Algoritmer och Datasturkturer
Programmeringsuppgift 4
Av
Daniel Westerlund
2020-02-26

Uppgift 3

multiplyAll

ArrayList

Enligt google doc så varierar resultat på 40 000 olika element ifrån 0 till 3760, vilket är en extrem stor skillnad och det har att göra med hur implementationen ser ut. Med brutal-force metoden fick jag något liknande, men med en optimerad algoritm så fick jag ner den till 8, dock anmärkningsvärt att den minskar till mellan 2 och 3 på de andra testerna.
En del kan förklaras med att cashningen gör exekveringen snabbare.

LinkedList

Datan i google doc varierar även här kraftigt, med 3000 element så är det även här ifrån 0 till över 10 000, vilken har att göra med hur den inre algoritmen är implementerad.

Kostnad

Tidskomplexiteten för min algoritm är O(N) eftersom den går igenom listan en gång för att summera ihop alla element och där tar den bort summan av ett element ifrån den totala summan för att sedan göra en multiplikation med den totala summan.
Dock så växer LinkedList snabbare än vad ArrayList gör. Eftersom linkedList har högre kostnad för att hämta element har den en tidskomplexitet på 2N men blir amorterad O(N).

multiplySum

ArrayList

Min algoritm var ganska långsam jämför med snitttiden. Dock verkar det som att de personer som har använt iterator, att deras algoritm går snabbare att exekvera.

LinkedList

Även här kan iterator spela stor roll när man jämfört med andras tester. Finns en del vars algoritmer är ~4 – 7 gånger snabbare än min på ArrayList men sedan Linked list marginellt snabbare eller till och med dubbel så långsam. Visst spelar hårdvara och cpu Load in, men endå anmärkningsvärt.

Kostnad

ArrayList har O(N) eftersom den har direkt access till elementen medan LinkedList behöver gå igenom listan för att hitta elementen vilket ger att den har $O(N^2)$ med mina algoritmer. Men ser man i dokumentet så finns det personer som fått ner båda till O(N).

Uppgift 4

Insertionsort

Den överlägset långsammaste algoritmen i uppgift 4. Insertions sort är $O(N^2)$, och som spreadsheetet visar så ökar exekveringstiden snabbt hos alla om man ökar antalet element.

Drakesort

Genom att använda extra minne så behöver man bara gå igenom arrayen två gånger vilket ger O(N) och dubblerar man antalet element så förlängs exekveringstiden lite mindre än dubbelt. Vilket stämmer överens med O(N).

Quicksort

Har $O(N^2)$ dock med $\Theta(n*\log(n))$. Om man jämför Quicksort med Drake sort kan man se att Quicksort är ganska mycket långsammare då den växer betydligt mycket fortare och plottar man den, ser den ut att gå i mot n*log(n).

Java API sort

Jag trodde sort använde Quicksort som standard, men efter dessa tester kan man se att det är någon annan algoritm som är snabbare och verkar vara O(N) alternativt O (n*log(n)). Skulle kunna t.ex. vara något sökträd som BST eller Timsort.