Informatik - Klausurvorbeireitung - 23.11.17

Selection Sort

Es wird das Array von index 0 bis $n-1$ durchlaufen. Zunächst gilt das es keinen sortierten Teil gibt, dieser kommt aber beim Durchlaufen des Arrays. Die innere Schleife wird von $i+1$ bis zum Ende des Arrays durchlaufen, es wird nach dem kleinsten Wert gesucht. Dieser wird dann mit dem ersten Wert des unsortierten Teils getauscht, wodurch der sortierte Teil um eins wächst, da dieser Wert jetzt am richtigen Platz ist. Man durchläuft die innere Schleife von $i+1$, da man sonst im ersten Vergleich `arr[i]` mit `arr[i]` vergleichen würde. Es wird die außere Schleife bis zu $n-1$ durchlaufen, da man weiß, dass der letzte Wert sortiert sein muss, wenn alle anderen am richtigen Platz sind.

```
1  for(int i = 0; i < arr.length - 1; i++) {
2    int smallest_index = i;
3
4    // kleinsten Wert im unsortierten Bereich finden
5    for(int j = i+1; j < arr.length; j++)
6      if(arr[smallest_index] > arr[j])
7        smallest_index = j;
8
9    // kleinsten Wert mit Anfang des unsortierten Teils tauschen
10   int temp = arr[smallest_index];
11   arr[smallest_index] = arr[i];
12   arr[i] = temp;
13 }
```

Array: `[4, 3, 2, 1]`

i	Array	C (Comparisons)	M (Moves)
0	`1, 3, 2, 4`	3 (*6: true*, *6: true*, *6: true*)	3 (*10*, *11*, *12*; 1 → 4)
1	`1, 2, 3, 4`	2 (*6: true 6: false*)	3 (*10*, *11*, *12*; 2 → 3)
2	`1, 2, 3, 4`	1 (*6: false*)	3 (*10*, *11*, *12*; 3 → 3)

$$C = \sum_{i=1}^{n-1} i = \frac{(n-1)\cdot((n-1)+1)}{2} = \frac{n\cdot(n-1)}{2}$$

$$C = \sum_{i=1}^{4-1} i = \frac{4\cdot(4-1)}{2} = 6$$

$$M = \sum_{i=1}^{n-1} 3 = 3\cdot(n-1)$$

$$M = \sum_{i=1}^{4-1} 3 = 3\cdot(4-1) = 9$$

Generell kann man sagen, dass die **Anzahl der Vergleiche** egal wie das Array sortiert ist gleich ist. Die **Anzahl der Movements** sollte auch gleich sein, da nicht extra nocheinmal überprüft wird ob überhaupt getauscht werden muss. Wenn nicht getauscht werden muss heißt das, dass der kleinste Wert schon am richtigen Platz ist. Da dies selten der Fall ist und die Vergleiche zum überprüfen ob getauscht werden soll viel mehr Aufwand bringen als einfach immer zu tauschen wird immer getauscht auch wenn dies nicht nötig wäre, da dies herauszufinden mehr Aufwand heißen würde.

Insertion Sort

Es wird das Array von index 1 bis n durchlaufen. Zunächst gilt der erste Werte des gesamten Arrays (`arr[0]`) als sortiert. Bei jedem Durchlauf wird der Wert bei `arr[i]` in das schon sortierte Array einsortiert. Am Ende besteht das ganze Array dann aus dem sortierten Teil des Arrays. Es wird der Wert bei `arr[i]` in `temp` kopiert. Danach wird solange der schon sortierte Teil des Arrays durchlaufen bis man die richtige Stelle für `arr[i]` / `temp` gefunden hat. Dabei wird jeweils der Wert am aktuellen index `j` nach `j+1` geschoben, da Platz für den neuen Wert gemacht werden muss, der an einer Stelle des sortierten Teil des Arrays einfügt werden soll. Sobald dieser Platz gefunden wurde wird das Durchlaufen des schon sortierten Arrays abgebrochen und der Wert (`temp`) an den entsprechenden Platz kopiert.

```
1  // Array wird von 1 bis n durchlaufen (index = i)
2  for(int i=1; i<arr.length; i++) {
3    int tmp = arr[i];
4    // Array wird von i-1 bis 0 rückwerts durchlaufen (index = j)
5    for(int j=i-1; j >= 0; j--) {
6      // es wird der index des inneren loops mit `tmp` verglichen
7      if(arr[j] > tmp) {
8        // Wert bei `j` wird eins nach rechts geschoben (`j+1`)
9        arr[j+1] = arr[j];
10       // falls man bei j = 0 angekommen ist, ist klar, dass der Wert
11       // in `tmp` der kleinste Wert ist und somit am Anfang des Arrays muss
12       if(j == 0)
13         arr[0] = tmp;
14     } else {
15       // wenn `tmp` größer / gleich arr[j] ist, dann muss als Wert **vor** arr[j]
16       // gesetzt werden (also arr[j+1], da **vor** sich auf die Reihenfolge der
17       // verglichenen Werte bezieht)
18       arr[j+1] = tmp;
19       break;
20     }
21   }
22 }
```

Wenn man von einem Array mit Länge 4 ausgeht, z.B. `[4, 3, 2, 1]` , dann wird die erste Schleife 3 mal ausgeführt (von 2 bis 4 also 2, 3, 4). Die innere Schleife wird dann in jedem Durchlauf der äußeren Schleife i (von $i - 1$ bis 0, also rückwerts) mal durchlaufen, solange sie nicht vorher abgebrochen wird indem der richtige Platz für den Wert gefunden wurde

Das bedeutet, im **worst case** (ein sortiertes Array, nur **reversed**) wird die innere Schleife nie vorher abgebrochen, da der Wert der einsortiert werden soll immer an die erste Stelle muss. Das heißt, dass die innere Schleife

$$\sum_{i=2}^{4} i = \sum_{i=1}^{4} i - \sum_{i=1}^{2-1} i = (1+2+3+4) - (1) = \frac{n \cdot (n+1)}{2} - 1$$

durchlaufen wird

> Summen regeln:

$$\sum_{i=1}^{n} i = \frac{n \cdot (n+1)}{2}$$

$$\sum_{i=k}^{n} i = \sum_{i=1}^{n} i - \sum_{i=1}^{k-1} i = \frac{n \cdot (n+1)}{2} - \frac{(k-1) \cdot ((k-1)+1)}{2}$$

Array: `[4, 3, 2, 1]`

i	Array	C (Comparisons)	M (Moves)
1	`3, 4, 2, 1`	2 (*7: true*, *12: true*)	3 (*3*, *9*, *13*)
2	`2, 3, 4, 1`	4 (*7: true*, *12: false*, *7*, *12: true*)	4 (*3*, *9*, *9*, *13*)
3	`1, 2, 3, 4`	6 (*7: true*, *12: false*, *7*, *12: false*, *7: true*, *12: true*)	5 (*3*, *9*, *9*, *9*, *13*)

Bei einem rückwerts sortiertem Array wird für jeden Wert im sortierten Bereich 2 mal verglichen und $2 + n$ mal gemoved (1 mal in tmp, 1 mal von tmp zum richtigen Platz).

$$C = \sum_{i=1}^{n-1} 2i = 2 \cdot \frac{(n-1) \cdot ((n-1)+1)}{2} = (n-1) \cdot n$$

$$C = \sum_{i=1}^{4-1} 2i = 2 \cdot \frac{(4-1) \cdot ((4-1)+1)}{2} = 3 \cdot 4 = 12$$

$$M = \sum_{i=1}^{n-1} 2 + i = (n-1) \cdot 2 + \sum_{i=1}^{n-1} i = (n-1) \cdot 2 + \frac{(n-1) \cdot n}{2}$$

$$M = \sum_{i=1}^{4-1} 2 + i = 6 + \frac{(4-1) \cdot 4}{2} = 12$$

Bubble Sort

Es wird das Array von index 0 bis n durchlaufen. Die innere Schleife wird von 0 bis $n - i - 1$ durchlaufen. Die innere schleife vergleicht den Wert beim index `j` (innere Schleife) mit dem nächsten Wert. Da man ja jeweils 2 Werte miteinander vergleicht braucht man $n - 1$ Vergleiche. Da sich aber von hinten aus ein sortierter Teil bildet, welcher mit jedem Durchlauf der inneren Schleife um eins wächst, zieht man nochmal den index `i` (äußere Schleife) ab, damit man den sortierten Teil nicht nochmal durchläuft. Wenn `arr[j]` größer ist als der nächste Wert, also `arr[j+1]` , dann wird `arr[j]` mit `arr[j+1]`

getauscht (3 movements). Dies passiert direkt nach dem Vergleichen von `arr[j]` und `arr[j+1]` . Sobald der sortierte Teil so groß ist wie das gesamte Array ist das Array sortiert.

```
int temp;
for(int i = 0; i < arr.length; i++)
  for(int j = 0; j < arr.length - i - 1; j++)
    if(arr[j] > arr[j+1]) {
      temp = arr[j];
      arr[j] = arr[j+1];
      arr[j+1] = temp;
    }
```

Array: `[4, 3, 2, 1]`

i	Array	C (Comparisons)	M (Moves)
0	`3, 2, 1, 4`	3 (*4: true*, *4: true*, *4: true*)	9 (*5*, *6*, *7*; 4 → 3; *5*, *6*, *7*; 4 → 2; *5*, *6*, *7*; 4 → 1)
1	`2, 1, 3, 4`	2 (*4: true*, *4: true*)	6 (*5*, *6*, *7*; 3 → 2; *5*, *6*, *7*; 3 → 1)
2	`1, 2, 3, 4`	1 (*4: true*)	3 (*5*, *6*, *7*; 1 → 2)

$$C = \sum_{i=1}^{n-1} i = \frac{(n-1)\cdot((n-1)+1)}{2} = \frac{n\cdot(n-1)}{2}$$

$$C = \sum_{i=1}^{4-1} i = \frac{(4-1)\cdot((4-1)+1)}{2} = \frac{4\cdot(4-1)}{2} = 6$$

$$M = \sum_{i=1}^{n-1} 3i = 3\cdot\sum_{i=1}^{n-1} = 3\cdot\frac{(n-1)\cdot((n-1)+1)}{2} = 3\cdot\frac{n\cdot(n-1)}{2}$$

$$M = \sum_{i=1}^{4-1} 3i = 3\cdot\sum_{i=1}^{4-1} = 3\cdot\frac{(4-1)\cdot((4-1)+1)}{2} = 3\cdot\frac{4\cdot(4-1)}{2} = 18$$

Die **Anzahl der Vergleiche** verändert sich nicht, egal wie das Array sortiert ist. Die **Anzahl der movements** ändert sich von minimal 0 (wenn das Array schon sortiert ist) zu maximal $3\cdot\frac{n\cdot(n-1)}{2}$ (wenn das Array rückwerts sortiert ist).