Algoritmer Elias Cherkaoui

Algorithm	Worst-case running time	Average-case/expected running time
Insertion sort	$\Theta(n^2)$	$\Theta(n^2)$
Merge sort	$\Theta(n \lg n)$	$\Theta(n \lg n)$
Heapsort	$O(n \lg n)$	—
Quicksort	$\Theta(n^2)$	$\Theta(n \lg n)$ (expected)
Counting sort	$\Theta(k + n)$	$\Theta(k + n)$
Radix sort	$\Theta(d(n + k))$	$\Theta(d(n + k))$
Bucket sort	$\Theta(n^2)$	$\Theta(n)$ (average-case)

1000 Insertion sort runtime: 0.743994 ms.
10000 Insertion sort runtime: 65.8132 ms.
100 000 Insertion sort Runtime is: 7895.73 ms.
100 000 Insertion sort Runtime is: 8118.64 ms.
100 000 Insertion sort Runtime is: 8072.83 ms.
100 000 Insertion sort Runtime is: 7996.81 ms.
100 000 Insertion sort Runtime is: 8015.38 ms.

1000 Quicksort runtime: 1.3897 ms.
10000 Quicksort runtime: 55.34ms
100 000 Quicksort Runtime is: 20.4813 ms.
100 000 Quicksort Runtime is: 20.9805 ms.
100 000 Quicksort Runtime is: 22.01 ms.
100 000 Quicksort Runtime is: 24.4121 ms.
100 000 Quicksort Runtime is: 23.6012 ms.

1000 Heap sort runtime: 0.081485 ms.
10000 Heap sort runtime: 0.93908 ms.
100 000 Heap sort Runtime is: 25.2077 ms.
100 000 Heap sort Runtime is: 25.8096 ms.
100 000 Heap sort Runtime is: 25.8929 ms.
100 000 Heap sort Runtime is: 25.3501 ms.
100 000 Heap sort Runtime is: 25.6948 ms.

Insertionsort beräkningar

Beräkning för C Insertion sort 1000 tal:
C * 1000^2 = 0.743994(ms)
C = 0.743994(ms) / 1000^2 ~= 0.000000743994(C)

Beräkning för C Insertion sort 10000 tal:
C * 10000^2 = 65.8132(ms)
C = 65.8132(ms) / 10000^2 ~= 0.000000658132(C)

Beräkning för T(n) Insertion sort 100 000 tal:
(65.8132(ms) / 10000^2) * 100 000^2 = 6581.32(ms)

Beräkning för T(n) Insertion sort 1 000 000 tal:
(65.8132(ms) / 10000^2) * 1 000 000^2 = 658132(ms)

Quicksort beräkningar

Beräkning för C Quicksort 1000 tal:
C * 1000^2 = 0.719051(ms)
C = 0.719051(ms) / 1000^2 ~= 0.00000000719051(C)

Beräkning för C Quicksort 10000 tal:
C * 10000^2 = 55(ms)
C = 55(ms) / 10000^2 ~= 0.00000055(C)

Beräkning för T(n) Quicksort 100 000 tal:
(55(ms) / 10000^2) * 100 000^2 = 5500(ms)

Beräkning för T(n) Quicksort 1 000 000 tal:
(55(ms) / 10000^2) * 1 000 000^2 = 550000(ms)

Heapsort beräkningar

Beräkning för C Heap sort 1000 tal:
C * (1000 * log(1000)) = 0.081485(ms)
C = 0.081485(ms) / (1000 * log(1000)) ~= 0.00002716(C)

Beräkning för C Heap sort 10000 tal:
C * (10000 * log(10000)) = 0.93908(ms)
C = 0.93908(ms) / (10000 * log(10000)) ~= 0.00002347(C)

Beräkning för T(n) Heap sort 100 000 tal:
0.93908(ms) / (10000 * log(10000)) * (100 000 * log(100 000)) = 11.7385(ms)

Beräkning för T(n) Heap sort 1 000 000 tal:
0.93908(ms) / (10000 * log(10000)) * (1 000 000 * log(1 000 000)) = 140.862(ms)

Tabell för jämförelse av resultat

Algorithm	Size(n)	Time in milliseconds (Runtime, ~Average) T(n)	Calculated time in milliseconds T(n)
Insertion sort	1000	0.743994	N/A
Insertion sort	10000	65.8132	N/A
Insertion sort	100000	8019.2	6581.32
Quicksort	1000	1.3897	N/A
Quicksort	10000	55.34	N/A
Quicksort	100000	22.292	5500 17.37125
Heap sort	1000	0.081485	N/A
Heap sort	10000	0.93908	N/A
Heap sort	100000	25.1268	11.7385

Som man ser från tabellen så skiljer sig det rödmarkerade 5500 väldigt mycket från resultatet från runtime. Det här gäller Quicksort vid en storlek på hundra tusen, anledning till den här stora skillnaden är användning av worst case scenario Ø(N^2). Skulle det ändras till Average case (Expected, n log n) så skulle beräkningen sett ut så här,
C * (1000 * log(1000)) = 1.3897(ms) Ekvation
C = 1.3897(ms) / (1000 * log(1000)) ~= 0.00046233333333333(C) Bryt ut C
1.3897(ms) / (10000 * log(10000)) * (100 000 * log(100 000)) = 17.37125(ms) Räkna tiden
Här ser vi att (n log n) gav ett mer precist svar för beräkning av runtime.

Beräkning av konstanten C
Beräkningen av C, beräknas mha av **T(n) = c * n**. genom att sedan lösa ut värdet på c, så får vi konstanten. För att denna funktion skall fungera optimalt krävs en del villkor. Främst att för olika värden på n, så anpassas konstanten. Exempelvis om vi skall beräkna tidskomplexiteten, övre eller undre gräns(omega) för 5 element, så kommer resultatet att anpassa sig. Ex för beräkning av insertionsort, har man små värden på n, så kommer resultatet att vara en längre tid jämfört med quicksort eller heapsort, däremot vid höga värden på n, så kommer insertion resultatet vara sämre.

Jämförelse sorterade och osorterade data

Algorithm	Sorterad T(n) ms	Osorterad T(n) ms
Insertion sort	**26.129**	**1927.25**
Quick sort	**51.3016**	**11.8602**
Heap sort	**9.64562**	**10.2804**

Insertion sort är väldigt effektiv på både mindre mängder data samt på data som är nästan sorterad, då insertion sort har potential för O(N) och i värsta fall O(N^2). I tabellen kan vi se hur sorterad data v.s osorterad data kraftigt påverkar Insertion sorts tidskomplexitet.

I en sorterad lista väljer Quicksort vanligtvis det första elementet som ett pivot element. När listan redan är sorterad kommer detta att leda till att listan delas upp i en tom vänster lista och en höger lista som innehåller n-1-element. Detta kommer ske för varje rekursion åberopas Quicksort gör vilket kommer att leda till O(n^2) på osorterade listor.

Heapsort hade ingen märkvärdig effekt vid de tester som kördes som uppvisas i tabellerna.