Oppgave 1 del 3:

I hvilken grad stemmer kjøretiden overens med kjøretidsanalysene (store O) for de ulike algoritmene?

//Kjøretider for de ulike algoritmene
Quick: Worst case = O(n^2), ofte O(nlog(n)) i praksis
Bubble: O(n^2)
Insertion: O(n^2)
Heap: O(nlog(n))

//Skjermbilder av figurer referert til finnes f.o.m. side 4/via hyperkobling.

Vår oppfatning er at resultatene stort sett stemmer overens med kjøretidsanalysen for algoritmene.
Vi opplever i flere tilfeller at quick sort ser ut til å kjøre i O(nlog(n)) enn worst case O(n^2) (random_/nearly_sorted_10000). Dette er sannsynligvis fordi quick sort (i praksis) vanligvis kjører med denne kompleksiteten, og sjelden når forutsetningene for sitt worst case. Ut over dette virker det generelt sett som resultatene stemmer relativt godt med forventningene.

Hvilke sorteringsalgoritmer utmerker seg positivt når n er veldig liten? Og når n er veldig stor?
Med veldig liten n utmerker insertion og bubble sort seg positivt (se random_/nearly_sorted_10), som de mest effektive algoritmene. Når det gjelder stor n, ser vi at quick og heap sort skiller seg positivt ut fremfor de andre (se random_/nearly_sorted_n > 100). Vi ser riktignok på analysen av nearly_sorted at insertion sort også utmerker seg positivt (nearly_sorted_10000). Dette skyldes sannsynligvis at en nesten sortert liste nærmer seg forutsetningene for beste-scenario for denne algoritmen, og dermed virker positivt inn på kjøretiden.

Hvilke sorteringsalgoritmer utmerker seg positivt for de ulike inputfilene?
For samtlige nearly_sorted-filer ser vi at insertion sort gjør det svært bra. Dette skyldes nok at en nesten sortert mengde tall krever liten grad av videre sortering gjennom innsetting. Dermed vil insertion sort håndtere datamengden effektivt ved å kunne sette de fleste tallene inn ferdig sortert, fremfor å måtte bytte plass på mange elementer i løpt av innsettingen.

På inputfilene med verdi > 10, ser vi at heap og quick sort er svært effektive og overlegne mtp. tid. Motsatt gjelder, som nevnt over, for insertion og bubble sort for n = 10.

Har du noen overraskende funn å rapportere?
Insertion sort fungerer bra uansett størrelse på n, så lenge listen er helt eller nesten sortert. Dette fordi det er få, og/eller små bytter som må finne plass, og dermed forsvinner lite tid til dette.

##########OBS: Grunnet verdier for avbrudd i prekoden ble enkelte algoritmer avbrutt pga. tid ved større n. Dette kan man bl.a. se på random_10000, hvor noen av kurvene stanser før andre, i tillegg til at x-aksen ikke strekker seg helt til verdien av n. Vi ser likevel, til tross

for dette, tendensene i utviklingen, og føler at selv de uferdige resultatene gir oss et god inntrykk av algoritmens kjøretid.############

Oppgave 2:

1)

Din første oppgave er å beregne en øvre grense for hvor mye kabel selskapet trenger. Hva er kostnaden av den dyreste måten å koble sammen alle signaltårn på? Oppgi svaret ditt som et tall.

Dyreste måte: 61.

(T1->T2->T4->T6->T7->T5->T3). Figur.

2)

Hva er den billigste måten å koble sammen alle signaltårn på? Oppgi dette som et tall. Videre, oppgi hvilken algoritme du brukte, samt hvordan du formaliserte problemet. Til slutt, oppgi hvilke andre algoritmer som er blitt dekket i pensum, som man kunne ha brukt i stedet.

Billigste måte: 34.

```
(                          T3         )
(                        /            )
( T1 -> T4 -> T2 -> T5 ---              )    Figur.
(                        \            )
(                          T6 -> T7   )
```

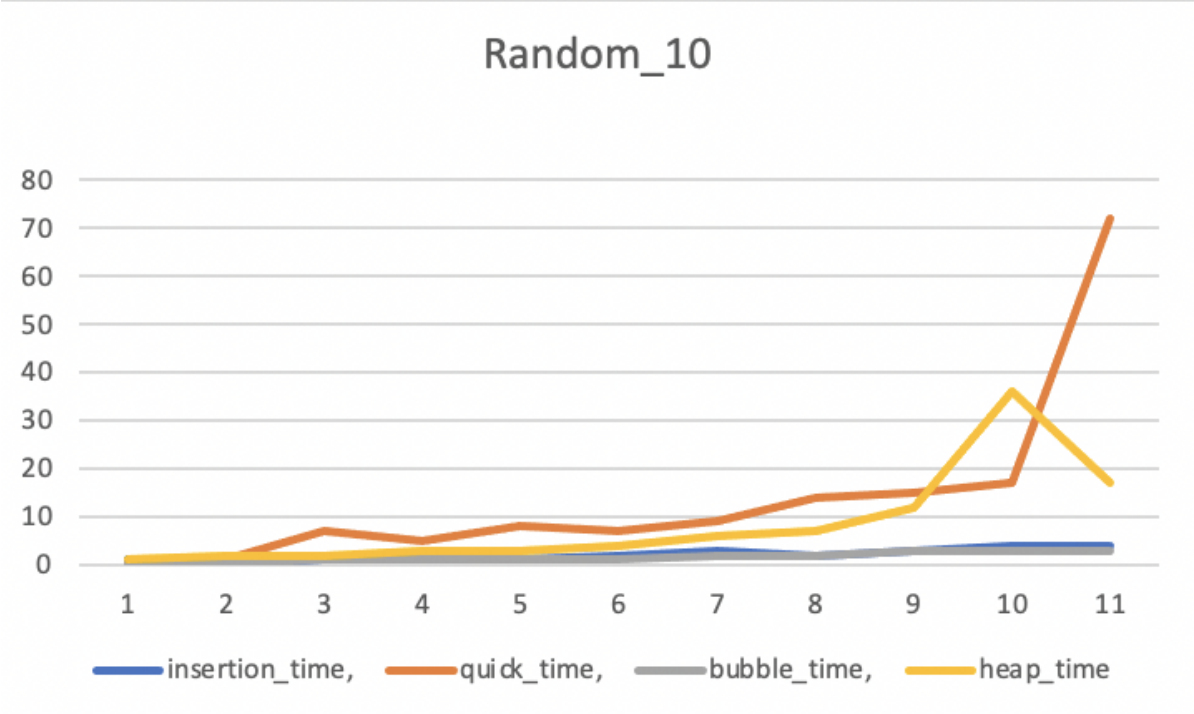
Vi brukte Prims algoritme for å finne billigste vei her. Vi så på grafen som et vektet spenntre, og vurderte det derfor som formålsmessig å bruke Prims algoritme for å løse problemet (se figur "formalisering"). Vi kunne alternativt brukt Kruskal eller Borůvka, men siden vi valgte å tegne opp og løse problemet "manuelt", syntes vi det var enklest å anvende fremgangsmåten i Prims.

Oppgave 3:

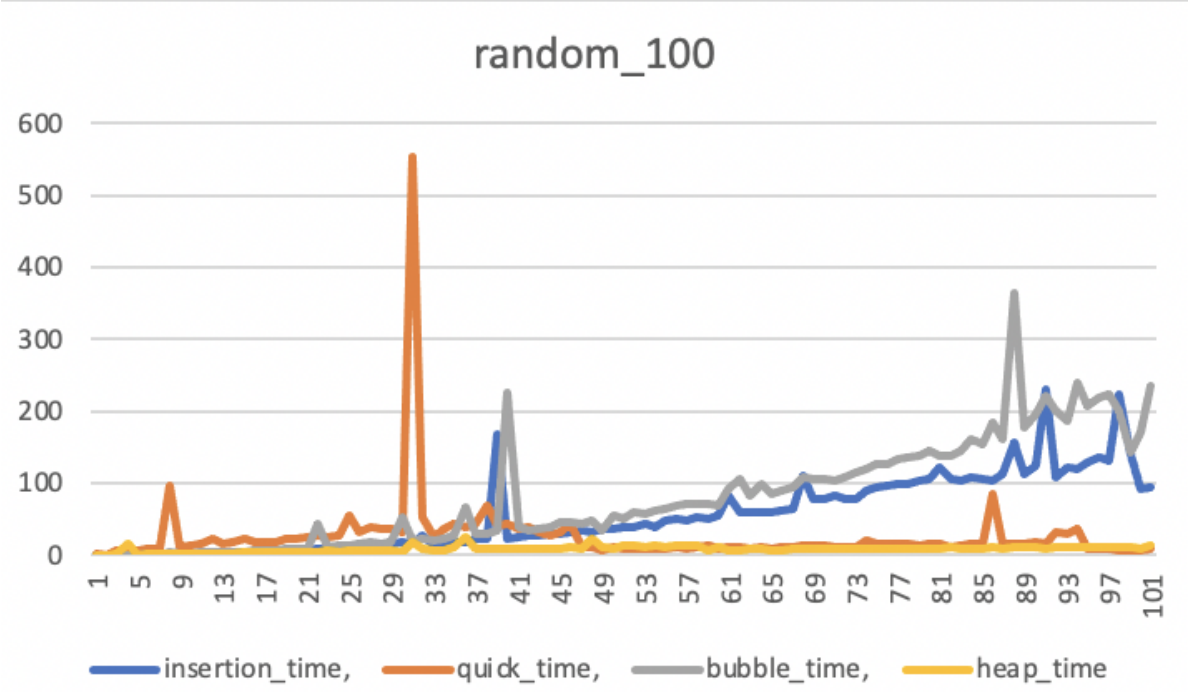
Algoritme 1:
Algoritmen har kjøretid O(1). Tallet som sendes inn, returneres +1, og størrelsen på dette påvirker ikke kjøretiden. Algoritmen vil bruke like lang tid på å kjøre med n=1 som n=1000.

Algoritme 2:
Algoritmen har kjøretid O(m*n). Den første for-løkken itererer m-ganger, mens den neste for-løkken itererer n-ganger for hver m. Inne i disse løkkene kalles det på Algo1, som har kjøretid O(1). Tregeste kjøretid, og dermed tidskompleksitet for Algo2 er O(m*n).

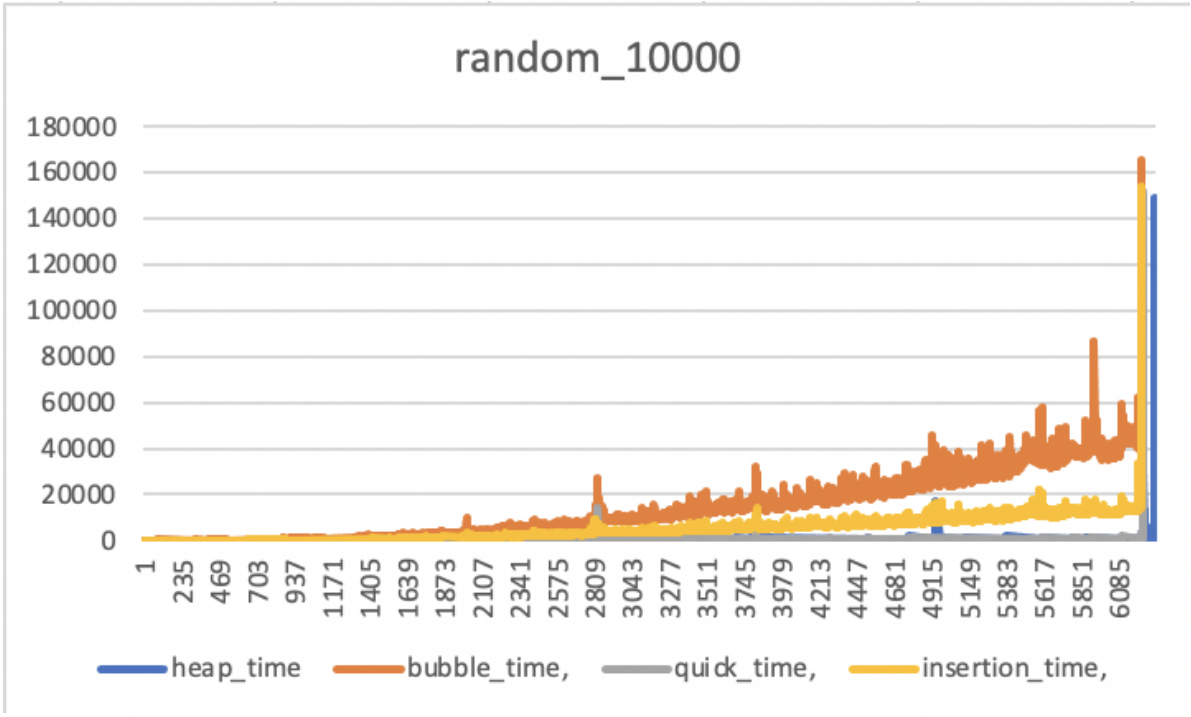
Algoritme 3:
Algoritmen iterer så lenge j > 1. For hver iterasjon, kalles Algo1, med kjøretid O(1) på, og verdien til j halveres. Siden j halveres for hver iterasjon, vil kjøretiden for algoritmen bli O(log(n).



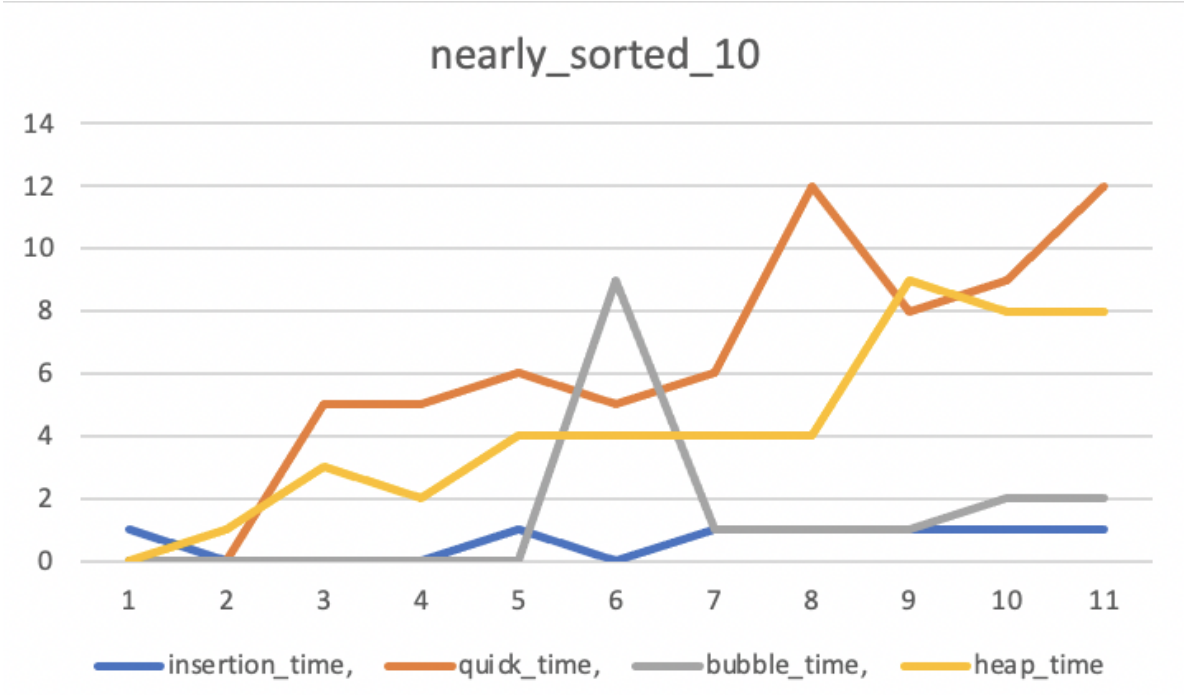
random_10



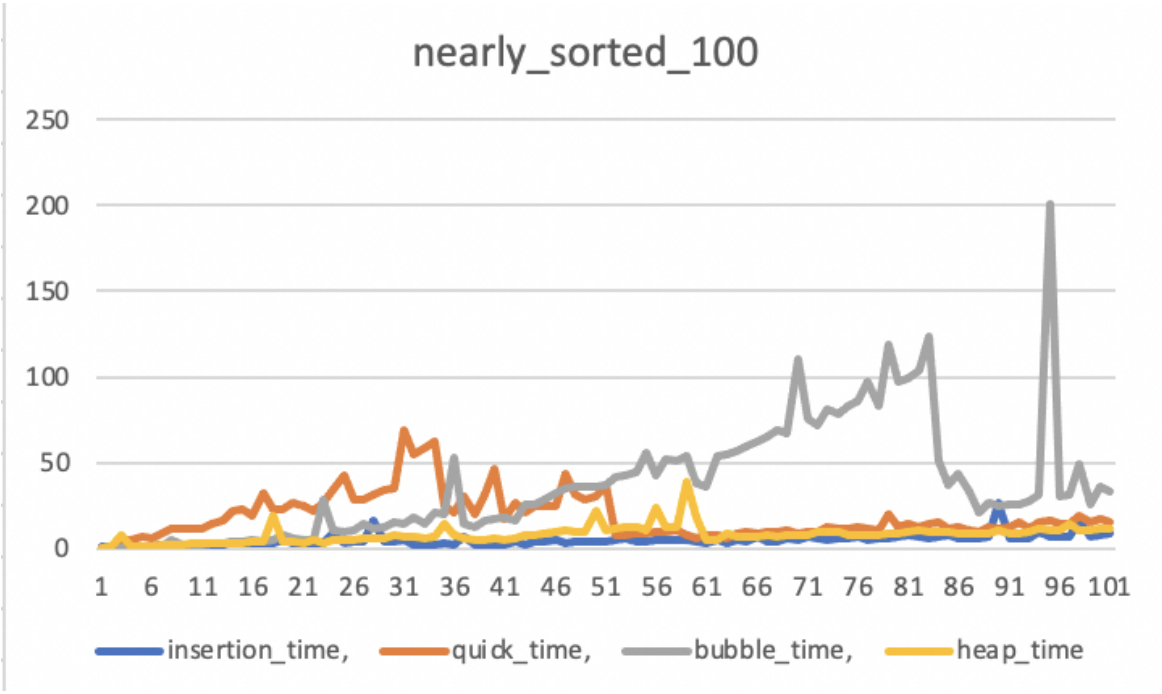
Random_100



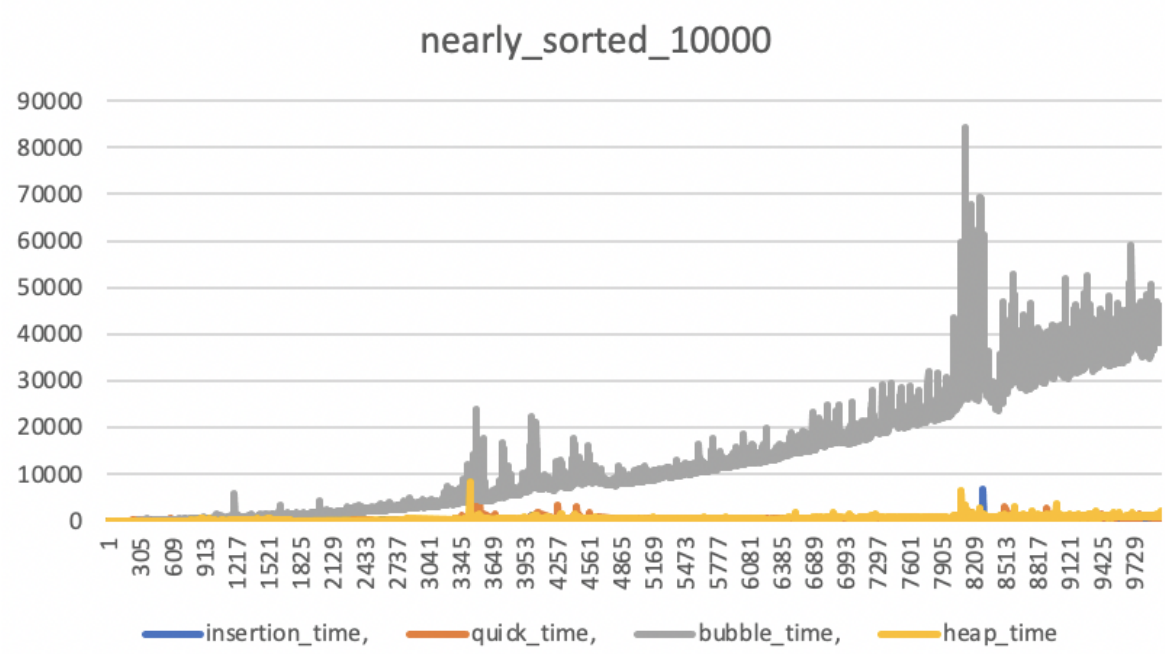
Random_10000



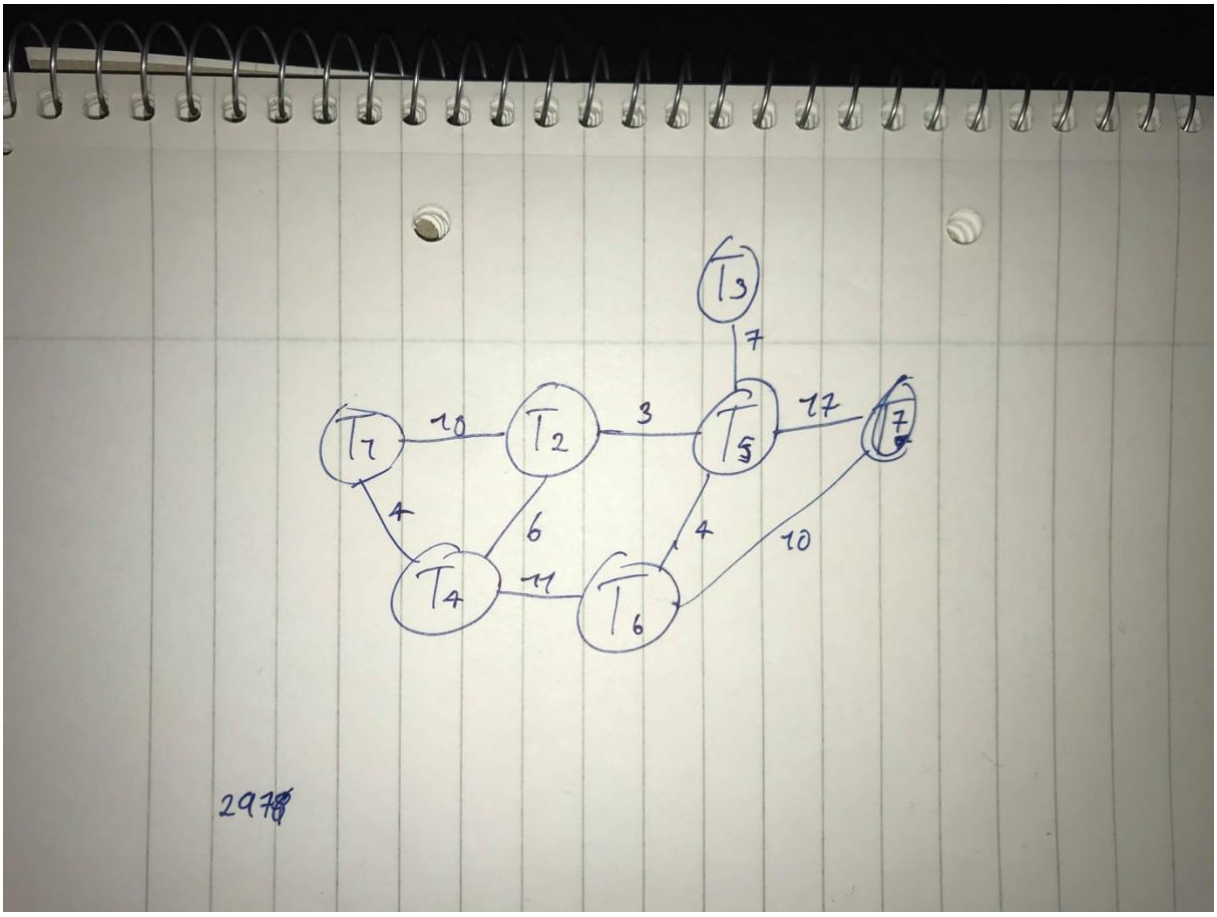
Nearly_sorted_10



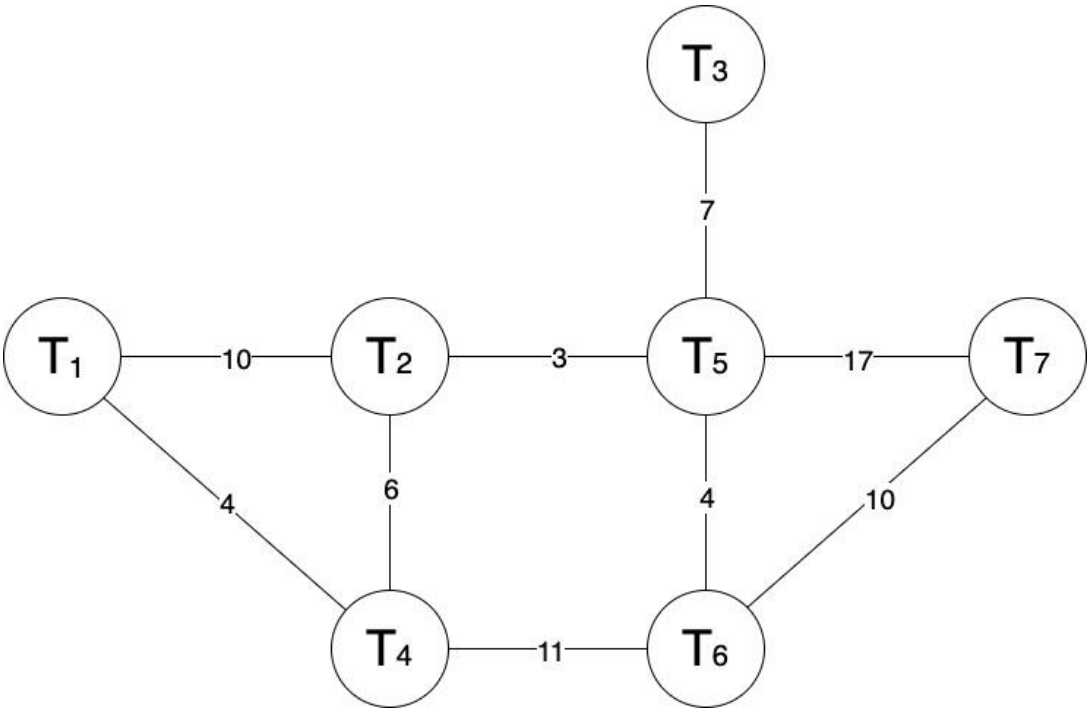
Nearly_sorted_100



Nearly_sorted_10000

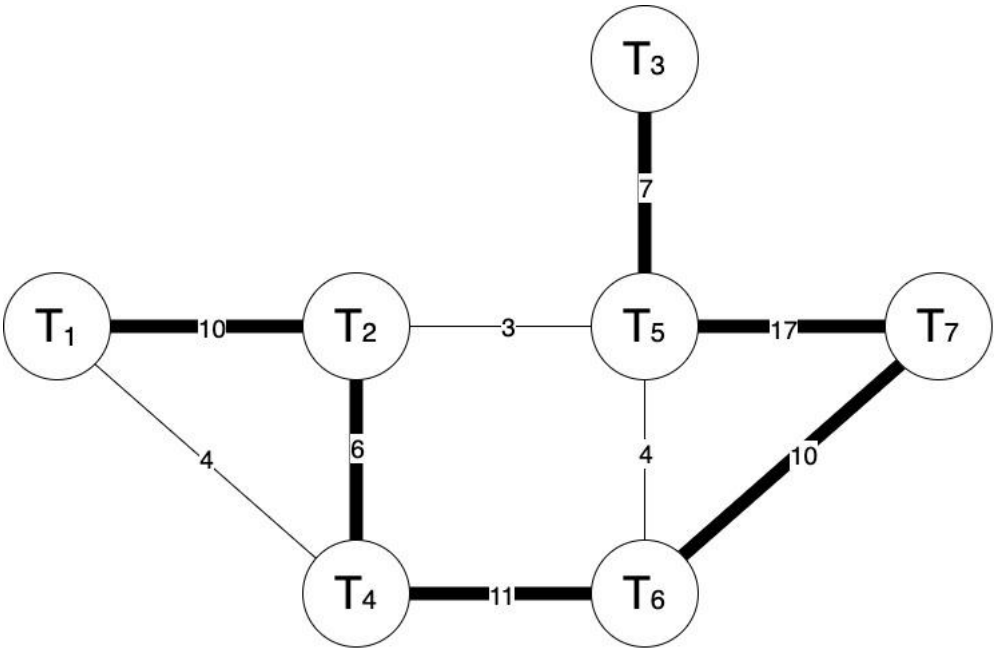


Håndtegning

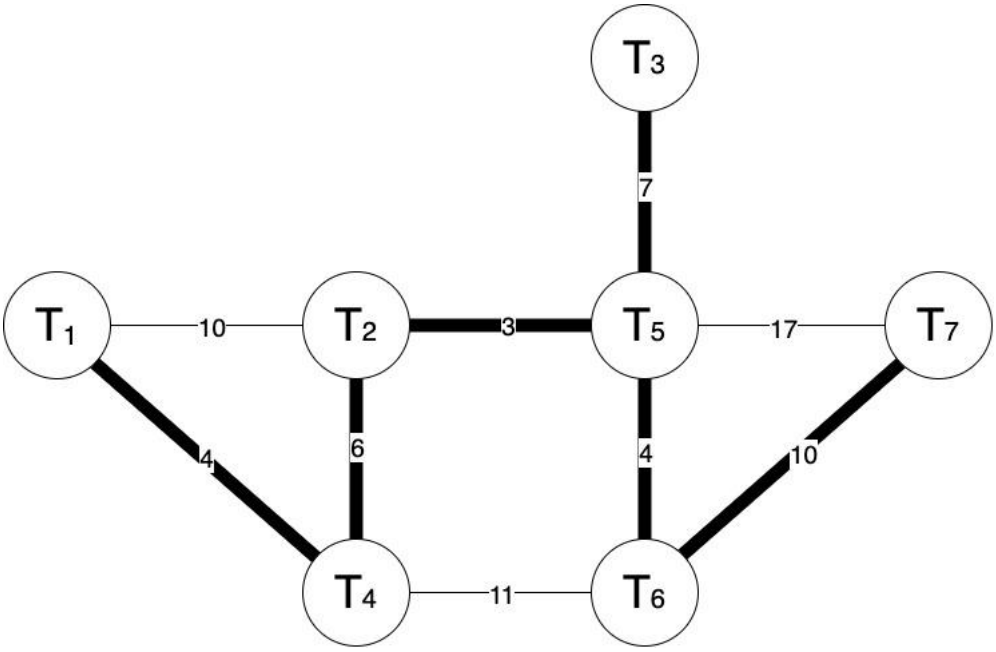


Uendret digital

Dyreste måte å koble sammen alle signaltårn:



Billigste måte å koble sammen alle signaltårn:



Billigste måte