Tidsforbruk for `lagHeapOrdning`

I denne delen fikk jeg hjelp av AI til å forstå og formidle hvordan dette beregnes. Her er tidsforbruket (O-notasjon) for `lagHeapOrdning` for de to tilfellene, basert på noder i treet:

1. Treet er komplett:

- **Tidsforbruk:** O(n)
- **Kort forklaring:** I et komplett (buskete) tre er de fleste nodene nær bunnen (blader eller nesten blader). `reparer`-metoden, som kalles for hver node, gjør derfor veldig lite arbeid for de fleste nodene. Selv om noen få noder øverst krever mer arbeid, er det så mange noder nederst som krever nesten ingen arbeid, at det er disse som har mest å si for totaltiden. Totalt blir arbeidet proporsjonalt med antall noder, n, fordi de aller fleste nodene (de nederste) er veldig raske å håndtere.

2. Treet er degenerert (hver node har maks ett barn, som en lenket liste):

- **Tidsforbruk:** O(n2)
- **Kort forklaring:** I et degenerert (dypt og tynt) tre er høyden nesten n. Når `reparer` kalles på en node høyt oppe, kan den måtte jobbe seg ned mange nivåer (opptil n−1 for roten). Siden `reparer` kalles for *alle* noder, må vi summere arbeidet: Noden nederst krever 0 steg, neste 1 steg, neste 2 steg, ..., opp til roten som krever ca. n−1 steg. Summen av 0+1+2+⋯+(n−1) er omtrent n2/2. Totalt blir arbeidet proporsjonalt med n2.