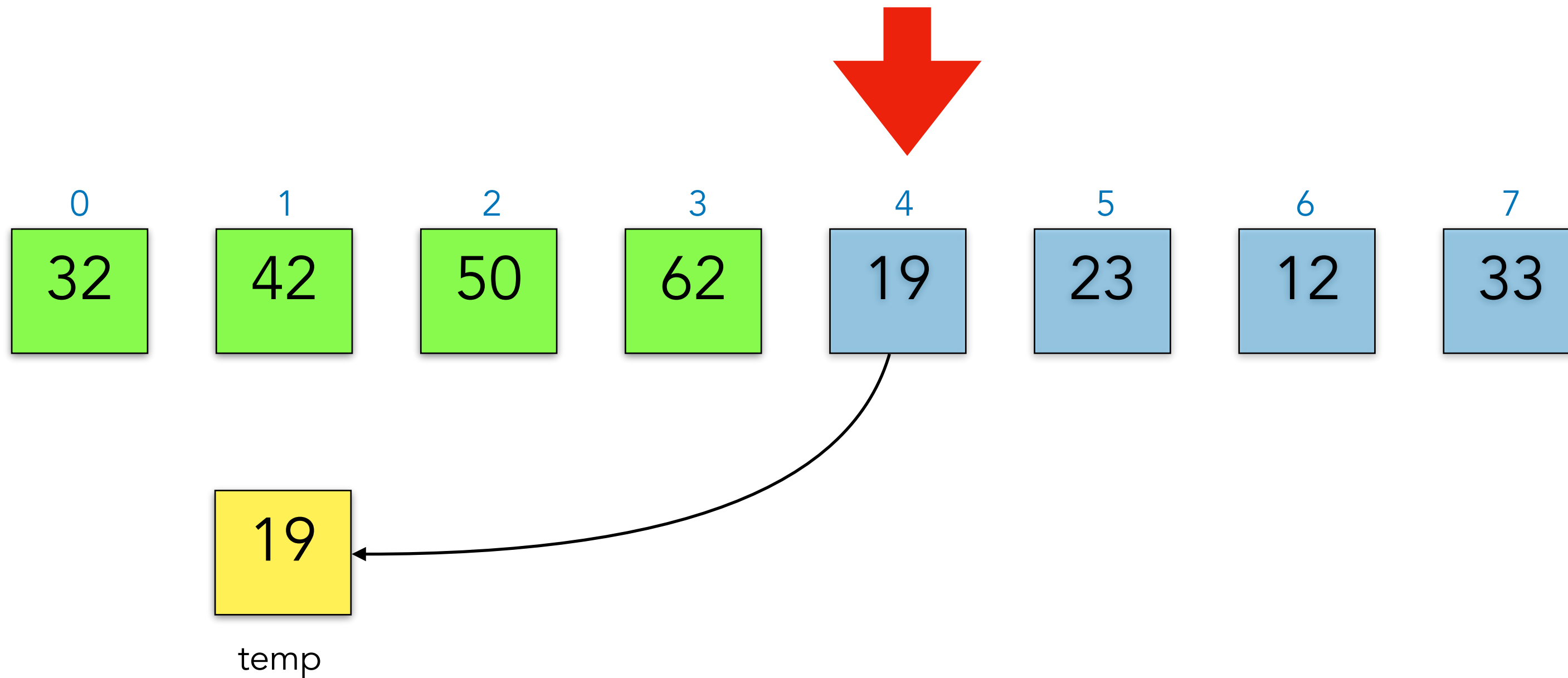
5.4 Insertionsort



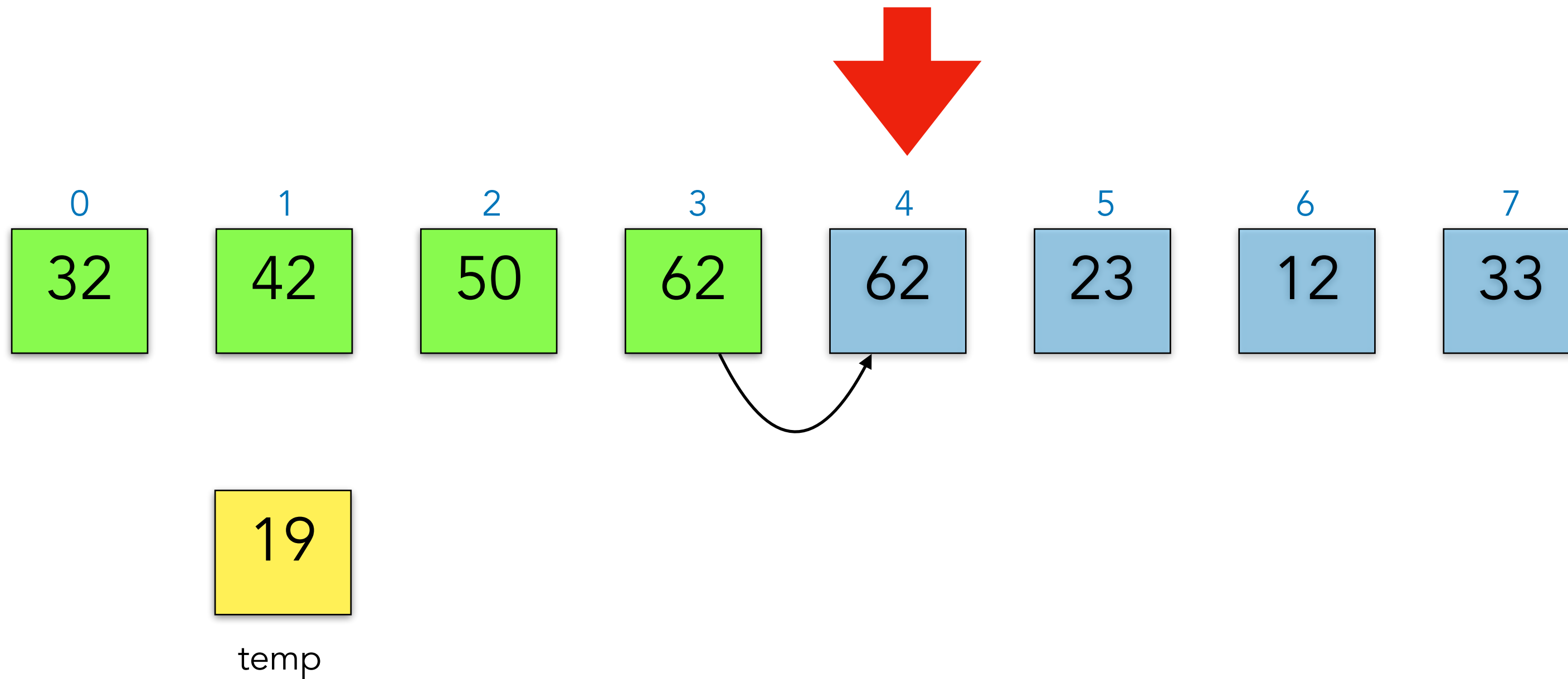
5.4 Insertionsort



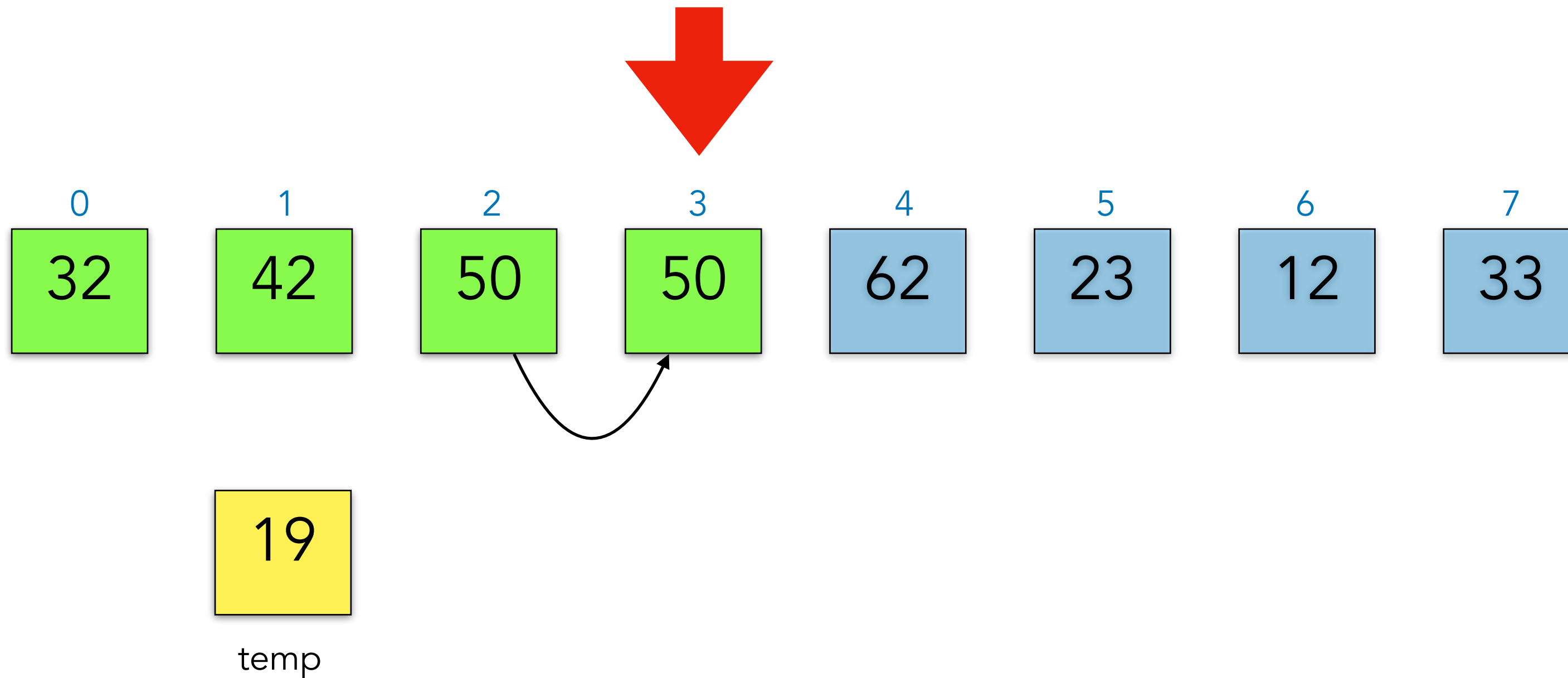
5.4 Insertionsort



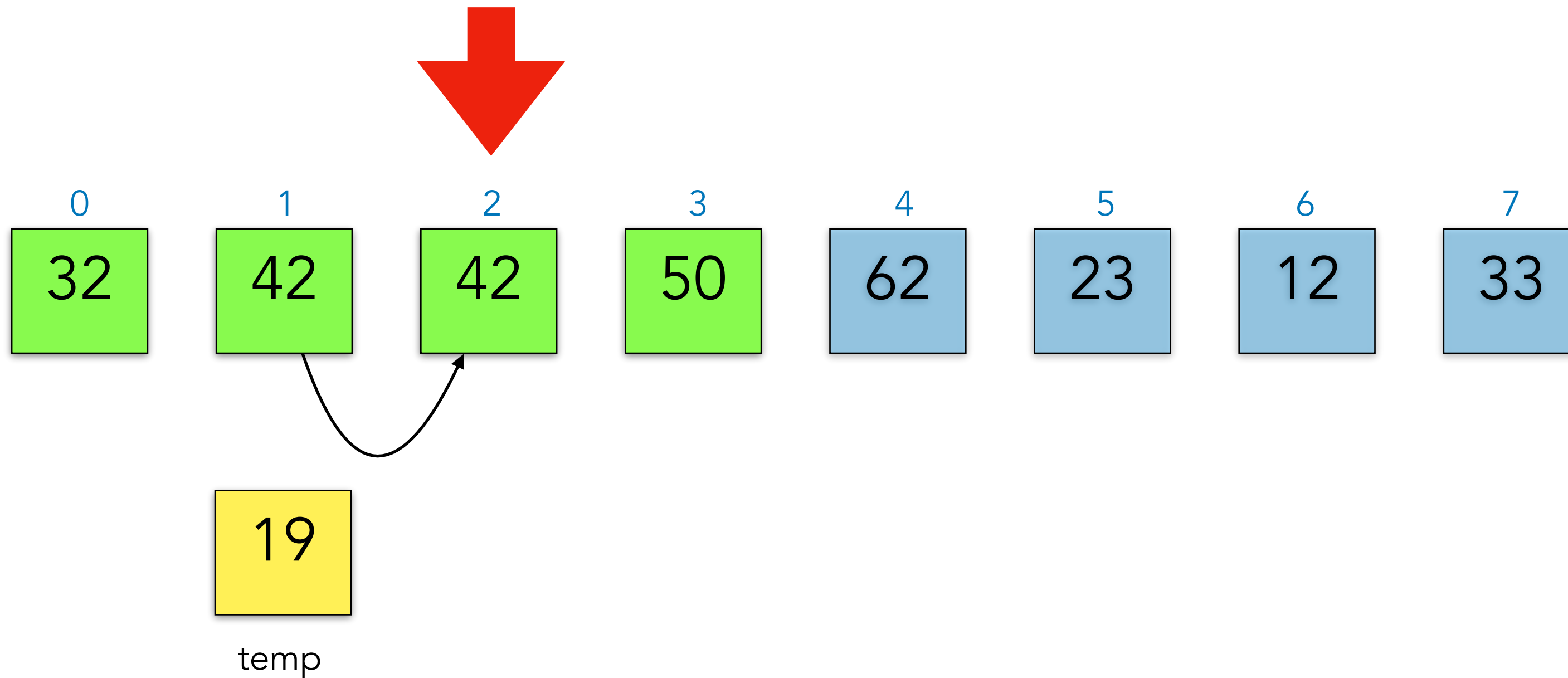
5.4 Insertionsort



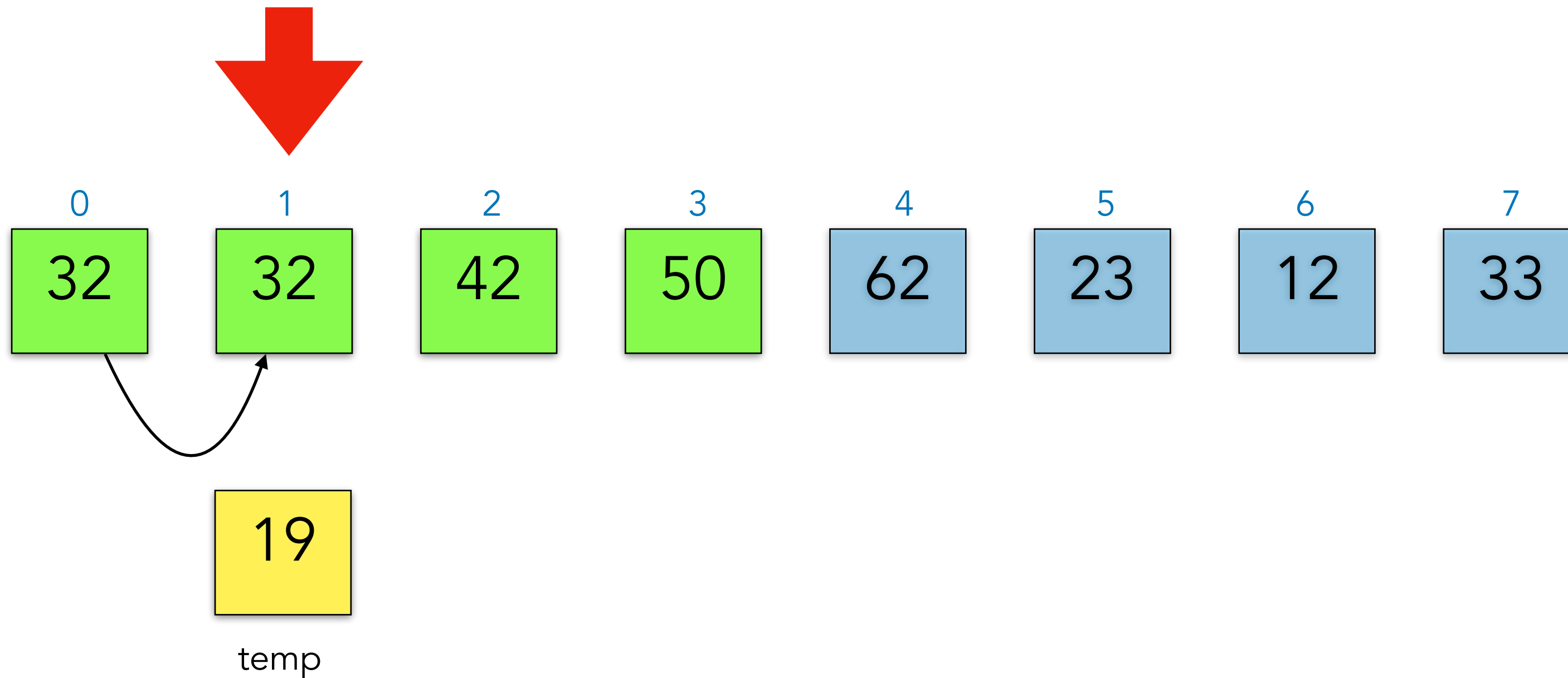
5.4 Insertionsort



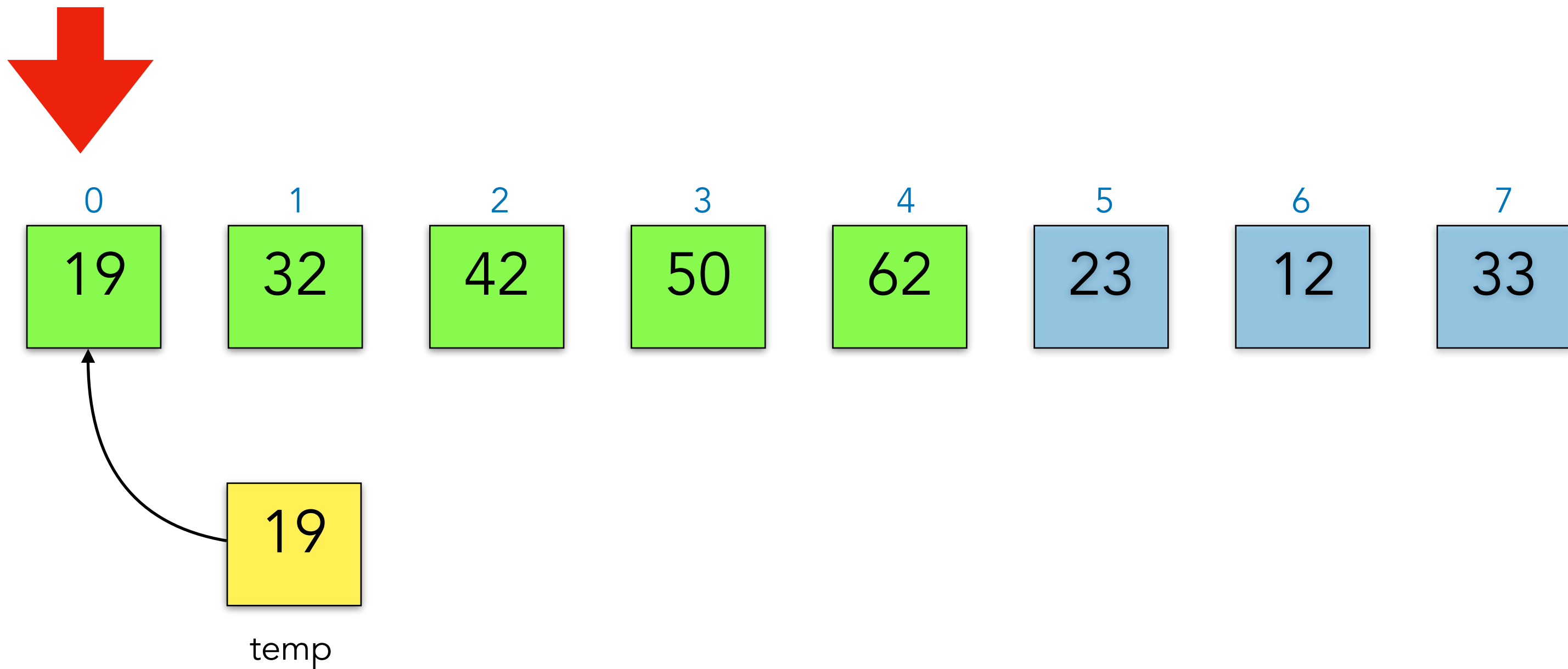
5.4 Insertionsort



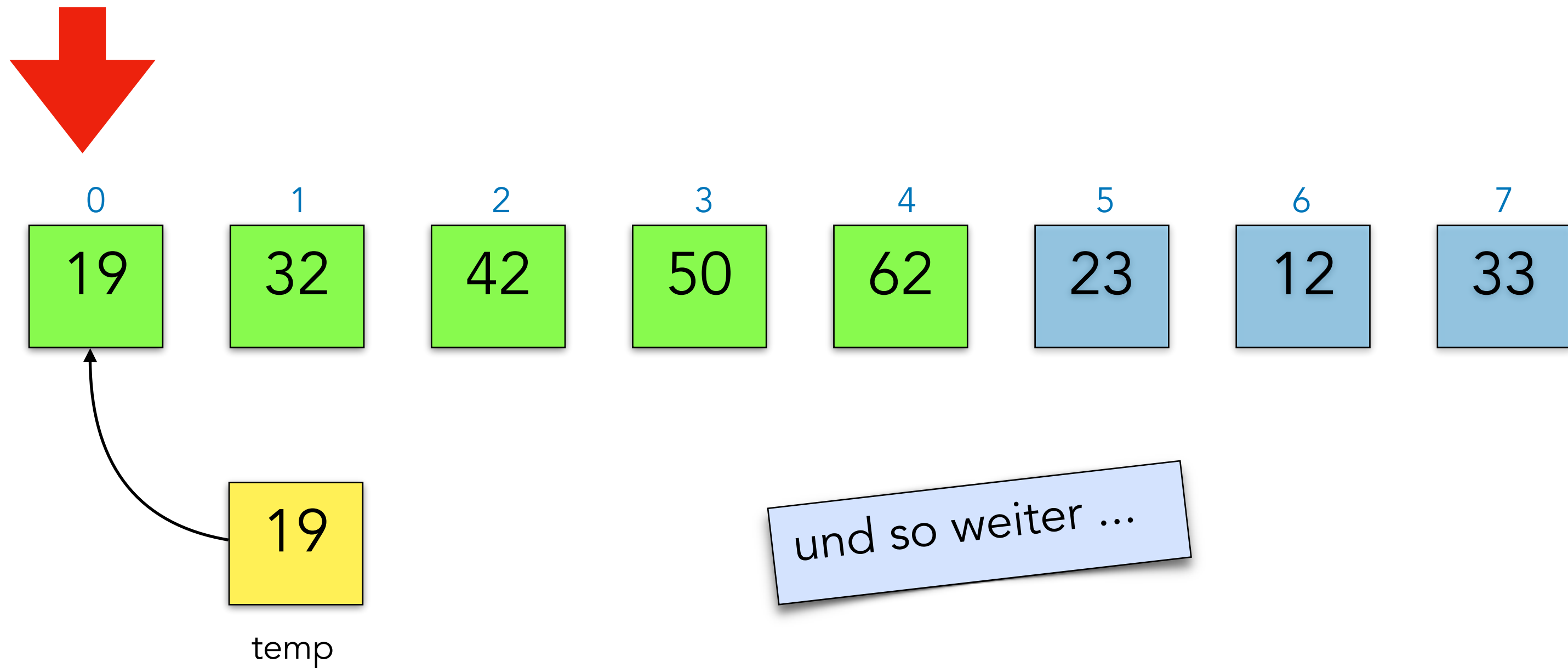
5.4 Insertionsort



5.4 Insertionsort



5.4 Insertionsort



5. Sortieren und Suchen

5.4 Der Insertionsort

Der Quelltext

5.4 Insertionsort

```
20 public void insertionsort()
21 {
22     int j, temp;
23
24     for (int i=1; i < MAX; i++)
25     {
26         j = i;
27         temp = zahl[i];
28
29         while ((j > 0) && (zahl[j-1]>temp))
30         {
31             zahl[j] = zahl[j-1];
32             j--;
33         }
34         zahl[j] = temp;
35     }
36 }
```

5.4 Insertionsort

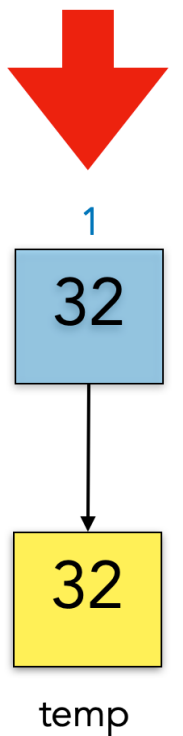
```
20 public void insertionsort()
21 {
22     int j, temp;
23
24     for (int i=1; i < MAX; i++)
25     {
26         j = i;
27         temp = zahl[i];
28
29         while ((j > 0) && (zahl[j-1]>temp))
30         {
31             zahl[j] = zahl[j-1];
32             j--;
33         }
34         zahl[j] = temp;
35     }
36 }
```

für alle Zahlen des Arrays - außer der ersten - mache der Reihe nach ...

5.4 Insertionsort

```
20 public void insertionsort()
21 {
22     int j, temp;
23
24     for (int i=1; i < MAX; i++)
25     {
26         j = i;
27         temp = zahl[i];
28
29         while ((j > 0) && (zahl[j-1]>temp))
30         {
31             zahl[j] = zahl[j-1];
32             j--;
33         }
34         zahl[j] = temp;
35     }
36 }
```

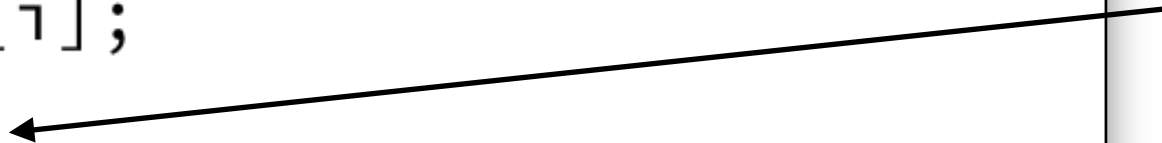
Kopiere die nächste Zahl,
die einsortiert werden soll.



5.4 Insertionsort

```
20 public void insertionsort()
21 {
22     int j, temp;
23
24     for (int i=1; i < MAX; i++)
25     {
26         j = i;
27         temp = zahl[i];
28
29         while ((j > 0) && (zahl[j-1]>temp))
30         {
31             zahl[j] = zahl[j-1];
32             j--;
33         }
34         zahl[j] = temp;
35     }
36 }
```

Solange der Anfang des
Arrays noch nicht erreicht...



5.4 Insertionsort

```
20 public void insertionsort()
```

```
21 {
```

```
22     int j, temp;
```

```
23  
24     for (int i=1; i < MAX; i++)
```

```
25     {
```

```
26         j = i;
```

```
27         temp = zahl[i];
```

```
28  
29         while ((j > 0) && (zahl[j-1]>temp))
```

```
30         {
```

```
31             zahl[j] = zahl[j-1];
```

```
32             j--;
```

```
33         }
```

```
34         zahl[j] = temp;
```

```
35     }
```

```
36 }
```

Solange der Anfang des
Arrays noch nicht erreicht...

und solange der Vorgänger
kleiner ist...

5.4 Insertionsort

```
20 public void insertionsort()
21 {
22     int j, temp;
23
24     for (int i=1; i < MAX; i++)
25     {
26         j = i;
27         temp = zahl[i];
28
29         while ((j > 0) && (zahl[j-1]>temp))
30         {
31             zahl[j] = zahl[j-1];
32             j--;
33         }
34         zahl[j] = temp;
35     }
36 }
```

Kopiere Vorgänger in
aktuelle Zahl



5.4 Insertionsort

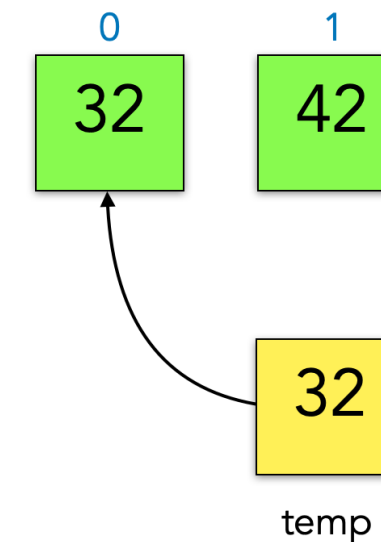
```
20 public void insertionsort()
21 {
22     int j, temp;
23
24     for (int i=1; i < MAX; i++)
25     {
26         j = i;
27         temp = zahl[i];
28
29         while ((j > 0) && (zahl[j-1]>temp))
30         {
31             zahl[j] = zahl[j-1];
32             j--;
33         }
34         zahl[j] = temp;
35     }
36 }
```

Kopiere Vorgänger in
aktuelle Zahl

und mache mit dem
Vorgänger weiter.

5.4 Insertionsort

```
20 public void insertionsort()
21 {
22     int j, temp;
23
24     for (int i=1; i < MAX; i++)
25     {
26         j = i;
27         temp = zahl[i];
28
29         while ((j > 0) && (zahl[j-1]>temp))
30         {
31             zahl[j] = zahl[j-1];
32             j--;
33         }
34         zahl[j] = temp;
35     }
36 }
```



Einfügestelle gefunden
Zahl mit Wert von temp
überschreiben.

5.4 Insertionsort

```
41 public void insertionsortBetter()
42 {
43     for (int i = 1; i < MAX; i++)
44     {
45         if (zahl[i - 1] <= zahl[i])
46             continue;
47
48         int temp = zahl[i];
49         int j = i;
50
51         while (j > 0 && zahl[j - 1] > temp)
52         {
53             zahl[j] = zahl[j - 1];
54             j--;
55         }
56         zahl[j] = temp;
57     }
58 }
```

Was ist an dieser Version besser?

5.4 Insertionsort

```
41 public void insertionsortBetter()
42 {
43     for (int i = 1; i < MAX; i++)
44     {
45         if (zahl[i - 1] <= zahl[i])
46             continue;
47
48         int temp = zahl[i];
49         int j = i;
50
51         while (j > 0 && zahl[j - 1] > temp)
52         {
53             zahl[j] = zahl[j - 1];
54             j--;
55         }
56         zahl[j] = temp;
57     }
58 }
```

Wenn die Zahl bereits an der richtigen Position steht, wird der nächste Schleifendurchgang gestartet mit **continue**.

5. Sortieren und Suchen

5.4 Der Insertionsort

Vergleich mit Bubblesort und Selectionsort

5.4 Insertionsort

Zahlenfolge	Vergleiche	Tausch- vorgänge	Summe
1, 2, 3, 4, 5	10	0	10
7, 2, 9, 1, 5	10	6	28
5, 4, 3, 2, 1	10	10	40

Bubblesort

Zahlenfolge	Vergleiche	Tausch- vorgänge	Summe
1, 2, 3, 4, 5	4	0	10
7, 2, 9, 1, 5	8	5	13
5, 4, 3, 2, 1	10	10	20

Insertionsort

Zahlenfolge	Vergleiche	Tausch- vorgänge	Summe
1, 2, 3, 4, 5	10	0	10
7, 2, 9, 1, 5	10	2	16
5, 4, 3, 2, 1	10	4	22

Selectionsort

Zahlenfolge	Vergleiche	Tausch- vorgänge	Summe
1, 2, 3, 4, 5	10	15	55
7, 2, 9, 1, 5	8	9	35
5, 4, 3, 2, 1	10	8	34

Quicksort

5.4 Insertionsort

Sortierverfahren	Vergleiche	Tauschvorgänge*	Gesamtoperationen
Bubblesort	49 995 000	24 997 500	124 987 500
Selectionsort	49 995 000	9 999	50 024 997
Insertionsort	25 000 000	25 000 000	50 000 000
Quicksort (A)	184 200	92 100	460 500
* Bei Insertionsort bedeutet „Tauschvorgänge“ Verschiebungen , nicht echte Dreier-Tausche.			

Berechnung von ChatGPT für einen völlig ungeordneten Array aus 10.000 int-Zahlen