Uppgift 2 a

Komplexiteten för algoritm a är $O(n^2)$. Detta då vi har en for-loop innanför en while-loop som går igenom hela listan. n är antalet element i listan vi har som parameter. For-loopen innuti while-loopen görs (n-4)/2 gånger, när vi tar bort en nod i listan så är detta O(n). Det sker varje varv i while-loopen. While-loopen gör vi (n-2)/2. Alltså får vi komplexiteten för hela metoden $O(n^2)$.

Uppgift 2 b

Om vi i en LinkedList letar efter ett element med ett index som är närmare sista indexet än index 0 så börjar LinkedList att leta i slutet av listan, annars börjar den vid index 0. Ju närmare mitten av listan man kommer desto längre tid tar det att hämta det efterfrågade elementet.
En vanlig array fungerar istället med pekare som pekar på första elementet, vill man åt index 5 så läggs detta på pekarens adress och hämtar direkt det som finns på den minnesadressen. Detta tar konstant tid.

Metoden remove(Object) går på linjär tid, O(n). Detta görs varje varv i while-loopen. Sedan är det while-loopen som görs ett antal gånger baserat på metodens parametrar. I värta fallet görs while-loopen också här (n-2)/2 gånger. Därmed tillhör också denna metod (Lab2b) $O(n^2)$. När vi lägger in elementen i PriorityQueue (som ligger utanför while-loopen) så tar detta (nlog(n)-2)/2 tid.
Konverteringen från array till en doubly linked list och sen tillbaka till en array har linjär komplexitet. Om vi har en stor array och ett litet k (metodens parameter) så kommer det ta längre tid än om vi har ett k nära antalet punkter i arrayen. Om man kan göra den rekursiv skulle man kunna få ner den till en O(log(n)) komplexitet.