**Oppg. 1b)**

Kjøretiden til "push_front()" vil worst case være O(n). Dette er fordi innsetting på indeks 0 krever at alle andre elementer i listen flyttes ett steg lenger frem,
f.eks. [0]<-[1], [1]<-[2] osv. Derfor vil kjøretiden avhenge av hvor mange elementer listen inneholder.

Kjøretid til "push_back()" vil være O(1). Her aksesserer vi bakerste indeks, og setter inn det nye elementet på indeksen etter dette. Vi trenger ikke iterere gjennom hele
listen, og tiden vil derfor være uavhengig av antall elementer.

Kjøretid for "push_middle()" vil være O(n). Dette fordi vi aksesserer midterste indeks, setter inn det nye elementet, og forskyver alt etter dette en plass lenger
bak, som i push_front(). Kjøretiden vil da bli O(1/2n), som forkortes til O(n).

Kjøretiden til get() er O(1). Her vet vi hvilken indeks elementet vi skal hente ut ligger på, og kan gå rett til denne plassen.
Derfor er ikke kjøretiden avhengig av hvor mange elementer listen inneholder, og vi får konstant tid.

**Oppg. 1c)**

Om vi vet at N er begrenset, for eksempel N = 100, så vil O-notasjonen bli O(100), som forenkles til O(1). Dermed blir kjøretiden konstant. Når N er begrenset, så kan ikke N bli høyere enn denne gitte verdien, og kjøretiden vil heller aldri bli større enn den gitte verdien til N.

**Oppg. 2)**

Kompleksiteten for binærsøk i en lenket liste er O(n*log(n)). Dette er fordi et normalt søk i en lenket liste har kompleksitet O(n), mens et binærsøk har kompleksitet O(log(n)). Søket må traversere gjennom listen n ganger, for å hente hvert element. Dette gjennomføres log(n) ganger, siden det er et binærsøk, og lagt sammen utgjør disse en kompleksitet O(n*log(n)) i worst case. Om det brukes en annen datastruktur, for eksempel Array eller ArrayList, som støtter indeksering, vil ikke listen måtte traverseres gjennom for hver log(n), og kjøretiden vil bli O(log(n)), som er binærsøkets kjøretid alene.