Øving 05 - Algoritmer og datastruktur

Gruppe 17: John I. Eriksen og Emil Slettbakk

Oppgave 1: Hashtabell med tekstnøkler

Filer:

- HashTable.java
- TestHashTable.java

Kjører programmet/fila `TestHashTable.java` på navnet `John-Ivar Dalhaug Eriksen`:

```
John-Ivar Dalhaug Eriksen er med i faget!

Lastfaktor: 0.3925925925925926
Total kollisjoner: 29
Gjennomsnittlig kollisjoner per person: 0.21481481481481482
```

På navnet `Emil Slettbakk`:

```
Emil Slettbakk er med i faget!

Lastfaktor: 0.3925925925925926
Total kollisjoner: 29
Gjennomsnittlig kollisjoner per person: 0.21481481481481482
```

Oppgave 2: Hashtabeller og ytelse

Filer:

- DoubleHashing
- HashingStrategy
- HashPerformance
- LinearProbing

Setter opp en `range` mengden tall vi ønsker. For å gjøre tallene tilfeldige blir dette tallet ganget med `Math.random()`, som gir et tilfeldig tall mellom 0 og 1, og så ganget med 10. Dette blir castet til `int` for å fjerne desimaler.

```java
int range = 5_000_000;
(...)
int random = (int) (Math.random() * range * 10);
```

Programmet kjører deretter tallene inn i en hashtabell, med strategiene `LinearProbing` og `DoubleHashing`. Tabellen er blir initialisert slik at den er større enn antallet tall vi skal putte inn.

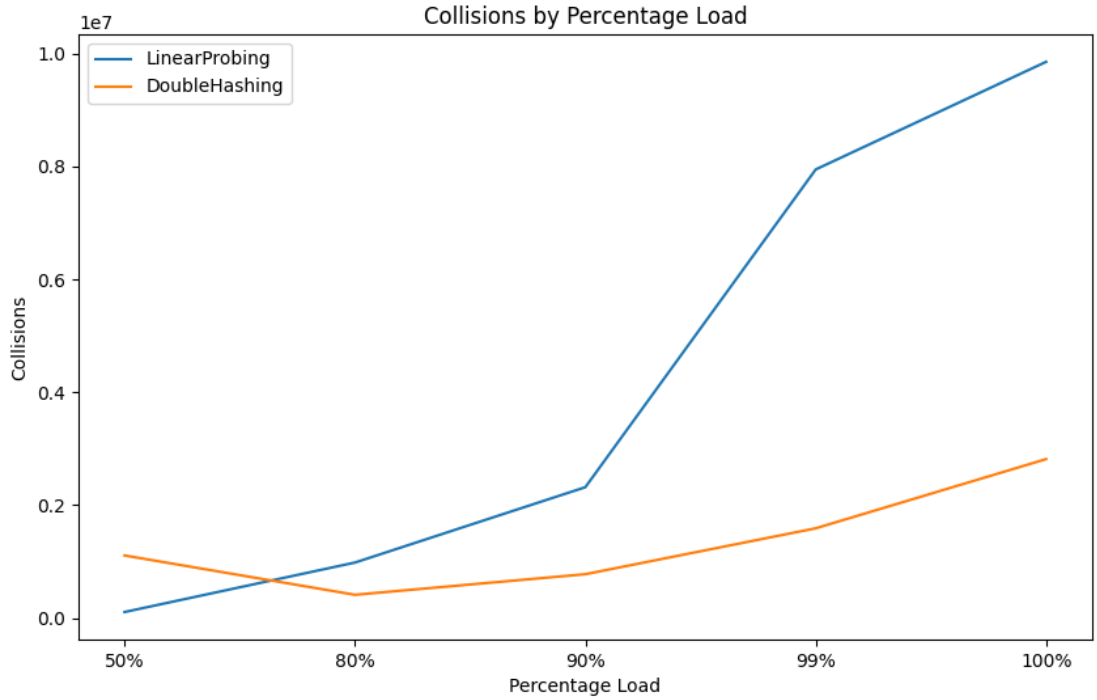
```
arr = new int[(int) Math.pow(2, Math.ceil(Math.log(length) / Math.log(2)))];
```

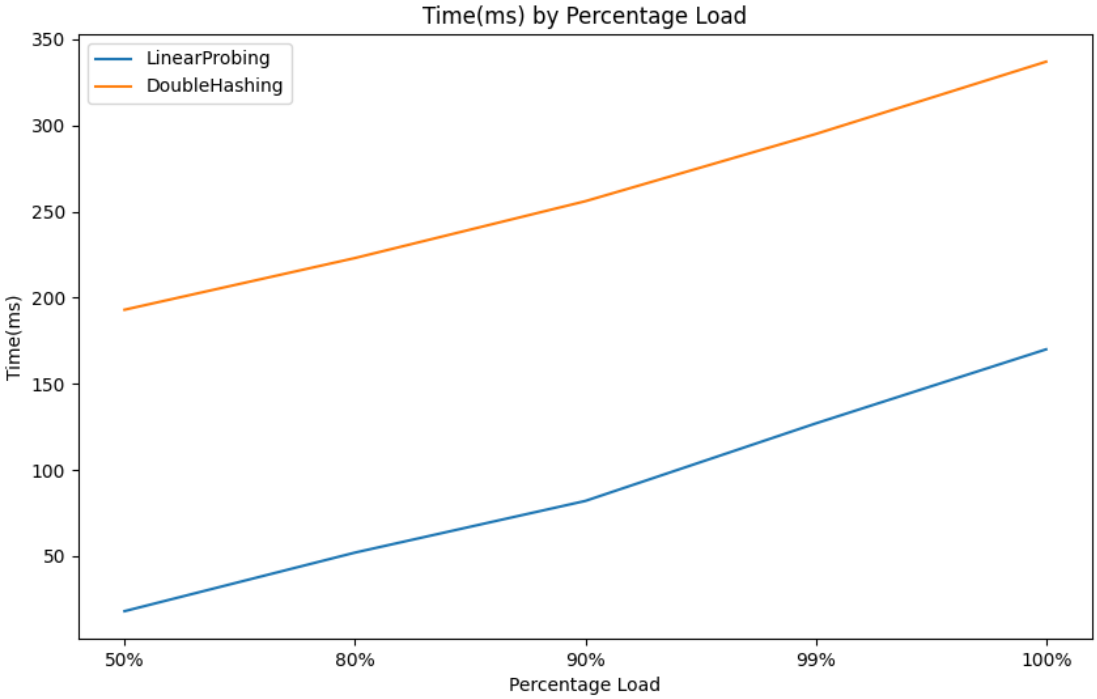
Oppgave 2.6.2 - Testdata

Tester med forskjellige tall på range. Som nevnt over vil dette tallet bli ganget med et tilfeldig tall og 10. Har med graf for kollisjoner og tidsforbruk for utvalgte range (1 million, 10 millioner, 50 millioner og 100 millioner).

range = 1_000_000

Strategy	Percentage	Collisions	Time(ms)
LinearProbing	50%	108492	18 ms
LinearProbing	80%	985005	52 ms
LinearProbing	90%	2320006	82 ms
LinearProbing	99%	7949302	127 ms
LinearProbing	100%	9855807	170 ms
DoubleHashing	50%	1110612	193 ms
DoubleHashing	80%	413926	223 ms
DoubleHashing	90%	778511	256 ms
DoubleHashing	99%	1591667	295 ms
DoubleHashing	100%	2817857	337 ms





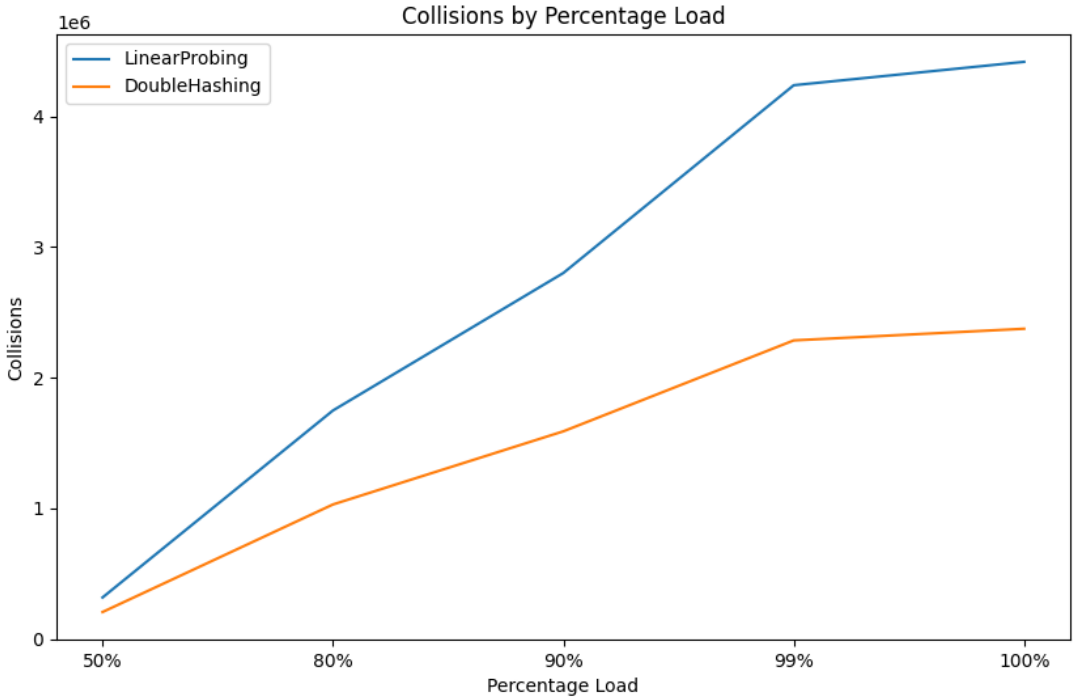
range = 5_000_000

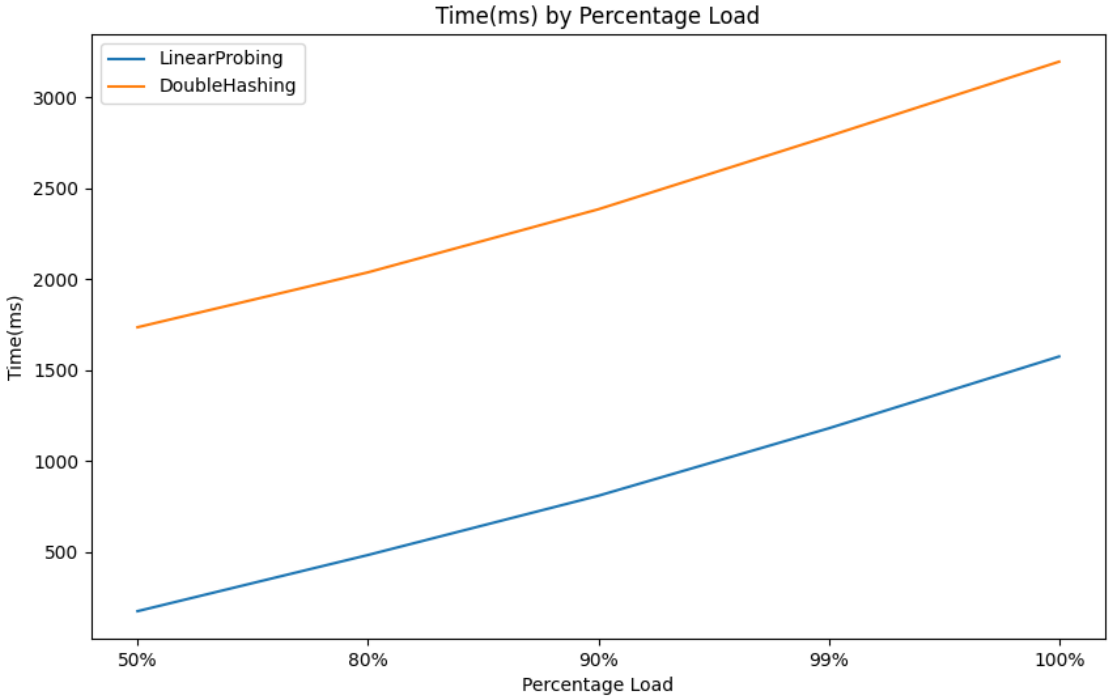
Strategy	Percentage	Collisions	Time (ms)
LinearProbing	50%	159048	72 ms
LinearProbing	80%	875937	205 ms
LinearProbing	90%	1404909	342 ms
LinearProbing	99%	2108510	494 ms

Strategy	Percentage	Collisions	Time (ms)
LinearProbing	100%	2208780	647 ms
DoubleHashing	50%	103411	723 ms
DoubleHashing	80%	517254	863 ms
DoubleHashing	90%	798094	1027 ms
DoubleHashing	99%	1150592	1215 ms
DoubleHashing	100%	1200096	1404 ms

range = 10_000_000

Strategy	Percentage	Collisions	Time (ms)
LinearProbing	50%	318118	173 ms
LinearProbing	80%	1749482	482 ms
LinearProbing	90%	2802081	808 ms
LinearProbing	99%	4239575	1179 ms
LinearProbing	100%	4418759	1574 ms
DoubleHashing	50%	206578	1735 ms
DoubleHashing	80%	1029989	2037 ms
DoubleHashing	90%	1589974	2384 ms
DoubleHashing	99%	2286019	2786 ms
DoubleHashing	100%	2375269	3196 ms





range = 15_000_000

Strategy	Percentage	Collisions	Time(ms)
LinearProbing	50%	1358771	280 ms
LinearProbing	80%	10773316	732 ms
LinearProbing	90%	22430659	1260 ms
LinearProbing	99%	50711069	1866 ms

Strategy	Percentage	Collisions	Time(ms)
LinearProbing	100%	56743568	2510 ms
DoubleHashing	50%	797095	2799 ms
DoubleHashing	80%	4945596	3437 ms
DoubleHashing	90%	25506281	4186 ms
DoubleHashing	99%	15262397	5067 ms
DoubleHashing	100%	16349660	5829 ms

range = 20_000_000

Strategy	Percentage	Collisions	Time (ms)
LinearProbing	50%	638854	403 ms
LinearProbing	80%	3495061	1138 ms
LinearProbing	90%	5621785	1985 ms
LinearProbing	99%	8452281	2931 ms
LinearProbing	100%	8843805	3890 ms
DoubleHashing	50%	411669	4334 ms
DoubleHashing	80%	2038947	5210 ms
DoubleHashing	90%	3139697	6257 ms
DoubleHashing	99%	4506012	7465 ms
DoubleHashing	100%	38242376	8761 ms

range = 25_000_000

Strategy	Percentage	Collisions	Time(ms)
LinearProbing	50%	1393284	530 ms
LinearProbing	80%	8843309	1455 ms
LinearProbing	90%	15384845	2524 ms
LinearProbing	99%	25792244	3732 ms
LinearProbing	100%	27313869	5075 ms

Strategy	Percentage	Collisions	Time(ms)
DoubleHashing	50%	856847	5682 ms
DoubleHashing	80%	4656430	6899 ms
DoubleHashing	90%	7507787	8356 ms
DoubleHashing	99%	11423656	10024 ms
DoubleHashing	100%	11973616	11708 ms

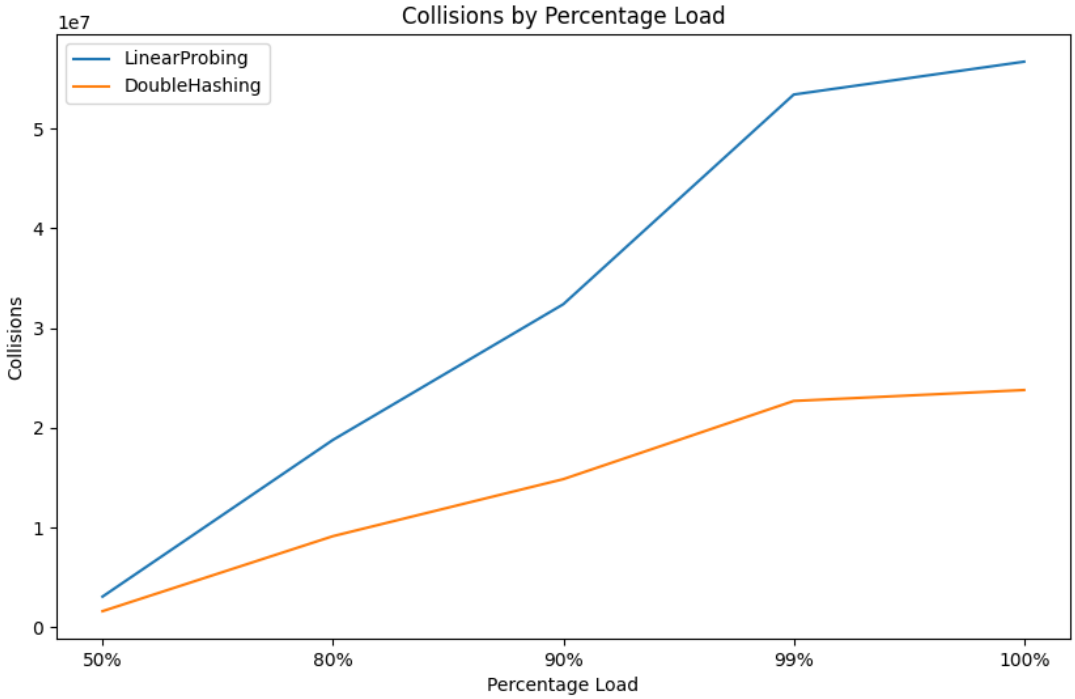
range = 30_000_000

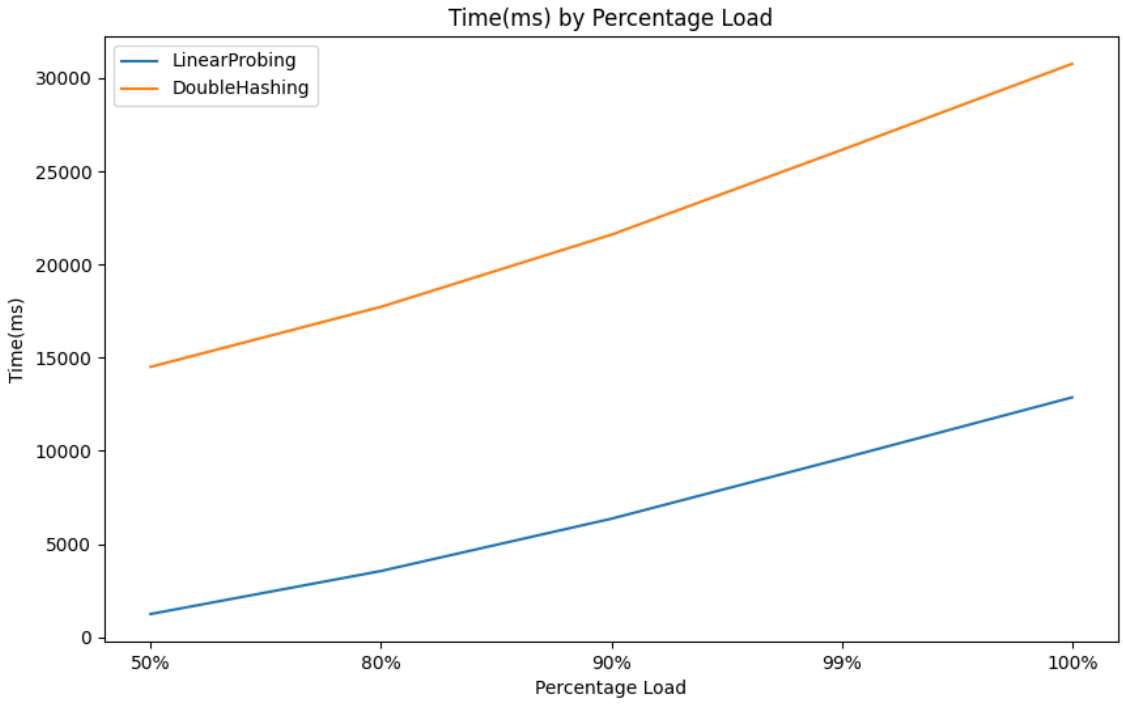
Strategy	Percentage	Collisions	Time(ms)
LinearProbing	50%	2717911	686 ms
LinearProbing	80%	21604403	1933 ms
LinearProbing	90%	44812787	3388 ms
LinearProbing	99%	101404933	5194 ms
LinearProbing	100%	113090407	6965 ms
DoubleHashing	50%	1589777	7773 ms
DoubleHashing	80%	9831362	9403 ms
DoubleHashing	90%	17336973	11424 ms
DoubleHashing	99%	30349375	13824 ms
DoubleHashing	100%	32479035	16281 ms

range = 50_000_000

Strategy	Percentage	Collisions	Time(ms)
LinearProbing	50%	3080058	1247 ms
LinearProbing	80%	18793932	3555 ms
LinearProbing	90%	32404890	6361 ms
LinearProbing	99%	53432350	9589 ms
LinearProbing	100%	56729228	12870 ms
DoubleHashing	50%	1616318	14506 ms

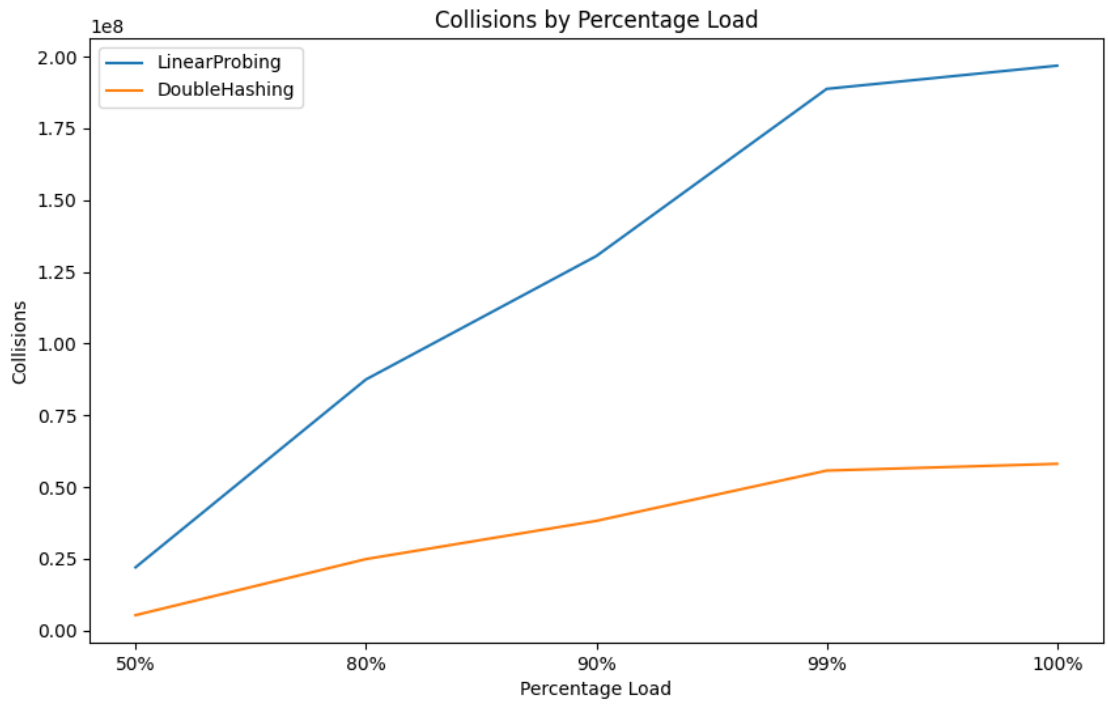
Strategy	Percentage	Collisions	Time(ms)
DoubleHashing	80%	9139034	17727 ms
DoubleHashing	90%	14858744	21598 ms
DoubleHashing	99%	22698660	26145 ms
DoubleHashing	100%	23795863	30766 ms

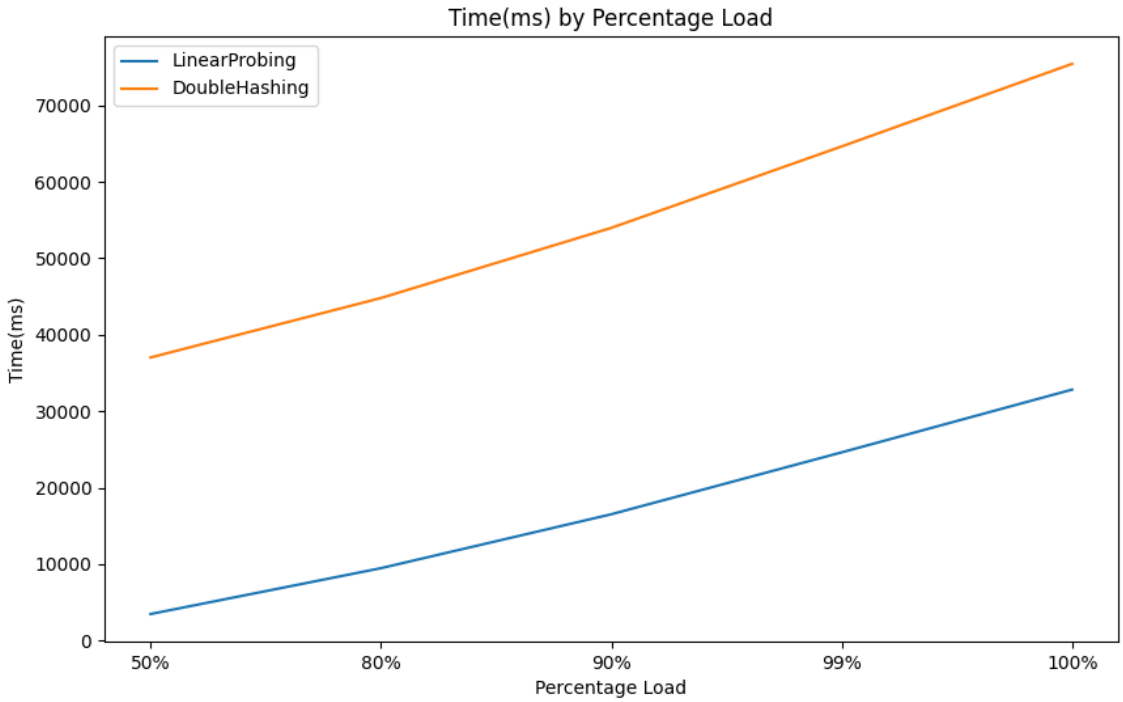




range = 100_000_000

Strategy	Percentage	Collisions	Time(ms)
LinearProbing	50%	22012284	3466 ms
LinearProbing	80%	87484994	9465 ms
LinearProbing	90%	130498707	16521 ms
LinearProbing	99%	188758990	24622 ms
LinearProbing	100%	196841043	32828 ms
DoubleHashing	50%	5338242	37029 ms
DoubleHashing	80%	24883472	44812 ms
DoubleHashing	90%	38201980	53990 ms
DoubleHashing	99%	55726258	64647 ms
DoubleHashing	100%	58085241	75450 ms





Oppgave 2.6.2

> a) Tar fler kollisjoner mer tid?

Ut fra tabellene ser vi at jo flere kollisjoner, jo mer tid går med til å fylle tabellen, men tidsøkningen er ikke dramatisk

> b) Er det grenser for hvor full en hashtabell bør være?

Ja. Denne grensen er tabellens lastfaktor (load factor), og beregnes ut fra antall elementer delt på tabellens kapasitet.

$$Lastfaktor = \frac{Antall\ elementer}{Tabellkapasitet}$$

En lavere lastfaktor vil gi bedre ytelse, men økt minnebruk, fordi tabellen må være mye større enn antall elementer og det er færre kollisjoner. En høyere lastfaktor vil gi mindre minnebruk, men dårligere ytelse, fordi tabellens størrelse er nærmere (eller likt) antallet elementer, men flere kollisjoner.

> c) Er ytelsen (i tid eller kollisjoner) ulik for ulike typer hashing? Er noen klart best, eller best ved visse fyllingsgrader?

Fra disse beregningene, er LinearProbing raskere enn DoubleHashing.

> d) Andre interessante observasjoner?

DoubleHashing har færre kollisjoner enn LinearProbing, og bruker lengre tid.