1a-b.

```
aleksawo@itstud:~/htdocs/oblig4$ ./oppg1
Jeg er prosess 972571, min forelder er 971888
Jeg er prosess 972572, min forelder er 972571
Jeg er prosess 972573, min forelder er 972571
Jeg er prosess 972574, min forelder er 972572
aleksawo@itstud:~/htdocs/oblig4$
```

```
├─sshd(971858)──sshd(971887)──bash(971888)──oppg1(972571)─┬─oppg1(972572)──+
                                                          └─oppg1(972573)
├─sshd(972482)──sshd(972493)──bash(972494)──pstree(972575)
```

Forelderen til 972574 er 972572

1c. Når 'fork()' kalles lager det en kopi av eksisterende prosessen,

den foreldre og nyopprettede prosessen vil fortsette å kjøre

koden etter 'fork()'. Den nye prosessen kjører parallelt med

hovedprosessen. Det kalles det andre 'fork()' i begge prosessene og begge prosessene kan oprette sin egne prosesser.

Foreldre prosessen lager andre barnet på andre 'fork()', første barnet som ble laget på første 'fork()' lager et trede barn ved andre 'fork()'

1d.

```
aleksawo@itstud:~/htdocs/oblig4$ ./oppg1d
Jeg er prosess 975229, min forelder er 971888
aleksawo@itstud:~/htdocs/oblig4$ Jeg er prosess 975231, min forelder er 1
Jeg er prosess 975230, min forelder er 1
Jeg er prosess 975232, min forelder er 1
./oppg1d > output.txt
aleksawo@itstud:~/htdocs/oblig4$ cat output.txt
Jeg er prosess 975363, min forelder er 971888
Jeg er prosess 975365, min forelder er 1
Jeg er prosess 975364, min forelder er 1
Jeg er prosess 975366, min forelder er 1
aleksawo@itstud:~/htdocs/oblig4$
```

Barn 1 og 2 sover før PID skrives ut, mens foreldren ikke sover og skriver ut PID og PPID uten å vente, nye prosessene vil skrive ut

PID etter sleep(1) som fører til at foreldre-barn forholdet endres.

2a.

```
aleksawo@itstud:~/htdocs/oblig4$ emacs oppgave_2.c
aleksawo@itstud:~/htdocs/oblig4$ gcc -o oppg2 oppgave_2.c
aleksawo@itstud:~/htdocs/oblig4$ ./oppg2
Jeg er barneprosessen med PID 975645
Jeg er forelderprosessen med PID 975644
Barneprosessen 975645 har terminert med returstatus lik 42
aleksawo@itstud:~/htdocs/oblig4$ emacs oppgave_2.c
aleksawo@itstud:~/htdocs/oblig4$ gcc -o oppg2 oppgave_2.c
aleksawo@itstud:~/htdocs/oblig4$ ./oppg2
Jeg er forelderprosessen med PID 975906
Barneprosessen 975907 har terminert med returstatus lik 0
Jeg er barneprosessen med PID 975907
aleksawo@itstud:~/htdocs/oblig4$
```

b. Forelderen skriver PID først før barnets PID som er motsatt fra

første utskrift. Forelderen rapportere uventet status kode '0'.

Uten 'wait()' kan barn bli zombie-prosess til forelderen henter returverdi.

3a.

```
aleksawo@itstud:~/htdocs/oblig4$ emacs oppgave_3.c
aleksawo@itstud:~/htdocs/oblig4$ gcc -o oppgave_3 oppgave_3.c
aleksawo@itstud:~/htdocs/oblig4$ ./oppgave_3 &
[1] 976283
aleksawo@itstud:~/htdocs/oblig4$ Forelderprosessen med PID 976283 starter en evig løkke
Barneprosessen med PID 976284 avslutter
ps -l
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  3297  971888  971887  0  80   0 -  2560 do_wai pts/6    00:00:00 bash
0 S  3297  976283  971888  0  80   0 -   615 hrtime pts/6    00:00:00 oppgave_3
1 Z  3297  976284  976283  0  80   0 -     0 -      pts/6    00:00:00 oppgave_3
0 R  3297  976291  971888 99  80   0 -  2805 -      pts/6    00:00:00 ps
aleksawo@itstud:~/htdocs/oblig4$
```

b. 'kill <PID>', eller legge til '-9' foran PID hvis signalet ignoreres