Laboration 1 – Empiriska Studier BST

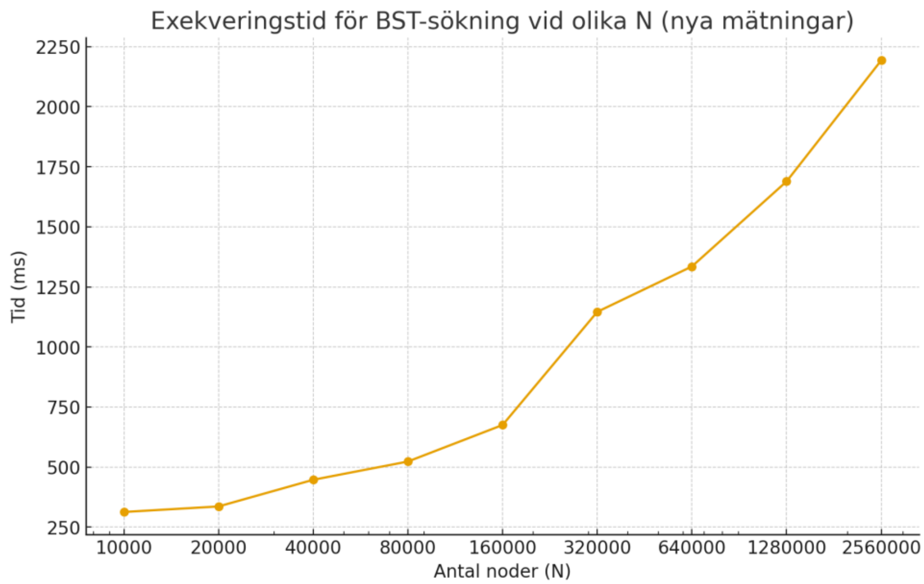
Studie 1 – Osorterade (Shuffle)

N (antal noder)	**T(N) [ms]**	**Kvot T(N)/T(N/2)**
10 000	313	–
20 000	336	1,07
40 000	447	1,33
80 000	523	1,17
160 000	675	1,29
320 000	1 146	1,70
640 000	1 335	1,16
1 280 000	1 689	1,27
2 560 000	2 193	1,30

Studie 2 – Sorterade data

Graf – Studie

N (antal noder)	**Tid (ms, medel)**
10 000	31 474
20 000	66 617



Svar på uppgifterna

1. Teorin säger att om man stoppar in slumpmässiga värden i ett BST brukar trädet bli ganska välbalanserat av sig självt. Det innebär att höjden växer ungefär som log(N), och då borde också exekveringstiden för sökningar och insättningar följa samma mönster, alltså **O (log N)**. När man jämför det här med de faktiska tiderna i tabellen ser man att tiden inte ökar särskilt mycket trots att vi fördubblar antalet element i trädet. Kvoterna hamnar runt **1,1 till 1,5**, vilket är helt rimligt om tillväxten verkligen är logaritmisk.

   Kort sagt: resultaten ser ut att stämma väldigt bra med teorin. Tiderna ökar långsamt på ett sätt som passar ett BST som är ungefär log(N) högt, vilket är precis vad man förväntar sig när värdena är slumpade.

2. Det kan vara svårt att verkligen se sambandet bara genom tabeller och kvoter, så ett bättre sätt är att rita upp all data i en graf. Om man låter x-axeln visa antalet noder och y-axeln visa exekveringstiden blir det mycket tydligare hur tiden faktiskt växer.

   Ett annat knep är att använda en logaritmisk skala på x-axeln, eller att helt enkelt plotta tiden direkt mot **log(N)**. Om kurvan då blir nästan rak eller växer väldigt långsamt, är det ett starkt tecken på att tidskomplexiteten är ungefär **O(log N)**.

Med andra ord: genom att visualisera data i en graf, gärna med log-skala, blir det betydligt enklare att se vilket asymptotiskt samband man har att göra med.

3. I studie 2 ser vi att exekveringstiderna blir betydligt längre än i studie 1. Förklaringen är att all data sorteras innan trädet byggs. När man stoppar in redan sorterade värden i ett vanligt BST hamnar varje nytt värde längst till höger (eller vänster), vilket gör att trädet i princip blir en lång kedja i stället för ett riktigt träd.

   Det betyder att höjden växer ungefär som N, och då blir också sökningar och insättningar **O(N)** i stället för **O (log N**). Det märks tydligt på mätningarna: tiden ökar nästan proportionellt med antalet element och ungefär fördubblas när N fördubblas. Det är ett klassiskt mönster för linjär tidsökning.