TDA416 VT16 Grupp 4
Emil Axelsson, 951114-2854, emilaxe@student.chalmers.se
Michelle Tran Luu, 940124-0024, luum@student.chalmers.se

Datastrukturer och Algoritmer

Laboration 3

Tio uppmättna värden för n till SortedLinkedList:

n = 10	17.210326784302787ns
n = 50	17.340557204550528ns
n = 100	18.094772613221735ns
n = 200	16.96298674044444ns
n = 300	17.74331119462881ns
n = 500	19.190903622846108ns
n = 600	17.967052643457887ns
n = 700	18.389986165611205ns
n = 900	17.685747601121992ns
n=1000	19.866789196675693ns

Tio uppmättna värden för SplayTreeSet

n = 1000	252.29833386497592ns
n = 2000	300.69050319160755ns
n = 3000	336.25047089966245ns
n = 4000	353.5662413029391ns
n = 5000	371.5388007930611ns
n = 6000	439.7435655905906ns
n = 7000	435.6310004793331ns
n = 8000	546.423005719956ns
n = 9000	440.15812941141866ns

Emil Axelsson, 951114-2854, emilaxe@student.chalmers.se
Michelle Tran Luu, 940124-0024, luum@student.chalmers.se

n=10000	484.3467783072084ns

TDA416 VT16 Grupp 4
Emil Axelsson, 951114-2854, emilaxe@student.chalmers.se
Michelle Tran Luu, 940124-0024, luum@student.chalmers.se

Tidskomplexitet för SortedLinkedListSet

Add

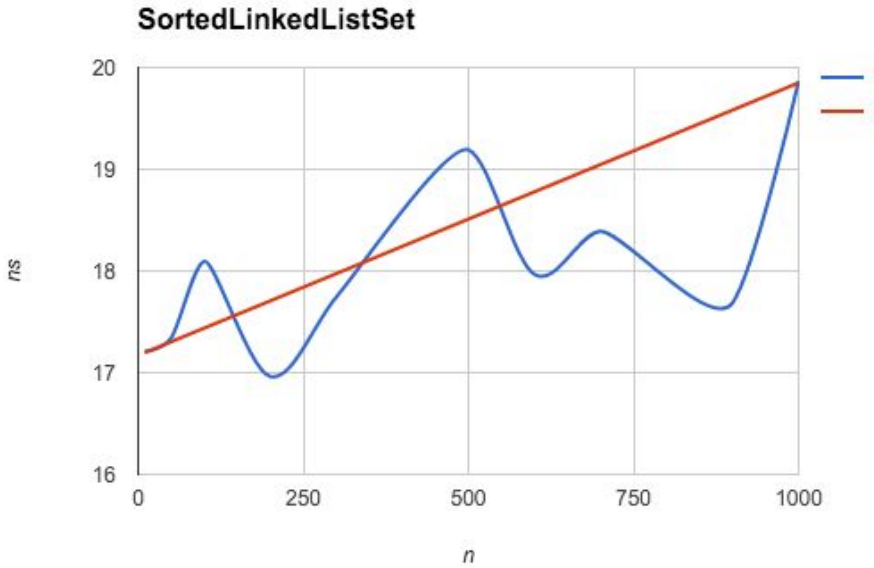
Metoden add kallar på metoden “addrec”. Den går igenom elementen för att se vart det nya elementet ska läggas in. Om man har “otur” så är det elementet man vill lägga till större än alla andra element i listan. Detta innebär att man måste gå igenom alla noder i listan (n st), vilket ger komplexiteten O(n).

Remove

Metoden remove kallar också på en rekursiv metod, “removeRec”. Den metoden används för att hitta den nod som ska tas bort. På samma sätt som i add så kan det element vi vill ta bort ligga längst bak i listan, vilket betyder att vi måste gå igenom alla noder innan vi kan ta bort det. Detta ger komplexiteten O(n).

Contains

Även denna metod har en rekursiv metod som den kallar på, “containsRec”. Även denna har komplexiteten O(n), då den i värsta fall kan tvingas gå igenom alla element innan den hittar elementet den letar efter.



TDA416 VT16 Grupp 4
Emil Axelsson, 951114-2854, emilaxe@student.chalmers.se
Michelle Tran Luu, 940124-0024, luum@student.chalmers.se

Tidskomplexitet för SplayTreeSet

Add

Sökning i ett binärt söktträd fungerar på så sätt att vid varje nod man hamnar i, kommer man antingen hamna på den vänstra barnnoden eller den högra barnnoden. Det gör så att om man väljer den vänstra barnnoden så kommer den högra barnnoden och dess träd att tas bort från sökningen.

Detta ger att första iterationen är O(1) och efter andra steget som är att välja vänster- eller högerbarnnod kommer enbart halva trädets storlek, *n* att återstå. Detta beskrivs som T(n/2). Denna processen kommer fortsättas tills man har hittat en lämplig position till den nya noden och i värsta fall kan det vara det vara noden som ligger längst ner, höjd *k*. Det kan beskrivas som $T((n/2)^2)$.

Remove

Remove fungerar som i add-metoden vid sökning efter den noden som ska tas bort. Detta ger oss också $T((n/2)^2)$.

Contain

Även här sker det en rekursiv sökning efter en nod och i värsta fall kan den noden som söks ligga längst ner på trädet och då ger det oss således $T((n/2)^2)$.