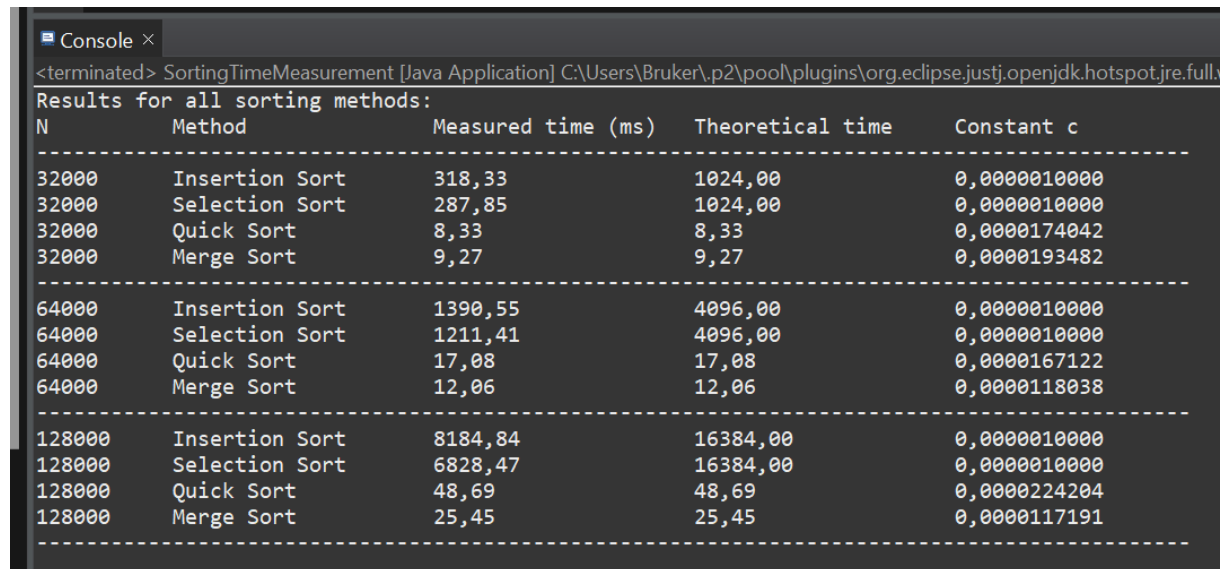


Gruppen består av Elias Daland, Endre Strand Norén, Kristoffer Albrigtsen og Marius Løvereide Borgen

Oppgave 2

a) Resultater for hver sorteringsmetode:



The screenshot shows a Java console window titled "Console x" with the following content:

```
<terminated> SortingTimeMeasurement [Java Application] C:\Users\Bruker\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full\
Results for all sorting methods:
N      Method      Measured time (ms)  Theoretical time    Constant c
-----
32000  Insertion Sort   318,33             1024,00             0,0000010000
32000  Selection Sort   287,85             1024,00             0,0000010000
32000  Quick Sort       8,33               8,33                0,0000174042
32000  Merge Sort       9,27               9,27                0,0000193482
-----
64000  Insertion Sort   1390,55            4096,00             0,0000010000
64000  Selection Sort   1211,41            4096,00             0,0000010000
64000  Quick Sort       17,08              17,08               0,0000167122
64000  Merge Sort       12,06              12,06               0,0000118038
-----
128000 Insertion Sort   8184,84            16384,00            0,0000010000
128000 Selection Sort   6828,47            16384,00            0,0000010000
128000 Quick Sort       48,69              48,69               0,0000224204
128000 Merge Sort       25,45              25,45               0,0000117191
-----
```

Drøfting:

De fire sorteringsalgoritmene — Insertion Sort , Selection Sort , Quick Sort , og Merge Sort — viser ulike mønstre i både teoretisk kompleksitet og praktiske kjøretider.

Insertion Sort og Selection Sort

Disse to har en teoretisk kompleksitet på $O(n^2)$ i gjennomsnitt og viser seg å være mindre effektivt for store datasett. Vi observerer at disse metodene bruker mer tid enn Quick Sort og Merge Sort når antall elementer øker. Eventuelle avvik for forklares ved konstantleddene i tidsuttrykket $T(n) = c * n^2$, som kan variere ut ifra maskinvareegenskaper eller implementasjonsdetaljer.

Quick Sort

Quick Sort har en teoretisk kompleksitet på $O(n \log n)$ i gjennomsnitt og viser seg å være veldig effektivt, spesielt for store datasett. I verste fall kan den degenerere til $O(n^2)$, for eksempel hvis pivot – elementet velges dårlig (som når alle elementene er like). Når datasettet er tilfeldig fordelt stemmer resultatene godt overens med teorien, men det kan oppstå avvik når datastrukturen fører til ubalanserte partisjoner.

Merge Sort

Merge Sort har også en teoretisk kompleksitet på $O(n \log n)$, og denne algoritmen leverer gode resultater både i verste og beste tilfelle. Stabil og forutsigbar, selv om den kan ha større minnebruk sammenlignet med Quick Sort. Kan finne små avvik fra teoretiske verdier, som skyldes overhead knyttet til rekursive funksjonskall.

b) Når alle elementene i tabellen er like, vil Quick Sort ikke fungere gunstig. Dette kan bli en worst case scenario for Quick Sort, der algoritmen ender med en kompleksitet på $O(n^2)$ istedenfor den vanlige $O(n \log n)$.

Når alle elementene er like, vil Quick Sort dele listen på en dårlig måte, noe som fører til mange unødvendige rekursive kall og gjør sorteringen mye tregere. For å unngå dette, kan man bruke en bedre pivot valgsstrategi eller vurdere å bruke en annen sorteringsalgoritme som er mer robust til de spesifikke tilfellene.