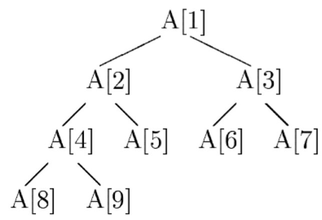
Kapitel_5_Effiziente Sortieralgorithmen

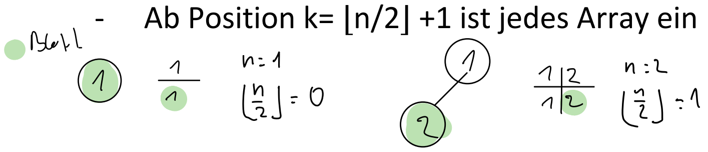
Heap-Sort

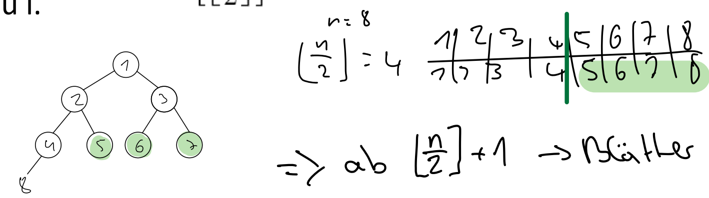


- ein Knoten ist immer größer als seine Kinder *(Reihenfolge egal)*
- Ein Array erfüllt die sogenannte Heap-Eigenschaft, falls:

$$A\left[\left\lfloor \frac{i}{2} \right\rfloor\right] \geq A[i] \;\text{ für } 2 \leq i \leq n$$

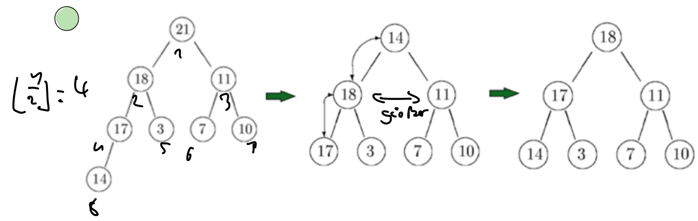
- Array-Positionen 2^i bis 2^{i+1}-1 gehören zum Niveau i.
- Ab Position k= ⌊n/2⌋ +1 ist jedes Array ein Heap.





Algorithmus:

- Nehme das oberste Element des Baumes weg,
- setzte das letzte Element nach oben
- Generiere aus dem Rest einen Heap
- Nehme wieder das oberste Element weg



Vorteile

- verbrauch keinen zusätzlichen Speicher

Nachteil: Heap Bedingung

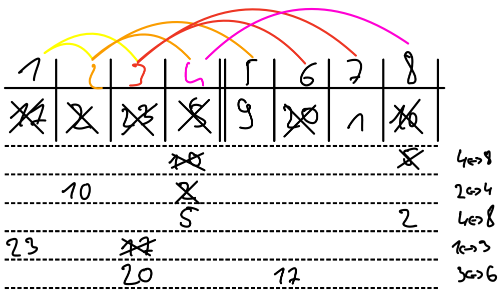
Laufzeit-Analyse

- Initialer Aufbau: O(n) | | (n-1) mal Auswählen und Versickern: (n-1) * O(log n)
- Gesamt: O(n)+O(n log n)= **O(n log n)**

Prinzip „Versickern":

- Vertausche falls nötig A[k] mit dem größeren der beiden Söhne A[2k] und A[2k+1].
- Wiederhole bis Rest-Array durchlaufen oder keine Vertauschung mehr nötig ist.

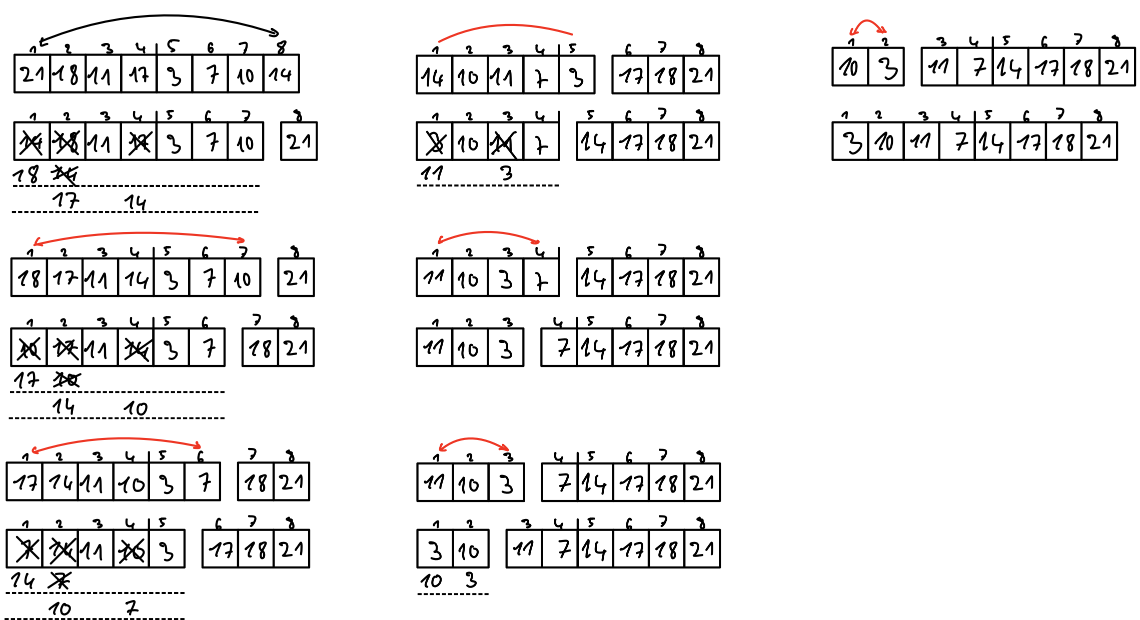
Beispiel:



Herstellen der initialen Heap-Bedingung

- Array ab k=⌊n/2⌋+1 automatisch ein Heap.
- Versickere rückwärts ab A[⌊n/2⌋] bis A[1] alle Elemente
- Danach heap-Bedingung hergestellt

Beispiel:



Hochschule Hannover
Fakultät IV - Abteilung Informatik
Hovestadt

Hannover, den 31. Mai 2021

Übungen zur Vorlesung Algorithmen und Datenstrukturen

(SS 2021, Aufgabenblatt 10)

Aufgabe 37 Heap (2 Punkte)

Erfüllen die folgenden Arrays die Heap-Eigenschaft? Bitte begründen Sie Ihre Antwort.

Bemerkung: Wie in der Vorlesung ist hier ein Max-Heap gemeint, also ein Heap bzgl. der Relation “>“ (größer). Verletzt ?

a) 15 ✓

b) 75, 29, 74, 10, 19, 9, 70, 3, 5, 6, 18, 8, 7, 68, 69, 4

Aufgabe 38 Heap-Eigenschaft (2 Punkte)

Beweisen oder **widerlegen** Sie die folgenden Behauptungen:

Bemerkung: Wie in der Vorlesung ist hier ein Max-Heap gemeint, also ein Heap bzgl. der Relation “>“ (größer).

a) Jedes absteigend sortierte Array ist ein Max-Heap. Ja

b) Die Darstellung eines Max-Heaps in einem Array ist absteigend sortiert. Nein

Aufgabe 39 Heapsort (2 Punkte)

Gegeben sei folgende Zahlenreihe: $10, 30, 60, 90, 20, 50, 80, 40, 70$

a) Führen Sie den ersten Schritt des Heapsort-Algorithmus aus. Erzeugen Sie also aus der genannten Zahlenreihe einen Max-Heap, also ein Heap bzgl. der Relation “>“ (größer).

b) Führen Sie nun den zweiten Schritt des Heapsort-Algorithmus aus. Überführen Sie also den gerade erzeugten Heap in ein aufsteigend sortiertes Array. Zeigen Sie hierbei bitte das Array nach jedem Schleifendurchlauf.

~~10~~ ~~30~~ ~~60~~ ~~90~~ | 20 50 ~~80~~ 40 ~~70~~
90 ~~90~~ 80 ~~30~~ 60 10 30
~~10~~ ~~70~~
70 ~~40~~
40

90 70 80 40 | 20 50 60 10 30

~~90~~ 70 ~~80~~ 40 | 20 50 ~~60~~ 10 || 90
~~30~~ ~~70~~ 30
80 60

~~80~~ ~~70~~ 60 ~~40~~ | 20 50 30 || 90 80
~~10~~ ~~40~~ 70
70 40

~~70~~ 40 ~~60~~ 10 | 20 ~~50~~ || 90 80 70
~~30~~ ~~30~~ 30
60 50

~~60~~ 40 ~~50~~ 10 | 20 || 90 80 70 60
~~30~~ 30
50

~~50~~ ~~40~~ 30 10 | || 90 80 70 60 50
~~20~~ 20
40

~~40~~ 20 ~~30~~ || 90 80 70 60 50 40
~~10~~ 10
30

~~30~~ ~~20~~ || 90 80 70 60 50 40 30
~~10~~ 10
20

10 || 90 80 70 60 50 40 30 20

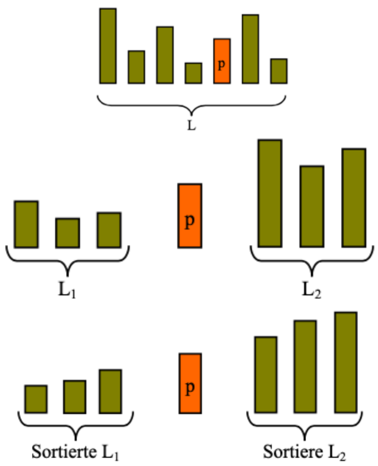
90 80 70 60 50 40 30 20 10

QuickSort

QuickSort ist das im Mittel schnellste Sortierverfahren

- C_{max}: $O(n^2)$,
- $C_{avg/best}$: $O(n \log n)$

Vorteil: geht auch für verkettete Listen



Ausgang:

- Gegeben sei unsortierte Liste L
- Gewählt wird ein beliebiges Element **p als Pivot-Element**

Divide:

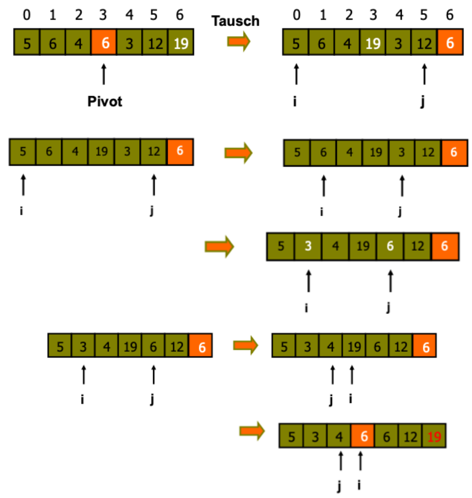
- Aufteilung von L in L_1 *(kleiner als p)* und L_2 *(größer als p)*
- Anwendung der Aufteilung rekursiv für L_1 und L_2, bis Problem trivial

Conquer:

- Mergen von L_1, p und L_2 zu sortiertem L

Strategie zur Wahl des Pivot-Elements

- **letzte Element**
  - Schlecht bei vorsortierten Listen, dann ist Pivot das größte Element
- **3-Median Strategie**
  - Wähle aus unsortierter Folge 3 beliebige Elemente, z.B. links, mitte, rechts.
  - Bestimme den mittleren der drei Schlüsselwerte, wähle diesen als Pivot-Element.
- **Zufalls-Strategie**



Partitionierung:

1. Vertausche das **Pivot-Element k** gegen das Letzte Element aus
   Setzte Zeiger auf A[left] <- i und A[right] <- j
2. Solange i<j:
   Bewege i nach rechts, solange Elemente kleiner als Pivot-Element sind
   Bewege j nach links, solange Elemente größer als Pivot-Element sind
   a. Tausche die Elemente wenn L[i] >= k und L[j] <= k
   b. falls sich i und j überschneiden tausche i gegen das k

Aufgabe 40 **Quicksort** **(2 Punkte)**

Gegeben sei folgende Zahlenreihe: 10, 30, 60, 90, 20, 50, 80, 40, 70

Sortieren Sie die Zahlenreihe mithilfe von QuickSort und wählen Sie das Pivot-Element wie angegeben. Zeigen Sie das Array nach jedem Aufruf von *Partitionieren* unter Angabe der jeweiligen Position des Pivot-Elements.

a) Wählen Sie das letzte Element als Pivot-Element.

b) Verfolgen Sie die 3-Median Strategie zur Wahl des Pivot-Elements.

a.)

10	30	60	90	20	50	80	40	70

10	30	60	40	20	50	80	90	70

10	30	60	40	20	50	70	90	80

b.)

10	30	60	90	20	50	80	40	70

same

überschnitten i gegen k

Aufgabe 3

Gegeben sei folgendes Array:

- **1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29**

Führen Sie auf diesen Array den Partitionierungs-algorithmus von QuickSort aus. Verwenden Sie hierbei das mittlere Element als Pivot-Element. Zeigen Sie das resultierende Array sowie die Position des Pivot-Elements.

1	3	5	7	9	11	13	15	17	19	21	23	25	27	29

1	3	5	7	9	11	13	29	17	19	21	23	25	27	15

überschnitten i mit letzten k auslassen

1	3	5	7	9	11	13	15	17	19	21	23	25	27	29

Aufgabe 4

Gegeben sei folgendes Array:

- **5, 15, 21, 1, 27, 11, 23, 3, 29, 17, 25, 7, 13, 9, 19**

Führen Sie auf diesen Array den Partitionierungs-algorithmus von QuickSort aus. Verwenden Sie hierbei das mittlere Element als Pivot-Element. Zeigen Sie das resultierende Array sowie die Position des Pivot-Elements.

5	15	21	1	27	11	23	3	29	17	25	7	13	9	19

5	15	21	1	27	11	23	19	29	17	25	7	13	9	3

1	15	21	5	27	11	23	19	29	17	25	7	13	9	3

i mit k

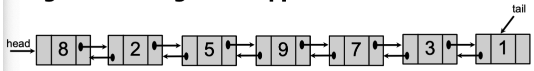
1	3	21	5	27	11	23	19	29	17	25	7	13	9	15

Anwendung auf doppelt verketteten Listen

```
Algorithm Partitioniere(Node left, Node right, Node piv)
   i ←left;
   j ←right.prev;
   k ←piv.data;
   swap(piv,right); // temp. Speicherung von Pivot-Element am Ende
   while i!=j
      while (i.data <= k && i != j)
         i ←i.next;
      while (j.data >= k && i != j)
         j ←j.prev;
      if(i!=j)
         swap(i,j);
   if(i.data>k)
      swap(i,right); // Bewege Pivot-Element an finale Position
   return i
```

Aufgabe 5

Gegeben sei folgende doppelt-verkettete Liste:



Führen Sie auf dieser Liste den Partitionierungsalgorithmus von QuickSort aus. *Left* zeige hierbei auf das Element 8, *right* zeige auf das Element 1 und *piv* zeige auf das Element 9. Zeigen Sie das resultierende Array sowie die Position des Pivot-Elements.

Quicksort auf einfach verketteten Listen

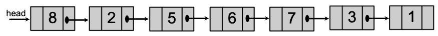
Problem: man kann nicht von rechts nach links laufen

- Offensichtlich wird nur vorwärts über die next-Struktur gelaufen
- Zugriff auf das letzte Element per tail-Zeiger möglich

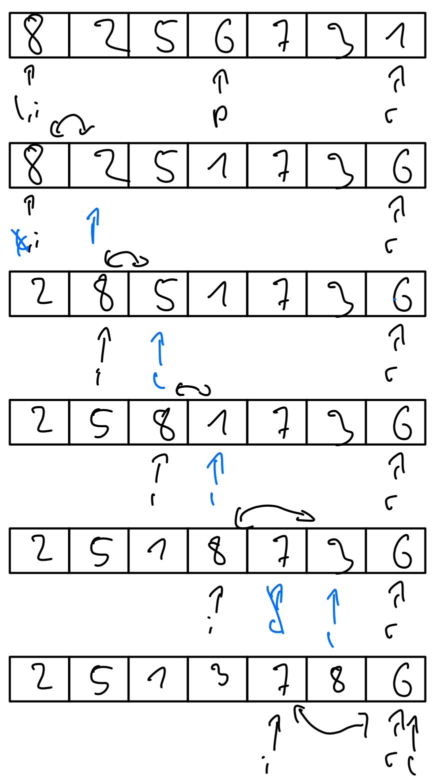
```
Algorithm Partitioniere(Node left, Node right, Node piv)
   k ←piv.data;
   swap(piv,right); // temp Speicherung von Pivot-Element am Ende
   Node index = left;
   while left!=right
      if (left.data < k)
         swap(left,index);
         index = index.next;
      left=left.next;
   swap(index,right); // Bewege Pivot-Element an finale Position
   return index
```

Aufgabe 6

Gegeben sei folgende einfach-verkettete Liste:



Führen Sie auf dieser Liste den Partitionierungsalgorithmus von QuickSort aus. *Left* zeige hierbei auf das Element 8, *right* zeige auf das Element 1 und *piv* zeige auf das Element 6. Zeigen Sie das resultierende Array sowie die Position des Pivot-Elements.



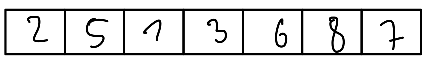
da l > p ist gehen wir weiter

da nun l < p werden i und l vertauscht

" und tauschen

"

?



da left = right => swap(i, r)