Lab 2 TDA416 VT16
Grupp 47
Pontus Eriksson, 940410-7378, ponerik@student.chalmers.se
Robin Jansson, 870609-5695, robin.seriousmail@gmail.com

Antal noder kommer i rapporten kallas för n

Varför ger inte Java API:s LinkedList en förbättrad komplexitet?

Med användning av LinkedList vid uppdatering av element används get-metoder vid varje användning av element, vilket gör att vi ändå måste iterera över listan för att hitta elementen.
I vår implementation räcker det att fråga om grannens nod, vilket är snabbare.

Tidskomplexitet för algoritm A

Första for-loopen körs n gånger.

While-satsen körs n-k gånger, distance uppdateras n-2 gånger.
k konstant.
$n + (n-k)*(n-2) + k \approx$
$\approx n + n^2 + k \approx$
$\approx n^2$

Tidskomplexiteten är alltså $O(n^2)$.

Tidskomplexitet för algoritm B

Första for-loopen körs n gånger.
While-satsen körs inte för första och sista elementet.
PriorityQueue's funktion poll och add har komplexitet log(n)
k konstant.
$n + (n-2) + (n-k)*(n*2+3*\log(n)) + k \approx$
$\approx 2n^2 + 3n*\log(n) + k \approx$
$\approx 2n^2 \approx$
$\approx n^2$

Tidskomplexiteten är alltså $O(n^2)$.

Orsaken till att komplexiteten blir samma är att PriorityQueue har komplexitet n för att ta bort element.
Genom att implementera en egen kö där objekt kan uppdateras utan att tas bort och läggas till så hade komplexiteten minskat märkbart.