Algoritmer och Datastrukturer Lab 2

Olof Zetterstrand

olof.z1337@gmail.com

Uppgift 1

1. Generiska typer är inte kovarianta. För att koden ska kompilera behöver den generiska typen extenda number istället.
2. Första delen deklarerar typparametern T, andra delen säger att T inte kan var vad som helt utan måste implementera eller ärva interfacet Comparable, sista delen är ett lower bound wildcard som säger att den ska kunna jämföras med sig själv eller superklasser.
3. PECS Rule (get and put principle) star för Producer Extends, Consumer Super. Detta är en minnesregel för när man ska använda upper och lowerbound wildcards, extend (<? Extends T> för in-argument som ger data till metoden och super (<? Super T>) för ut-argument som tar emot resultat. Inget wildcard behövs när det ör både in och ut-argument och man ska aldrig använda wildcards för returvärden.

Uppgift 2

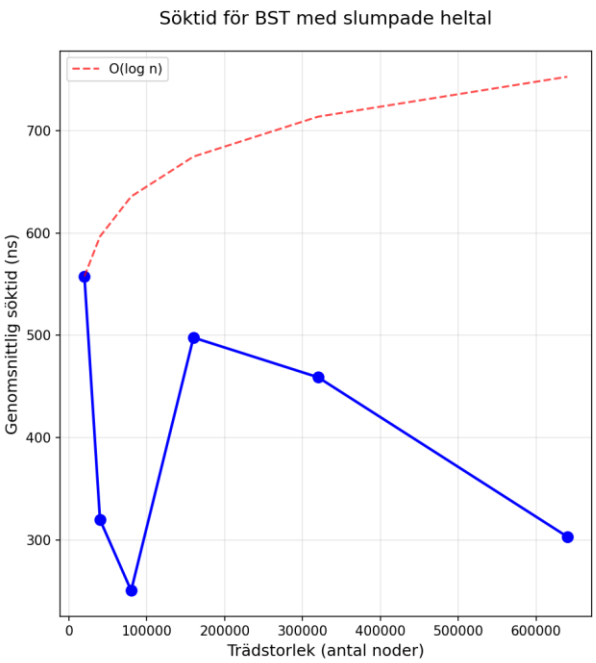
1. Ett binärt sökträd med BinarySearchTree och inre klassen BSTNode. Iterativa metoder för add och searchFor. Varje nod har ett värde och två barn-referenser. Size-räknare uppdateras vid ändringar.
2. Ignorerar dubbletter vid add.
3. Eftersom add ignorerar dubbletter behöver remove bara ta bort en nod per värde.

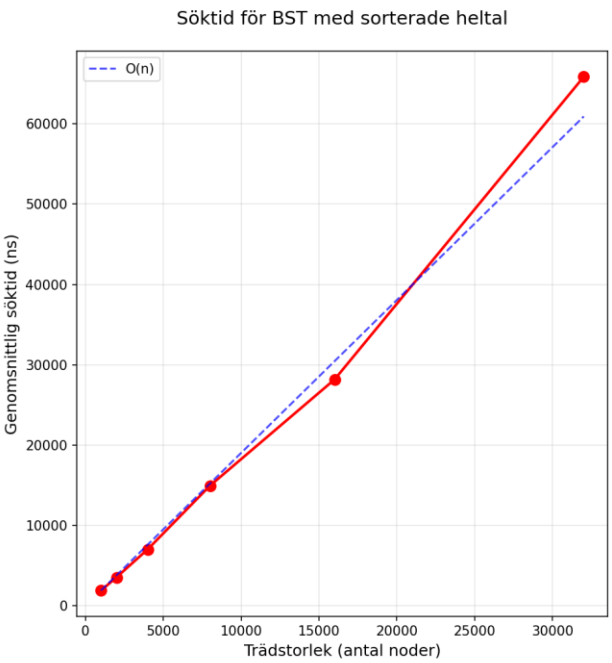
Uppgift 3

För slumpade heltal är söktiden ganska konstant oavsett trädstorlek, vilket bekräftar O(log n) tidskomplexitet för balanserade BST.

För sorterade heltal ökar söktiden linjärt och fördubblad trädstorlek ger fördubblad söktid vilket bekräftar O(n) tidskomplexitet.

Tidskomplexiteten avgörs genom hur söktiden förändras med trädstorleken, konstant eller långsamt ökande för O(log n), linjärt ökande för O(n).


Söktid för BST med slumpade heltal
O(log n)
Genomsnittlig söktid (ns)
Trädstorlek (antal noder)
300
400
500
600
700
0
100000
200000
300000
400000
500000
600000


Söktid för BST med sorterade heltal
O(n)
Genomsnittlig söktid (ns)
Trädstorlek (antal noder)
0
10000
20000
30000
40000
50000
60000
0
5000
10000
15000
20000
25000
30000