Laboration 1 – Algoritmer och datastrukturer

Studie 1 (Shuffle) ↓

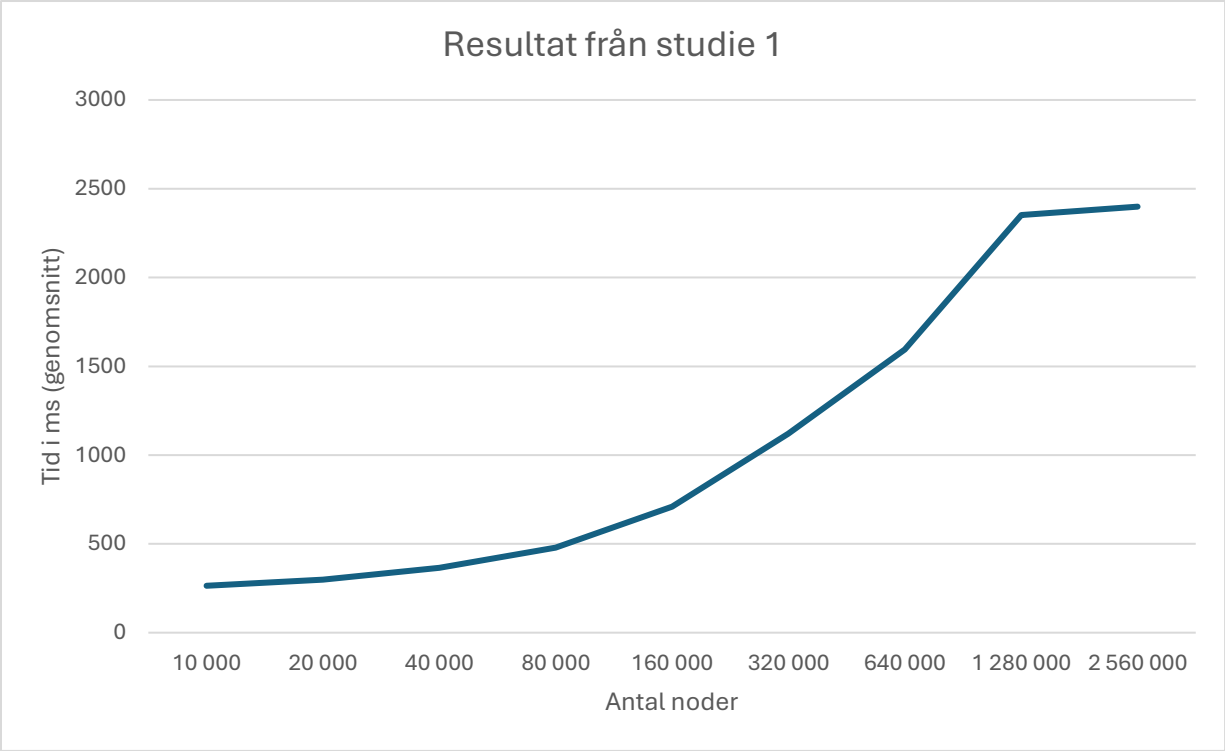
N	Tid i ms (genomsnitt)
10 000	264
20 000	298
40 000	366
80 000	478
160 000	710
320 000	1 121
640 000	1 594
1 280 000	2 351
2 560 000	2 399

Studie 2 (Sorterad) ↓

N	Tid i ms (genomsnitt)
10 000	32 418
20 000	64 848

Uppgift 2

1. Enligt teorin bör ett binärt sökträd med slumpade tal ha en balanserad struktur och höjden växer logaritmiskt, alltså **O(log N)**. När antal noder fördubblas ökas höjden med ungefär +1. Det är även därför tiden ökar lite även fast antal noder ökar drastiskt, vilket visar på att de troligtvis stämmer med teorin. På grund av detta bör tidskomplexiteten vara just **O(log N)**.
2. För att få en tydligare bild över tidskomplexiteten är de lämpligt att stoppa in värdena i en graf för att se en trend och kunna visualisera hur tiden ökar med noderna. Detta gör man bästa genom att plotta exekveringstiden som funktion av N. Hur denna graf ser ut visas i figur 1.



Figur 1 Resultat från studie 1 i en graf

Som grafen visar så växer den snabbt i början och i mitten men planar ut mot slutet som är karaktäristiskt för log(N).

3. Exekveringstiderna för studie 2 när listan är sorterad skiljer sig på de sättet att tiden blir betydligt mycket långsammare för ett samma antal noder. Vid 10 000 noder blir exekveringstiden 246 ms för osorterad lista och 32 418 ms för sorterad lista, detta är mer 100 gånger långsammare. Vid dubbelt så många noder dubbleras exekveringstiden till 64 848 vilket tyder på en linjär ökning. Detta på grund av vid en sorterad lista kommer trädet att bli väldigt obalanserat och alla noder kommer att hamna på samma sida, vilket innebär att vid en dubblering av noderna kommer trädet att bli dubbelt så högt då höjden kommer bli ungefär samma som antal noder. Detta till skillnad från studie 1 där trädet blir mycket mer balanserad och tidskomplexiteten är **O(log N)** och inte linjär vilket förklarar den betydligt mycket högre exekveringstiden i studie 2.