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, vil vi kunne utelukke alle utfall > n. Noen algoritmer er raske på kort sikt, men tregere på lang sikt. Om man har et begrenset område å se på
N innenfor, vil valget av algoritme kunne være annerledes enn om vi skulle tatt hensyn til en uendelig stor N. Dette fordi vi slipper å ta hensyn til N større enn begrensningen,
og dermed har et mer nøyaktig bilde å gå ut ifra. Det kan også påvirke valget av beholder for dataene. Et binærtre har for eksempel kjøretid O(log2(n)), mens en lenket liste har kjøretid O(n) på henting av data. Dermed vil en gitt mengde n kunne gjøre det gunstig å velge lenket liste til oppbevaring fremfor et binærtre. I motsatt fall, med en høyere kjent n, vil det kunne være vesentlig raskere å velge et binærtre til lagring.

Oppg. 2)

I utgangspunktet vil kjøretiden for en binærsøk-algoritme være O(log2(n)). Dette er fordi hvert steg i algoritmen fører til en halvering av søkeområdet, og dermed en halvering av økt kjøretid for hvert element som ikke er det du leter etter. Utviklingen for kjøretidskompleksiteten vil være n, n/2, n/4, n/8 osv. Kjøretiden i algoritmen vi har her vil imidlertid være O(n), siden get() i lenkelister ikke kan slå opp på indekser på samme måte som for eksempel Arrays, og derfor må gå gjennom hvert element for å komme til ønsket indeks. Dermed blir worst-case estimatet for binærsøk med lenkede lister O(n).

Om man heller velger en annen datastruktur, for eksempel Array eller ArrayList, vil get() ha kjøretid O(1). Dermed vil binærsøkets O(log2(n)) være "tregeste" operasjon, og kjøretiden for algoritmen blir O(log2(n)), fremfor lenkelistens O(n).