Assignment 3 - Searching in java

Jesper Hesselgren

August 21, 2024

Introduktion

Syftet med denna uppgift är att analysera och jämföra tidskomplexiteten hos tre olika sorteringsalgoritmer. De algoritmer som ska undersökas är **Selection Sort**, **Insertion Sort**, **Merge Sort**. Effektiviteten hos varje algoritm undersöks genom att mäta deras prestanda på slumpmässigt genererade arrayer av olika storlekar som ska storleksordnas från minsta till största.

Selection sort

Selection sort algoritmen fungerar genom att upprepade gånger leta efter det minsta elementet i den osorterade delen av arrayen och placera det längst fram. Efter varje iteration betraktas den främre delen av arrayen som sorterad, och algoritmen fortsätter med att söka det minsta elementet i den återstående osorterade delen tills hela arrayen är helt sorterad. Selection sort algoritmen har tidskomplexiteten $O(n^2)$.

Insertion sort

Insertion sort-algoritmen fungerar genom att bygga upp en sorterad del av arrayen genom att infoga varje element på rätt plats. Den börjar med att ta de andra elementet och jämför det med föregående element i den sorterade delen. Om det är mindre än föregående element flyttas det bakåt i arrayen tills det når sin korrekta position. Denna metod upprepas för varje nytt element tills hela arrayen är sorterad. Insertion sort algortimen har en tidskomplexitet av $O(n^2)$ i värsta fall. Värsta fall för insertion sort algoritmen är när arrayen är omvänt sorterad.

Selection sort vs Insertion sort

Selection sort och Insertion sort implementeras på olika sätt, men båda har en tidskomplexitet på $O(n^2)$. I grafen nedan kan man se hur algoritmerna

presterat på arrayer av olika storlekar (100, 200, 400, 800, 1600, 3200, 6400 element). Det framgår tydligt att båda algoritmernas tidskomplexitet växer kvadratiskt, men att de presterar likvärdigt på mindre arrayer. Först vid större arraystorlekar blir skillnaden tydlig, där Insertion sort visar sig vara mer effektiv än Selection sort.

Figur 1: Sortering av n element med olika algoritmer

Merge sort

Merge sort är en algoritm som implementeras rekursivt. Algoritmen fungerar genom att den tar emot en array som input och rekursivt delar arrayen i två delar, sortera dessa, och sedan slår ihop dem till en enda sorterad array. Algoritmen börjar med att kontrollera om arrayen har färre än två element (basfallet). Om så är fallet avslutas funktionen, eftersom en sådan array redan är sorterad.

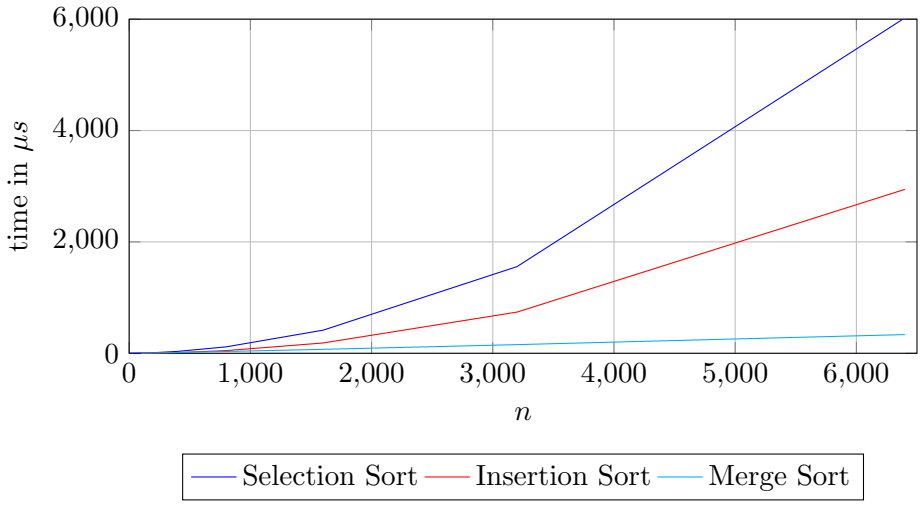
```
// Base case: If the array has less than 2 elements, do nothing
  if (arrayLength < 2) {
      return;
  }
  int mid = arrayLength / 2;

  int[] leftArray = new int[mid];
  int[] rightArray = new int[arrayLength - mid];

  // Copy the left half of the array into leftArray
  for (int i = 0; i < mid; i++) {
      leftArray[i] = array[i];
  }
```

```
        // Copy the right half of the array into rightArray
        for (int i = mid; i < arrayLength; i++) {
            rightArray[i - mid] = array[i];
        }

        // Recursively sort the left and right arrays
        mergeSort(leftArray);
        mergeSort(rightArray);

        // Merge the sorted left and right arrays
        merge(leftArray, rightArray, array);
    }
```

Merge-metoden fungerar genom att slå samman två sorterade delarrayer, vänster- och högerarrayen, till en enda sorterad array. Den börjar med att jämföra det första elementet i båda delarna och placerar det minsta av de två i den ursprungliga arrayen. Detta upprepas tills alla element från någon av delarna har placerats i den sorterade arrayen. När den ena arrayen har tömts kopieras de återstående elementen från den andra delen direkt in i den ursprungliga arrayen.

```
        int l = 0; , int r = 0; , int k = 0;

        while (l < leftLength && r < rightLength) {
            if (leftArray[l] <= rightArray[r]) {
                array[k] = leftArray[l];
                l++;
            } else {
                array[k] = rightArray[r];
                r++;
            }
            k++;
        }
```

Merge sort algoritmen har en tidskomplexitet på $O(nlog(n)$. I grafen nedan går det att rent visuellt att se hur mycket snabbare merge sort algoritmen är än det två tidigare algoritmerna.



Figur 2: Sortering av n element med selection sort, insertion sort och merge sort

Antalet element	**Merge**(us)	**Insertion**us)	**Selection**(us)
100	2.71	1.21	3.41
200	6.54	4.00	10.6
400	14.4	12.9	34.0
800	32.5	47.9	115.5
1600	70.7	185.5	415.1
3200	155.5	740.0	1556.4
6400	335.9	2943.8	6022.9

Tabell 1: Jämföreslse av hur algoritmerna presterar på arrayer av olika storlekar

Diskussion/Slutsats

Det kan konstateras att Merge Sort är den mest effektiva algoritmen för att sortera stora datamängder, tack vare dess tidskomplexitet på $O(n \log(n))$. Selection Sort har en betydligt sämre tidskomplexitet på $O(n^2)$, vilket även gäller för Insertion Sort i dess värsta fall, som inträffar när arrayen är omvänt sorterad. Trots detta presterar Insertion Sort bättre än Selection Sort i detta test, vilket indikerar att Insertion Sort i praktiken ofta har en bättre genomsnittlig prestanda än dess teoretiska värsta fall ($O(n^2)$).

Baserat på den presenterade grafen och tabellen kan vi dra slutsatsen att algoritmerna presterar relativt lika för små datamängder. Det är först när

datamängden blir större som skillnaderna i prestanda blir tydliga. Med denna kunskap är det viktigt att ha en uppfattning om hur stor datamängd som ska sorteras innan man väljer att implementera en specifik algoritm.