TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Software Entwicklung 1

Annette Bieniusa

AG Softech
FB Informatik
TU Kaiserslautern

Klassische Sortieralgorithmen

Motivation

- Suchen und Sortieren sind Teilprobleme, die in einer Vielzahl von Programmen auftreten
- Beispiel: Musiksammlungen, Profile in sozialen Netzwerken, Berechnung von Median, Erstellen von Histogrammen
- Große Datenmengen erfordern effiziente Algorithmen (Laufzeit und Speicherbedarf)

Sortieren ist eine Standardaufgabe, die Teil vieler spezieller oder auch umfassenderer Aufgabenstellungen ist.

Untersuchungen zeigen, dass „mehr als ein Viertel der kommerziell verbrauchten Rechenzeit auf Sortiervorgänge entfällt“
(Ottmann, Widmayer: Algorithmen und Datenstrukturen, Kap. 2).

Begriffsklärung: Sortierproblem

Gegeben:

- Liste von Elementen $S = [x_0, x_1, \ldots, x_{n-1}]$
- Totale Ordnung ($\leq$) auf Elementen

Gesucht:

- Liste $\pi = [\pi_0, \pi_1, \ldots, \pi_{n-1}]$, so dass:
- π ist **sortiert**:

$$\pi_0 \leq \pi_1 \leq \cdots \leq \pi_{n-1}$$

- π ist **Permutation** von S:
  π enthält die gleichen Elemente, aber eventuell in anderer Reihenfolge

Hinweis: In der Mathematik bezeichnet man eine bijektive Selbstabbildung auf einer Menge mit n Elementen als Permutation.

Algorithmen

Wir behandeln in SE1 die folgenden klassischen Sortieralgorithmen:

- Insertion Sort ($\rightarrow$ Vorlesung)
- Selection Sort ($\rightarrow$ Vorlesung)
- Mergesort ($\rightarrow$ Übungsblatt)
- Quicksort ($\rightarrow$ Vorlesung)

Diese Algorithmen basieren alle auf dem **Vergleichen von Elementen** und benötigen daher keine weiteren Annahmen als die zuvor genannten.

Insertionsort / Sortieren durch Einfügen

Algorithmische Grundidee:

- Füge die noch nicht sortierten Elemente nacheinander in die bereits sortierte Teilliste an der richtigen Stelle ein.
- Beginne mit der leeren Liste (welche bereits sortiert ist)

Beispiel: Sortiere die Liste $[17, 12, 6, 19, 23, 8, 5, 10]$

Insertionsort Beispiel

Beispiel: Sortiere die Liste $[17, 12, 6, 19, 23, 8, 5, 10]$

Noch nicht sortiert	Bereits sortiert
$[17, 12, 6, 19, 23, 8, 5, 10]$	$[]$

Insertionsort Beispiel

Beispiel: Sortiere die Liste $[17, 12, 6, 19, 23, 8, 5, 10]$

Noch nicht sortiert	Bereits sortiert
$[17, 12, 6, 19, 23, 8, 5, 10]$	$[]$
$[12, 6, 19, 23, 8, 5, 10]$	$[17]$

Insertionsort Beispiel

Beispiel: Sortiere die Liste $[17, 12, 6, 19, 23, 8, 5, 10]$

Noch nicht sortiert	Bereits sortiert
$[17, 12, 6, 19, 23, 8, 5, 10]$	$[]$
$[12, 6, 19, 23, 8, 5, 10]$	$[17]$
$[6, 19, 23, 8, 5, 10]$	$[12, 17]$

Insertionsort Beispiel

Beispiel: Sortiere die Liste $[17, 12, 6, 19, 23, 8, 5, 10]$

Noch nicht sortiert	Bereits sortiert
$[17, 12, 6, 19, 23, 8, 5, 10]$	$[]$
$[12, 6, 19, 23, 8, 5, 10]$	$[17]$
$[6, 19, 23, 8, 5, 10]$	$[12, 17]$
$[19, 23, 8, 5, 10]$	$[6, 12, 17]$

Insertionsort Beispiel

Beispiel: Sortiere die Liste $[17, 12, 6, 19, 23, 8, 5, 10]$

Noch nicht sortiert	Bereits sortiert
$[17, 12, 6, 19, 23, 8, 5, 10]$	$[]$
$[12, 6, 19, 23, 8, 5, 10]$	$[17]$
$[6, 19, 23, 8, 5, 10]$	$[12, 17]$
$[19, 23, 8, 5, 10]$	$[6, 12, 17]$
$[23, 8, 5, 10]$	$[6, 12, 17, 19]$

Insertionsort Beispiel

Beispiel: Sortiere die Liste $[17, 12, 6, 19, 23, 8, 5, 10]$

Noch nicht sortiert	Bereits sortiert
$[17, 12, 6, 19, 23, 8, 5, 10]$	$[]$
$[12, 6, 19, 23, 8, 5, 10]$	$[17]$
$[6, 19, 23, 8, 5, 10]$	$[12, 17]$
$[19, 23, 8, 5, 10]$	$[6, 12, 17]$
$[23, 8, 5, 10]$	$[6, 12, 17, 19]$
$[8, 5, 10]$	$[6, 12, 17, 19, 23]$

Insertionsort Beispiel

Beispiel: Sortiere die Liste $[17, 12, 6, 19, 23, 8, 5, 10]$

Noch nicht sortiert	Bereits sortiert
$[17, 12, 6, 19, 23, 8, 5, 10]$	$[]$
$[12, 6, 19, 23, 8, 5, 10]$	$[17]$
$[6, 19, 23, 8, 5, 10]$	$[12, 17]$
$[19, 23, 8, 5, 10]$	$[6, 12, 17]$
$[23, 8, 5, 10]$	$[6, 12, 17, 19]$
$[8, 5, 10]$	$[6, 12, 17, 19, 23]$
$[5, 10]$	$[6, 8, 12, 17, 19, 23]$

Insertionsort Beispiel

Beispiel: Sortiere die Liste $[17, 12, 6, 19, 23, 8, 5, 10]$

Noch nicht sortiert	Bereits sortiert
$[17, 12, 6, 19, 23, 8, 5, 10]$	$[]$
$[12, 6, 19, 23, 8, 5, 10]$	$[17]$
$[6, 19, 23, 8, 5, 10]$	$[12, 17]$
$[19, 23, 8, 5, 10]$	$[6, 12, 17]$
$[23, 8, 5, 10]$	$[6, 12, 17, 19]$
$[8, 5, 10]$	$[6, 12, 17, 19, 23]$
$[5, 10]$	$[6, 8, 12, 17, 19, 23]$
$[10]$	$[5, 6, 8, 12, 17, 19, 23]$

Insertionsort Beispiel

Beispiel: Sortiere die Liste $[17, 12, 6, 19, 23, 8, 5, 10]$

Noch nicht sortiert	Bereits sortiert
$[17, 12, 6, 19, 23, 8, 5, 10]$	$[]$
$[12, 6, 19, 23, 8, 5, 10]$	$[17]$
$[6, 19, 23, 8, 5, 10]$	$[12, 17]$
$[19, 23, 8, 5, 10]$	$[6, 12, 17]$
$[23, 8, 5, 10]$	$[6, 12, 17, 19]$
$[8, 5, 10]$	$[6, 12, 17, 19, 23]$
$[5, 10]$	$[6, 8, 12, 17, 19, 23]$
$[10]$	$[5, 6, 8, 12, 17, 19, 23]$
$[]$	$[5, 6, 8, 10, 12, 17, 19, 23]$

Insertionsort auf verketteten Listen

Sortiertes Einfügen:

```
// Fuegt Knoten rekursiv an die richtige Stelle ein;
// liefert Pointer auf modifizierte Ergebnisliste
node_t *insert_sorted_rec(node_t *first, node_t *n)
{
  // Basisfall: Einfuegestelle ist gefunden
  if (first == NULL || n->value < first->value)
  {
    n->next = first;
    return n;
  }
  else
  {
    // Rekursives Einfuegen in Restliste
    first->next = insert_sorted_rec(first->next, n);
    return first;
  }
}
```

Insertionsort auf verketteten Listen

Sortieren:

```
// Rekursives Sortieren durch Einfuegen
void insertion_sort_list_r(linked_list_t *list)
{
  // Pointer auf sortierte Ergebnisliste
  node_t *target = NULL;
  // n ist Pointer auf naechsten einzufuegenden Knoten
  node_t *n = list->first;
  while (n != NULL)
  {
    // Entnehme die Knoten immer vom Anfang der (Rest-)Liste
    node_t *next = n->next;
    // Fuege den Knoten an der richtigen Stelle der
    Ergebnisliste ein
    target = insert_sorted_rec(target, n);
    n = next;
  }
  // Setze den Pointer auf die sortierte Ergebnisliste
  list->first = target;
}
```

Insertionsort auf verketteten Listen (iterativ)

```
void insert_element_sorted(node_t **target_p, node_t *n)
{
  node_t **p = target_p;
  while (*p != NULL && (*p)->value < n->value)
  {
    // Iteriere an die Einfuegestelle
    p = &((*p)->next);
  }
  // Fuege den Knoten hier ein
  n->next = *p;
  *p = n;
}
```

Insertionsort auf verketteten Listen (iterativ)

```
void insertion_sort_list (linked_list_t *ll)
{
  // Pointer auf sortierte Ergebnisliste
  node_t *target = NULL;
  // Pointer auf Pointer auf naechsten einzufuegenden Knoten
  node_t **curr = &(ll->first);
  while (*curr) {
    // n ist Pointer auf naechsten einfuegenden Knoten
    node_t *n = *curr;
    *curr = (*curr)->next;
    // Fuege n in die Ergebnisliste ein
    insert_element_sorted(&target, n);
  }
  // Setze den Pointer auf die sortierte Ergebnisliste
  ll->first = target;
}
```

Selectionsort / Sortieren durch Auswahl

Algorithmische Grundidee:

- Entferne minimales Element `min` aus Liste
- Sortiere die übrige Liste
- Füge `min` als erstes Element vorne an die sortierte Teilliste an

Selectionsort Implementierung auf Array

1. Finde einen Index $imin$ des Arrays f, so dass $f[imin]$ ein minimales Element von $f[0]$ bis $f[N-1]$ enthält
2. Vertausche $f[imin]$ und $f[0]$
3. Sortiere dann den Bereich $f[1]$ bis $f[N-1]$ auf gleiche Art

```
void selectionsort(int *f, int n) {
  /* Iteriere ueber die Indizes des Arrays */
  for (int i = 0; i < n - 1; i++) {
    /* Finde Index fuer minimales Element */
    int imin = i;
    for (int j = i+1; j < n; j++) {
      if (f[j] < f[imin]) {
        imin = j;
      }
    }
    /* Vertausche Elemente */
    swap(&f[i], &f[imin]);
  }
}
```

Divide-and-Conquer-Strategie

Allgemeine Strategie zum Algorithmen-Entwurf:

- Zerlege das Problem in **kleinere Teilprobleme**.
- Wende den Algorithmus **rekursiv** auf die Teilprobleme an.
- Füge die Teilergebnisse wieder zusammen.

Divide-and-Conquer-Strategie

Allgemeine Strategie zum Algorithmen-Entwurf:

- Zerlege das Problem in **kleinere Teilprobleme**.
- Wende den Algorithmus **rekursiv** auf die Teilprobleme an.
- Füge die Teilergebnisse wieder zusammen.

Wir betrachten hier **Mergesort** und **Quicksort** als Beispiele dieser Strategie

Mergesort / Sortieren durch Mischen

Anwenden der Divide-and-Conquer-Strategie zum Entwurf von Mergesort

- Zerlege das Problem in kleinere Teilprobleme.
- Wende den Algorithmus rekursiv auf die Teilprobleme an.
- Füge die Teilergebnisse wieder zusammen.

Mergesort / Sortieren durch Mischen

Anwenden der Divide-and-Conquer-Strategie zum Entwurf von Mergesort

- Zerlege das Problem in kleinere Teilprobleme.
  - Zerlegen in zwei gleich große Teillisten
- Wende den Algorithmus rekursiv auf die Teilprobleme an.
- Füge die Teilergebnisse wieder zusammen.

Mergesort / Sortieren durch Mischen

Anwenden der Divide-and-Conquer-Strategie zum Entwurf von Mergesort

- Zerlege das Problem in kleinere Teilprobleme.
  - Zerlegen in zwei gleich große Teillisten
  - (bei ungerader Anzahl Längenunterschied von einem Element)
- Wende den Algorithmus rekursiv auf die Teilprobleme an.
- Füge die Teilergebnisse wieder zusammen.

Mergesort / Sortieren durch Mischen

Anwenden der Divide-and-Conquer-Strategie zum Entwurf von Mergesort

- Zerlege das Problem in kleinere Teilprobleme.
  - Zerlegen in zwei gleich große Teillisten
  - (bei ungerader Anzahl Längenunterschied von einem Element)
  - Listen mit ≤ 1 Elementen sind bereits sortiert (Abbruch der Rekursion)
- Wende den Algorithmus rekursiv auf die Teilprobleme an.

- Füge die Teilergebnisse wieder zusammen.

Mergesort / Sortieren durch Mischen

Anwenden der Divide-and-Conquer-Strategie zum Entwurf von Mergesort

- Zerlege das Problem in kleinere Teilprobleme.
  - Zerlegen in zwei gleich große Teillisten
  - (bei ungerader Anzahl Längenunterschied von einem Element)
  - Listen mit ≤ 1 Elementen sind bereits sortiert (Abbruch der Rekursion)
- Wende den Algorithmus rekursiv auf die Teilprobleme an.
  - Liefert zwei gleich große sortierte Teillisten
- Füge die Teilergebnisse wieder zusammen.

Mergesort / Sortieren durch Mischen

Anwenden der Divide-and-Conquer-Strategie zum Entwurf von Mergesort

- Zerlege das Problem in kleinere Teilprobleme.
  - Zerlegen in zwei gleich große Teillisten
  - (bei ungerader Anzahl Längenunterschied von einem Element)
  - Listen mit ≤ 1 Elementen sind bereits sortiert (Abbruch der Rekursion)
- Wende den Algorithmus rekursiv auf die Teilprobleme an.
  - Liefert zwei gleich große sortierte Teillisten
- Füge die Teilergebnisse wieder zusammen.
  - Mischen der zwei Listen zu einer Ergebnis-Liste
  - Wähle jeweils das kleinere Element von den beiden Listen und füge es vorne an das Ergebnis ein

Mergesort Beispiel

- Gegeben sei die Liste $[17, 12, 6, 19, 23, 8, 5, 10]$.
- Die Liste wird aufgeteilt in $[17, 12, 6, 19]$ und $[23, 8, 5, 10]$.
- Die zwei Teillisten werden durch rekursive Aufrufe sortiert: $[6, 12, 17, 19]$ und $[5, 8, 10, 23]$
- Mischen: immer das kleinste Element vorne wegnehmen:

Ergebnis	Teilliste 1	Teilliste 2
[]	$[6, 12, 17, 19]$	$[5, 8, 10, 23]$

Mergesort Beispiel

- Gegeben sei die Liste $[17, 12, 6, 19, 23, 8, 5, 10]$.
- Die Liste wird aufgeteilt in $[17, 12, 6, 19]$ und $[23, 8, 5, 10]$.
- Die zwei Teillisten werden durch rekursive Aufrufe sortiert: $[6, 12, 17, 19]$ und $[5, 8, 10, 23]$
- Mischen: immer das kleinste Element vorne wegnehmen:

Ergebnis	Teilliste 1	Teilliste 2
$[]$	$[6, 12, 17, 19]$	$[5, 8, 10, 23]$
$[5]$	$[6, 12, 17, 19]$	$[8, 10, 23]$

Mergesort Beispiel

- Gegeben sei die Liste $[17, 12, 6, 19, 23, 8, 5, 10]$.
- Die Liste wird aufgeteilt in $[17, 12, 6, 19]$ und $[23, 8, 5, 10]$.
- Die zwei Teillisten werden durch rekursive Aufrufe sortiert: $[6, 12, 17, 19]$ und $[5, 8, 10, 23]$
- Mischen: immer das kleinste Element vorne wegnehmen:

Ergebnis	Teilliste 1	Teilliste 2
$[]$	$[6, 12, 17, 19]$	$[5, 8, 10, 23]$
$[5]$	$[6, 12, 17, 19]$	$[8, 10, 23]$
$[5, 6]$	$[12, 17, 19]$	$[8, 10, 23]$

Mergesort Beispiel

- Gegeben sei die Liste $[17, 12, 6, 19, 23, 8, 5, 10]$.
- Die Liste wird aufgeteilt in $[17, 12, 6, 19]$ und $[23, 8, 5, 10]$.
- Die zwei Teillisten werden durch rekursive Aufrufe sortiert: $[6, 12, 17, 19]$ und $[5, 8, 10, 23]$
- Mischen: immer das kleinste Element vorne wegnehmen:

Ergebnis	Teilliste 1	Teilliste 2
$[]$	$[6, 12, 17, 19]$	$[5, 8, 10, 23]$
$[5]$	$[6, 12, 17, 19]$	$[8, 10, 23]$
$[5, 6]$	$[12, 17, 19]$	$[8, 10, 23]$
$[5, 6, 8]$	$[12, 17, 19]$	$[10, 23]$

Mergesort Beispiel

- Gegeben sei die Liste $[17, 12, 6, 19, 23, 8, 5, 10]$.
- Die Liste wird aufgeteilt in $[17, 12, 6, 19]$ und $[23, 8, 5, 10]$.
- Die zwei Teillisten werden durch rekursive Aufrufe sortiert: $[6, 12, 17, 19]$ und $[5, 8, 10, 23]$
- Mischen: immer das kleinste Element vorne wegnehmen:

Ergebnis	Teilliste 1	Teilliste 2
$[]$	$[6, 12, 17, 19]$	$[5, 8, 10, 23]$
$[5]$	$[6, 12, 17, 19]$	$[8, 10, 23]$
$[5, 6]$	$[12, 17, 19]$	$[8, 10, 23]$
$[5, 6, 8]$	$[12, 17, 19]$	$[10, 23]$
$[5, 6, 8, 10]$	$[12, 17, 19]$	$[23]$

Mergesort Beispiel

- Gegeben sei die Liste $[17, 12, 6, 19, 23, 8, 5, 10]$.
- Die Liste wird aufgeteilt in $[17, 12, 6, 19]$ und $[23, 8, 5, 10]$.
- Die zwei Teillisten werden durch rekursive Aufrufe sortiert: $[6, 12, 17, 19]$ und $[5, 8, 10, 23]$
- Mischen: immer das kleinste Element vorne wegnehmen:

Ergebnis	Teilliste 1	Teilliste 2
$[]$	$[6, 12, 17, 19]$	$[5, 8, 10, 23]$
$[5]$	$[6, 12, 17, 19]$	$[8, 10, 23]$
$[5, 6]$	$[12, 17, 19]$	$[8, 10, 23]$
$[5, 6, 8]$	$[12, 17, 19]$	$[10, 23]$
$[5, 6, 8, 10]$	$[12, 17, 19]$	$[23]$
$[5, 6, 8, 10, 12]$	$[17, 19]$	$[23]$

Mergesort Beispiel

- Gegeben sei die Liste $[17, 12, 6, 19, 23, 8, 5, 10]$.
- Die Liste wird aufgeteilt in $[17, 12, 6, 19]$ und $[23, 8, 5, 10]$.
- Die zwei Teillisten werden durch rekursive Aufrufe sortiert: $[6, 12, 17, 19]$ und $[5, 8, 10, 23]$
- Mischen: immer das kleinste Element vorne wegnehmen:

Ergebnis	Teilliste 1	Teilliste 2
$[]$	$[6, 12, 17, 19]$	$[5, 8, 10, 23]$
$[5]$	$[6, 12, 17, 19]$	$[8, 10, 23]$
$[5, 6]$	$[12, 17, 19]$	$[8, 10, 23]$
$[5, 6, 8]$	$[12, 17, 19]$	$[10, 23]$
$[5, 6, 8, 10]$	$[12, 17, 19]$	$[23]$
$[5, 6, 8, 10, 12]$	$[17, 19]$	$[23]$
$[5, 6, 8, 10, 12, 17]$	$[19]$	$[23]$

Mergesort Beispiel

- Gegeben sei die Liste $[17, 12, 6, 19, 23, 8, 5, 10]$.
- Die Liste wird aufgeteilt in $[17, 12, 6, 19]$ und $[23, 8, 5, 10]$.
- Die zwei Teillisten werden durch rekursive Aufrufe sortiert: $[6, 12, 17, 19]$ und $[5, 8, 10, 23]$
- Mischen: immer das kleinste Element vorne wegnehmen:

Ergebnis	Teilliste 1	Teilliste 2
$[]$	$[6, 12, 17, 19]$	$[5, 8, 10, 23]$
$[5]$	$[6, 12, 17, 19]$	$[8, 10, 23]$
$[5, 6]$	$[12, 17, 19]$	$[8, 10, 23]$
$[5, 6, 8]$	$[12, 17, 19]$	$[10, 23]$
$[5, 6, 8, 10]$	$[12, 17, 19]$	$[23]$
$[5, 6, 8, 10, 12]$	$[17, 19]$	$[23]$
$[5, 6, 8, 10, 12, 17]$	$[19]$	$[23]$
$[5, 6, 8, 10, 12, 17, 19]$	$[]$	$[23]$

Mergesort Beispiel

- Gegeben sei die Liste $[17, 12, 6, 19, 23, 8, 5, 10]$.
- Die Liste wird aufgeteilt in $[17, 12, 6, 19]$ und $[23, 8, 5, 10]$.
- Die zwei Teillisten werden durch rekursive Aufrufe sortiert: $[6, 12, 17, 19]$ und $[5, 8, 10, 23]$
- Mischen: immer das kleinste Element vorne wegnehmen:

Ergebnis	Teilliste 1	Teilliste 2
$[]$	$[6, 12, 17, 19]$	$[5, 8, 10, 23]$
$[5]$	$[6, 12, 17, 19]$	$[8, 10, 23]$
$[5, 6]$	$[12, 17, 19]$	$[8, 10, 23]$
$[5, 6, 8]$	$[12, 17, 19]$	$[10, 23]$
$[5, 6, 8, 10]$	$[12, 17, 19]$	$[23]$
$[5, 6, 8, 10, 12]$	$[17, 19]$	$[23]$
$[5, 6, 8, 10, 12, 17]$	$[19]$	$[23]$
$[5, 6, 8, 10, 12, 17, 19]$	$[]$	$[23]$
$[5, 6, 8, 10, 12, 17, 19, 23]$	$[]$	$[]$

Implementierung von Mergesort

Siehe Übungsblatt!

Quicksort

Anwenden der Divide-and-Conquer-Strategie zum Entwurf von Quicksort

- Zerlege das Problem in kleinere Teilprobleme.
- Wende den Algorithmus rekursiv auf die Teilprobleme an.
- Füge die Teilergebnisse wieder zusammen.

Quicksort

Anwenden der Divide-and-Conquer-Strategie zum Entwurf von Quicksort

- Zerlege das Problem in kleinere Teilprobleme.
  - Wähle ein beliebiges Element p aus der Liste aus (**Pivotelement**)
  - Teile die restlichen Elemente in zwei Teillisten:
    Teil 1: die Elemente $< p$.
    Teil 2: die Elemente $\geq p$
- Wende den Algorithmus rekursiv auf die Teilprobleme an.

- Füge die Teilergebnisse wieder zusammen.

Quicksort

Anwenden der Divide-and-Conquer-Strategie zum Entwurf von Quicksort

- Zerlege das Problem in kleinere Teilprobleme.
  - Wähle ein beliebiges Element p aus der Liste aus (**Pivotelement**)
  - Teile die restlichen Elemente in zwei Teillisten:
    Teil 1: die Elemente $< p$.
    Teil 2: die Elemente $\geq p$
- Wende den Algorithmus rekursiv auf die Teilprobleme an.
  - Liefert sortierten Teil 1 und Teil 2
- Füge die Teilergebnisse wieder zusammen.

Quicksort

Anwenden der Divide-and-Conquer-Strategie zum Entwurf von Quicksort

- Zerlege das Problem in kleinere Teilprobleme.
  - Wähle ein beliebiges Element p aus der Liste aus (**Pivotelement**)
  - Teile die restlichen Elemente in zwei Teillisten:
    Teil 1: die Elemente $< p$.
    Teil 2: die Elemente $\geq p$
- Wende den Algorithmus rekursiv auf die Teilprobleme an.
  - Liefert sortierten Teil 1 und Teil 2
- Füge die Teilergebnisse wieder zusammen.
  - Erst Teil 1, dann das Pivotelement p, dann Teil 2

Quicksort Beispiel

- Gegeben Sei die Liste $[17, 12, 6, 19, 23, 8, 5, 10]$.
- Wähle zum Beispiel 10 als Pivotelement.
- Teil 1: Elemente < 10:
  zum Beispiel: $[5, 8, 6]$
- Teil 2: Elemente ≥ 10:
  zum Beispiel: $[23, 12, 17, 19]$
- Sortiere die 2 Teile:
  Teil 1: $[5, 6, 8]$
  Teil 2: $[12, 17, 19, 23]$
- Zusammenfügen: $[5, 6, 8, \quad 10, \quad 12, 17, 19, 23]$

Beispiel: Partitionierung auf Arrays

17	12	6	19	23	8	5	10

Wähle als Pivotelement das letzte Element (10):

17	12	6	19	23	8	5	10

Aufteilen der Übrigen Elemente:

17	12	6	19	23	8	5	10

Vertausche 5 und 17:

5	12	6	19	23	8	17	10

Vertausche 8 und 12:

5	8	6	19	23	12	17	10

Keine Vertauschung mehr möglich:

5	8	6	19	23	12	17	10

Tausche nun das Pivotelement zwischen die beiden Partitionen:

5	8	6	10	23	12	17	19

Partitionierung auf Arrays I

- Bearbeite rekursiv Teilbereiche des Arrays (`n` Elemente, startend bei `f`)
- Realisiere das Teilen der Liste durch Vertauschen
  - → Indexzähler `left`, `right` laufen von links bzw. rechts und suchen Einträge, die vertauscht werden können.
  - → Für die zu tauschenden Einträge gilt:

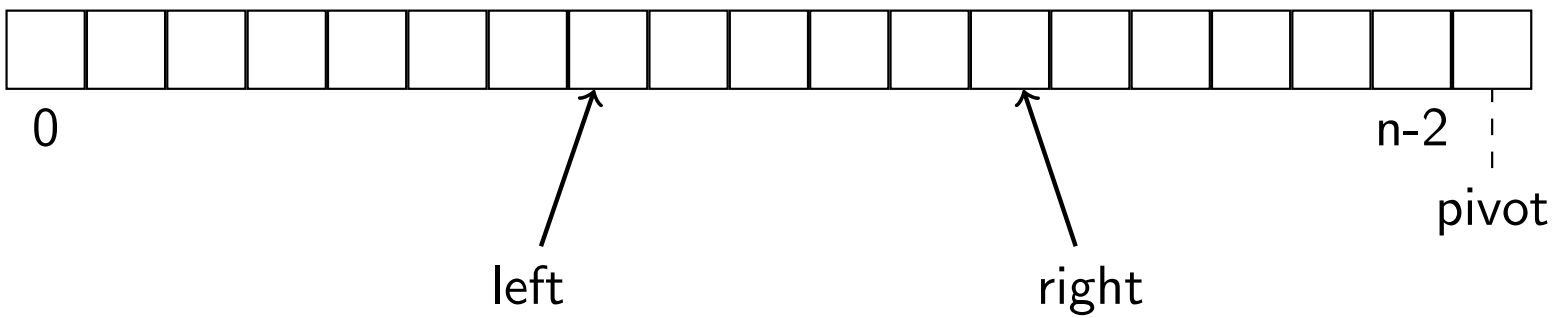
    $$\texttt{f[left]} \geq \texttt{pivot} \text{ und } \texttt{f[right]} < \texttt{pivot}$$

  - → In jedem Schritt gilt:
    - Für alle i in [0, `left-1`] : `f[i]` < `pivot`
    - Für alle i in [`right+1`, `n-2`] : `pivot` $\leq$ `f[i]`

Partitionierung auf Arrays II

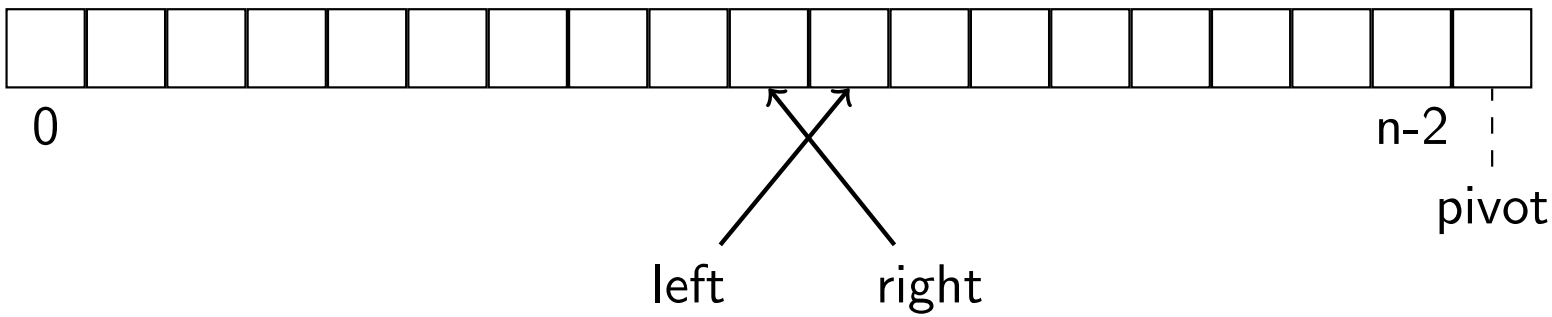
Wenn ein Paar zum Vertauschen gefunden wurde gilt `left < right`:

- Vertausche `f[left]` und `f[right]`
- Inkrementiere `left` und dekrementiere `right`
- Fahre dann mit der Suche nach vertauschbaren Elementpaaren fort.



Partitionierung auf Arrays III

Die Umsortierung des (Teil-)Arrays ist abgeschlossen, sobald `left > right`.



Zum Schluss wird das `pivot` an die richtige Stelle getauscht, indem das Element an Position `left` und das an Position `n-1` vertauscht werden.

Implementierung der Partitionierung

```
int partition(int *f, int n){
  int left = 0;
  int right = n - 2;

  int pivot = f[n-1];
  while (true) {
    while (f[left] < pivot) { left++; }
    while (left <= right && f[right] >= pivot){ right--; }
    if( left > right ) {
      break;
    } else {
      swap(&f[left], &f[right]);
      left++; right--;
    }
  }
  // Pivotelement kommt zwischen die beiden Partitionen
  swap(&f[left], &f[n-1]);
  return left;
}
```

Implementierung von Quicksort

```
void quicksort(int *f, int n) {
  if (n > 1) {
    int ixsplit = partition2(f, n);
    quicksort(f, ixsplit);
    quicksort(&f[ixsplit+1], n - 1 - ixsplit);
  }
}
```

Optimierungen

Die vorgestellte Quicksort-Fassung arbeitet schlecht auf schon sortierten Arrays. Dies kann man durch folgende Strategien verhindern:

- → Shuffle des Arrays, um Vorsortierung aufzuheben
- → Randomisierte Auswahl des Pivot-Elements (Beispiel: Median von 3 Elementen)

Der Aufwand für die rekursiven Funktionsaufrufe ist außerdem vergleichsweise hoch, wenn die Teil-Arrays nur noch wenige Elemente enthalten. Man kann die Anzahl dieser Aufrufe reduzieren, in dem andere Sortierverfahren wie InsertionSort zur Sortierung von Teil-Arrays mit nur wenigen Elemente verwendet wird.

Übersicht Sortieralgorithmen

- Insertionsort:
  Einzelne Elemente Sortiert in Liste einfügen
- Selectionsort:
  Minimum entfernen und Vorne einfügen
- Mergesort:
  Liste in gleichgroße Teile aufteilen
- Quicksort:
  Pivotelement auswählen, Liste in kleinere und größere Aufteilen

Ausblick

Offene Frage: Welcher Sortieralgorithmus ist der Schnellste?

Nächste Vorlesung:

- Algorithmenanalyse
- Messen der Laufzeit