Algdat Excercise 3

Tomas Beranek og Callum Gran

September 2022

Løsningen vi hovedsakelig presenterer er quicksorten som står i boka og dual-pivot fra geeksforgeeks med endringene du har foreslått. Om det er mindre enn 50 elementer i en partition, så byttes algoritmen til insettings sortering. Noen C optimaliseringer er gjort til algoritmene, dette tror vi også har påvirket resultatet. Dataen blir også testet om den er sortert før og etter algoritmen, samt sjekksummen. Det som er verdt å merke er at med single thread varianten så kan du få feil på de første gjennomgangene av arrayet om du bruker O3 og Ofast med GCC under compilering.

Tidtaking ble gjort med clock_gettime(), dette er en relativt nøyaktig metode, men ikke like nøyaktig som CPU klokka. Grunnen til at dette er brukt er fordi det også er en multithreaded variant. Om man var til å se på CPU klokken med multithreading vil man få en vilt forskjellig tid enn det som faktisk ble brukt. For å få den mer nøyaktig har vi brukt MONOTONIC_RAW for at den fortsatt skal se på hardware time uten justeringer av NTP eller adjtime.

På de neste sidene ser du forskjellige tider brukt på forskjellige mengder tall i et grafisk felt. Målingene ble gjort med optimaliserings nivå O2 i GCC compiler mens pcen var plugget i strøm (dvs maks CPU kraft på laptopen). Vi velger også kun å vise dataen som er etterspurt i oppgaven, men om du er interessert i den andre dataen så skrives den ut av programmet på en pen og leselig måte.
Y-aksen: Tid i millisekunder.
X-aksen: Antall elementer i millioner.

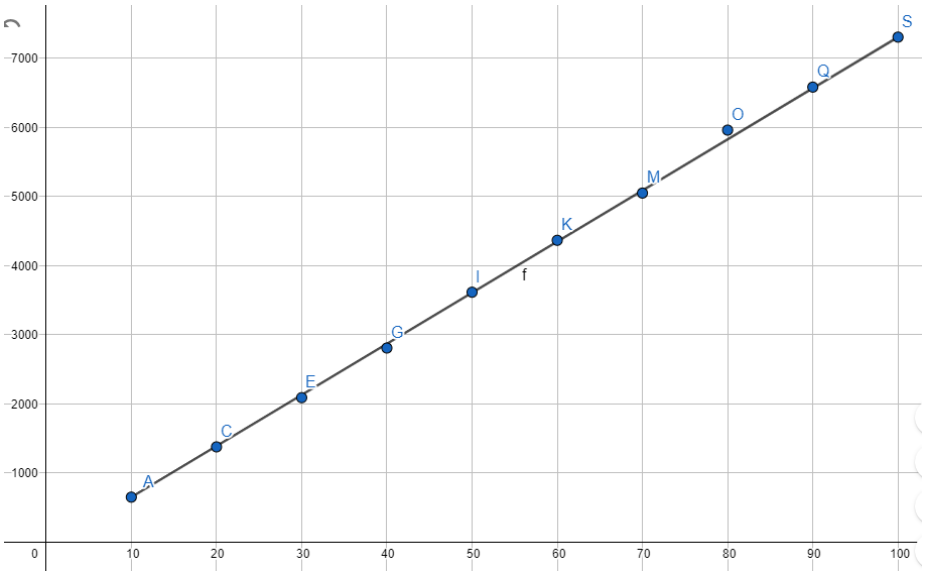

Figure 1: Bilde viser tidskompleksiteten til single pivot med tilfeldige tall

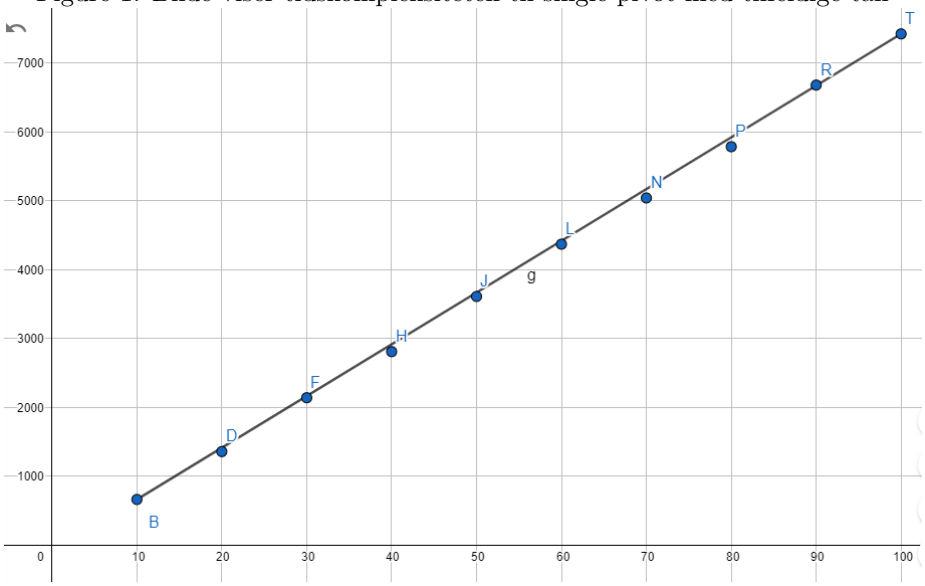

Figure 2: Bilde viser tidskompleksiteten til dual pivot med tilfeldige tall

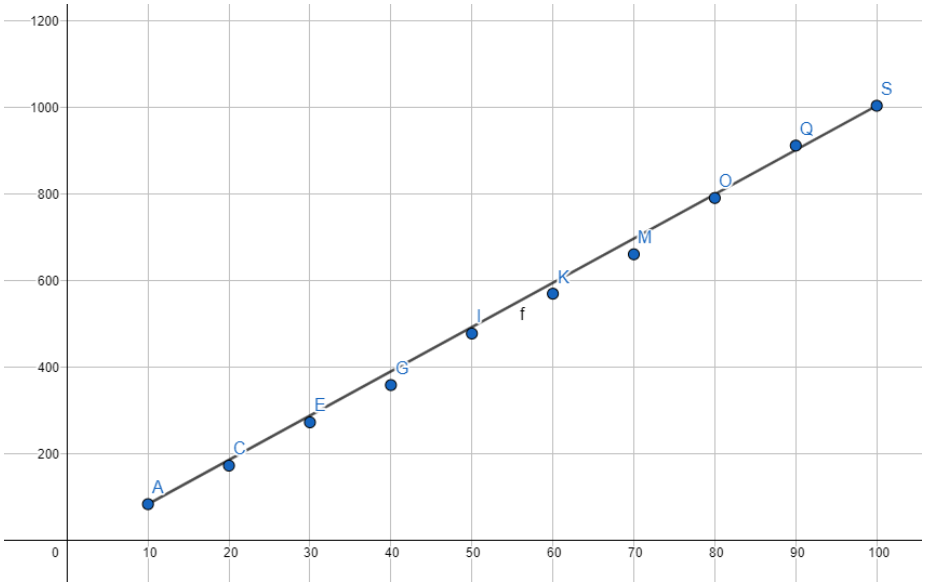

Figure 3: Bilde viser tidskompleksiteten til single pivot med sortert tall

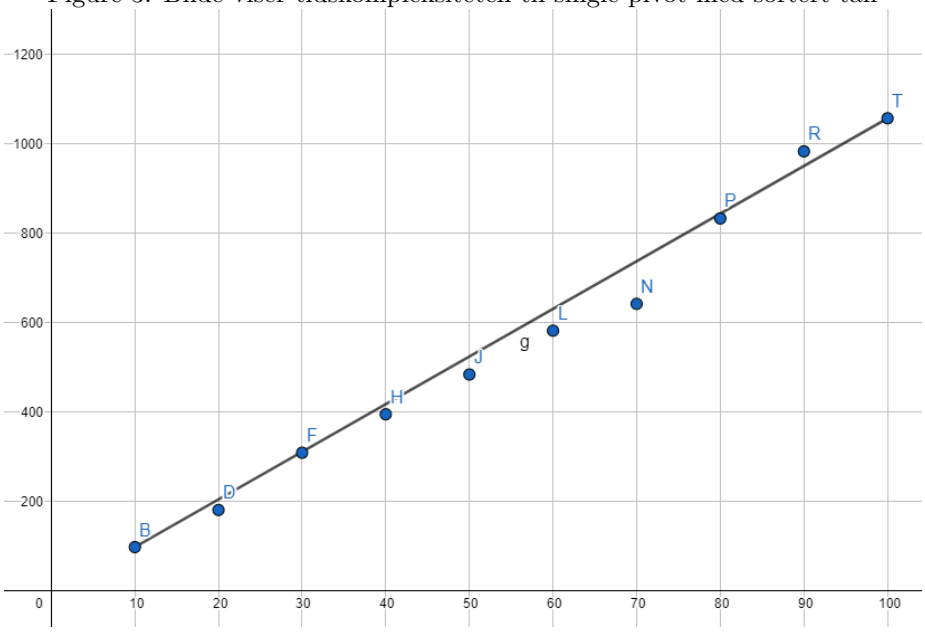

Figure 4: Bilde viser tidskompleksiteten til dual pivot med sortert tall

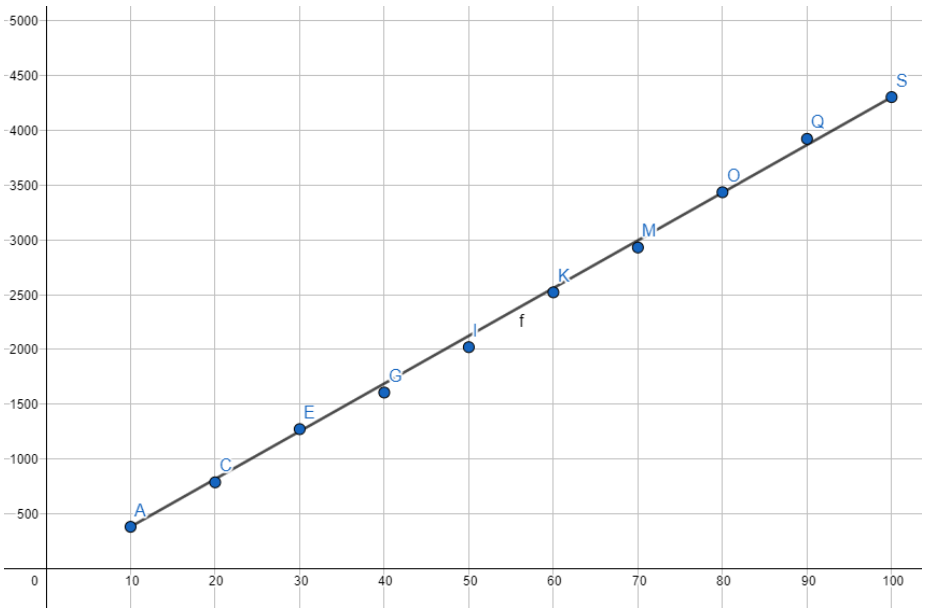

Figure 5: Bilde viser tidskompleksiteten til single pivot med dupliserte tall

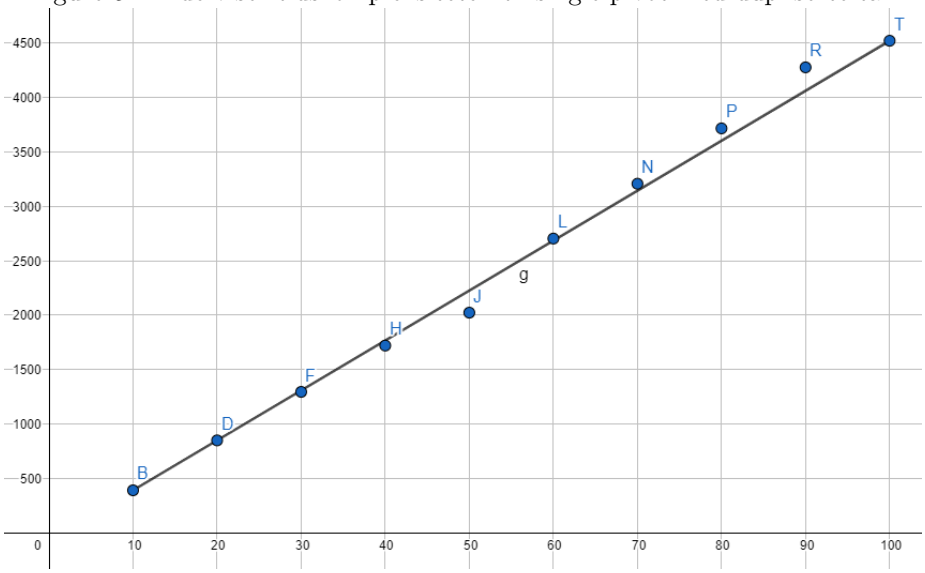

Figure 6: Bilde viser tidskompleksiteten til dual pivot med dupliserte tall

Resultatene over viser at begge løsningene er så og si lineære for alle cases, slik etterspurt. I tillegg det er svært liten differanse mellom algoritmene, men single pivot litt raskere. Dette bryter med det vi hadde sett for oss, i følge geeks for geeks skal dual- være litt raskere enn single pivot. Vi har konkludert at det er flere grunner til dette:
1. Vi bruker preprocessor macroer for å kjøre enkle operasjoner som swap, median3sort og allokering av minne. Dette gjør at de utføres raskere fordi du slipper overheaden som kreves for å kalle en funksjon. Siden hele median3sort er en macro så sparer single pivot ovenfor dual pivot, selv om dual mid point blir funnet ved en macro. I tillegg kjøres bitwise shifting for å dele på 2, og ikke for å dele på 3.
2. Kompileringsmetoden vi brukte var gcc -O2. Siden det er et optimaliserings nivå, så kan det hende at koden blir kompilert mer optimalt for single enn dual.
3. PCen er rask og har raskt minne. Dual pivot bruker x/3. Å dele er generelt sett en treig operasjon i alle programmerings språk, C inkludert. Siden det ikke er noen lett måte å dele på 3 med bitshifting. Når man har mange tall så blir denne operasjonen kalt mange ganger. Da blir deling på 3, etter det vi har konkludert en større knekk for programmet enn antatt. Vi har i etterkant sjekket programmet når pcen er plugget ut av strøm, dvs at cpu ikke kan kjøre på turbo 4.2GHz men heller 1.8GHz. Da så vi at dual pivot var raskere.

I tillegg tenkte vi det var kult å prøve oss på multithreading. Vi har en versjon av programmet som bruker dette også. Her kaller vi kun nye threads når det trengs, dvs når det er mer enn 150000 elementer i "arrayet". Dette er fordi å kalle på en ny thread er en krevende operasjon og ikke egner seg når det er mindre mengder med tall. Laptopen min kjører med 16 threads og minne med hastighet på 4266MHz. Det betyr klart at programmet kjører fort, men det som var interessant å se her var at forskjellen i hastighet oppførte seg likt som single threaded. Samtidig så vi at jo lavere tallene man har (de tar mindre bytes), så kjører programmet fortere, selv om dette skal være en dårlig case for quicksort. Derfor har vi konkludert at minnehastighet er en av de større knekkene for algoritmene når de er multithreaded.