Obligatorisk innlevering 23.mars

# Oppgave 1

**I)**

package no.hvl.dat102;

public class SolveTowers {

public static void main(String[] args) {

long startTid = System.nanoTime();

TowersOfHanoi towers = new TowersOfHanoi(28);

towers.solve();

System.out.println("Antall flyttinger: " + towers.flyttinger);

long slutttid = System.nanoTime();

System.out.print("kjøringen tok: " + (slutttid - startTid) + " ns");

}

}

package no.hvl.dat102;

public class TowersOfHanoi {

private int totalDisks;

int flyttinger;

public TowersOfHanoi(int disks) {

this.totalDisks = disks;

flyttinger = 0;

}

public void solve() {

moveTower(totalDisks, 1, 3, 2);

}

private void moveTower(int numDisks, int start, int end, int temp) {

if (numDisks == 1) {

moveOneDisk(start, end);

} else {

moveTower(numDisks - 1, start, temp, end);

moveOneDisk(start, end);

moveTower(numDisks - 1, temp, end, start);

}

}

private void moveOneDisk(int start, int end) {

// System.out.println("Move one disk from " + start + " to " + end);

flyttinger++;

}

}

package no.hvl.dat102;

public class SumSn {

public static int oppsum(int n) {

if (n == 1) {

return 1;

} else {

return oppsum(n - 1) + n;

}

}

// 5an-1 -6an-2 +2

public static int hvertledd(int n) {

if (n > 1) {

return 5 \* hvertledd(n - 1) - 6 \* hvertledd(n - 2) + 2;

} else if(n == 1) {

return 5;

} else {

return 2;

}

}

public static void main(String[] args) {

System.out.println(oppsum(100));

System.out.println("----------------------");

// test hvert ledd test

for (int i = 0; i < 10; i++) {

System.out.println(hvertledd(i));

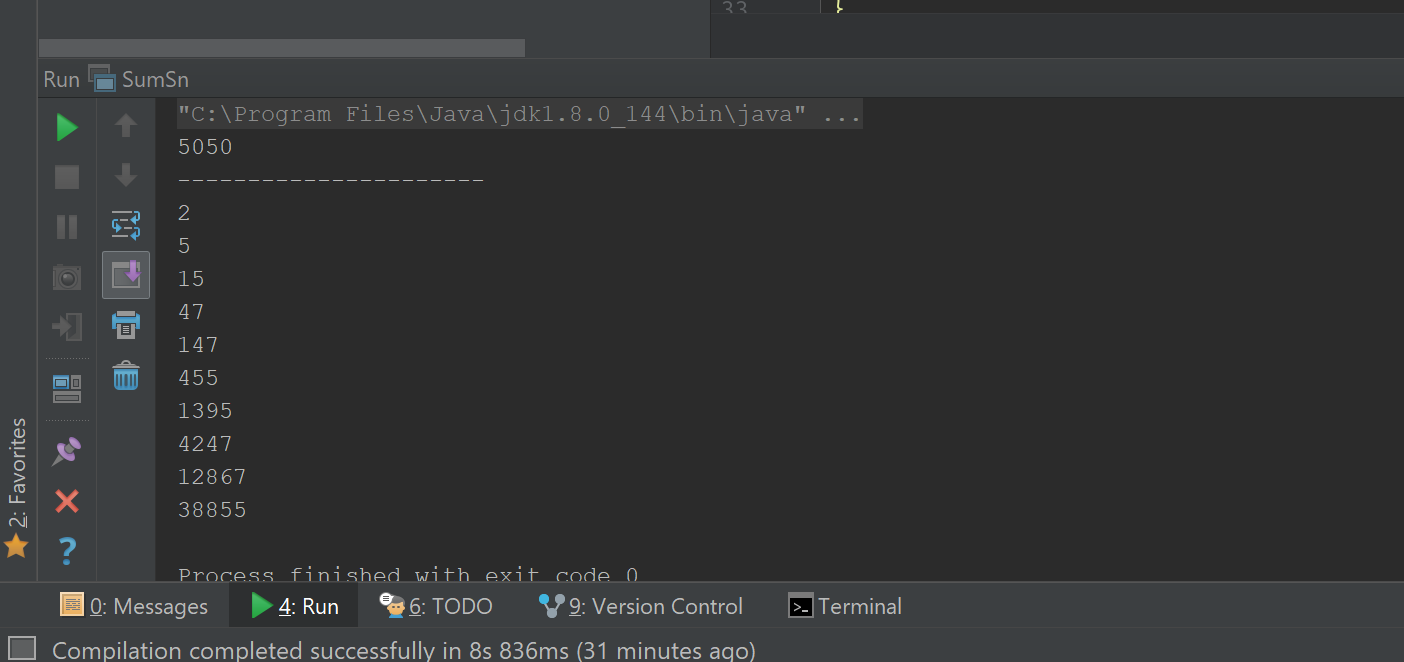
}

}

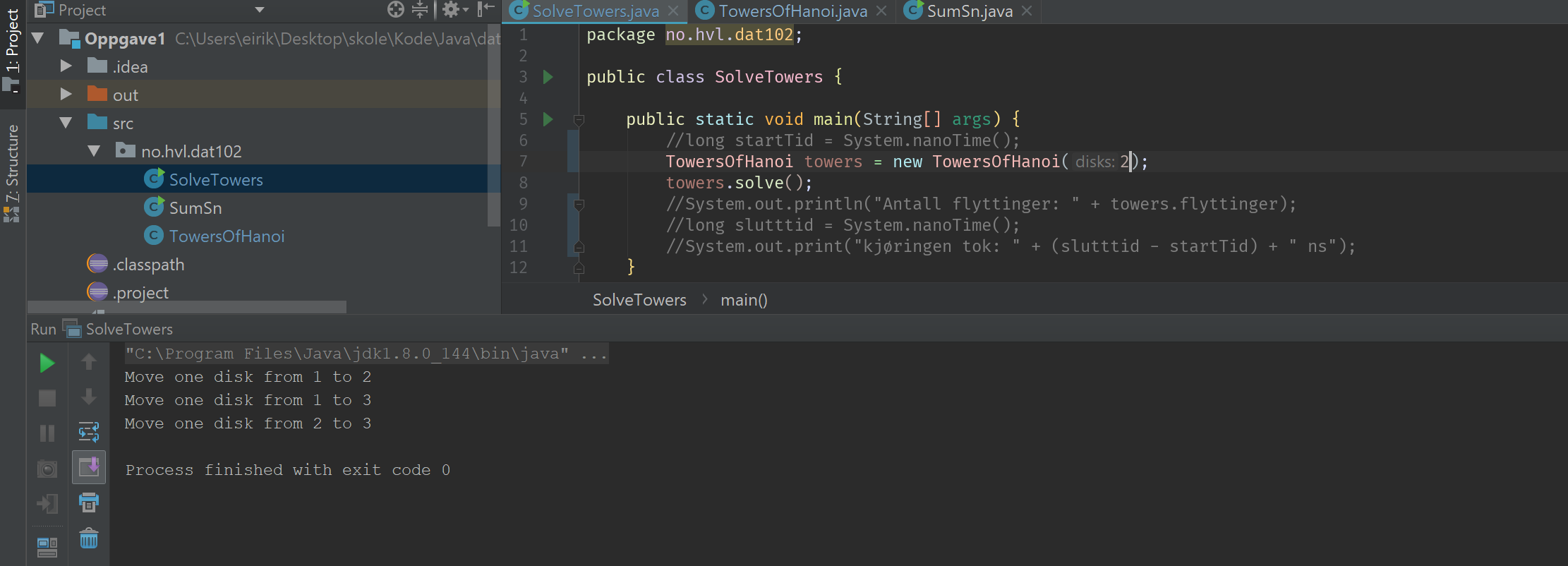
}

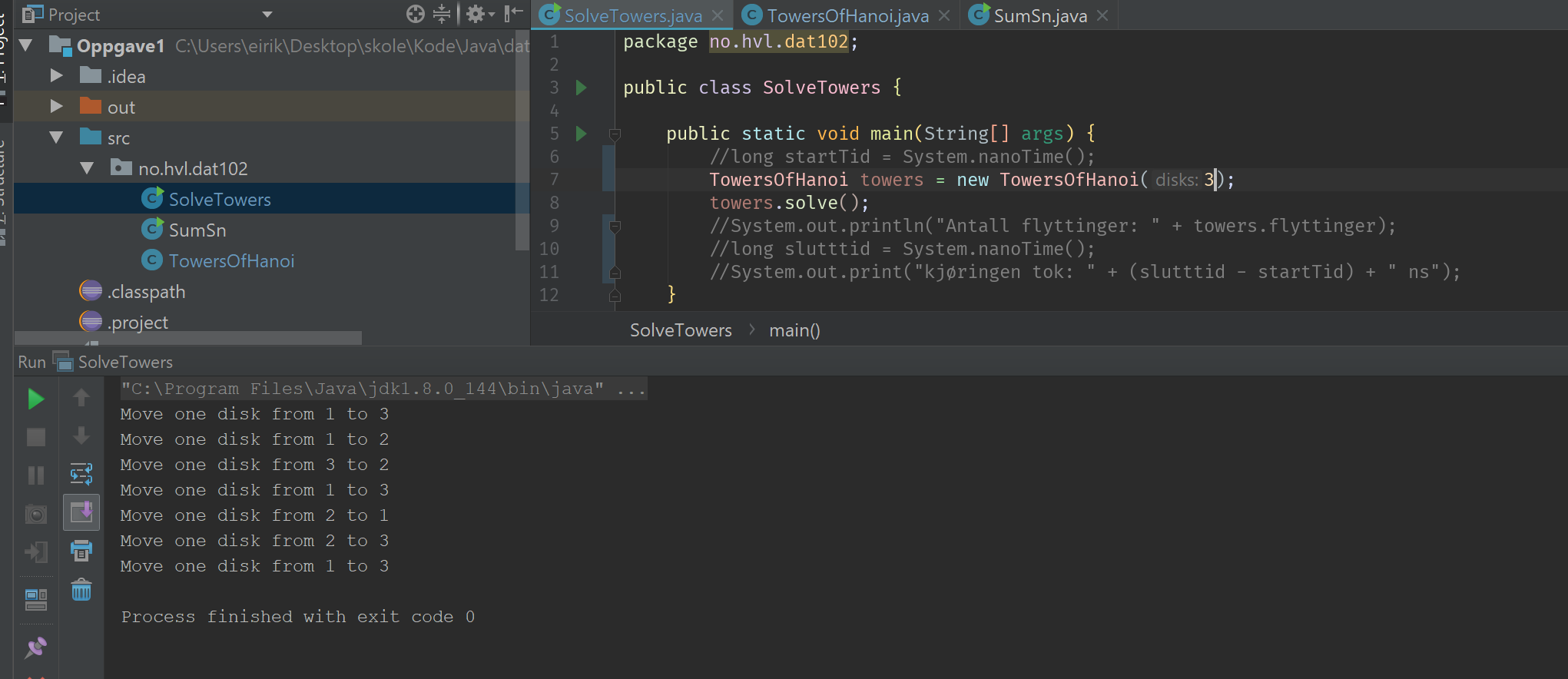
UTSKRIFT

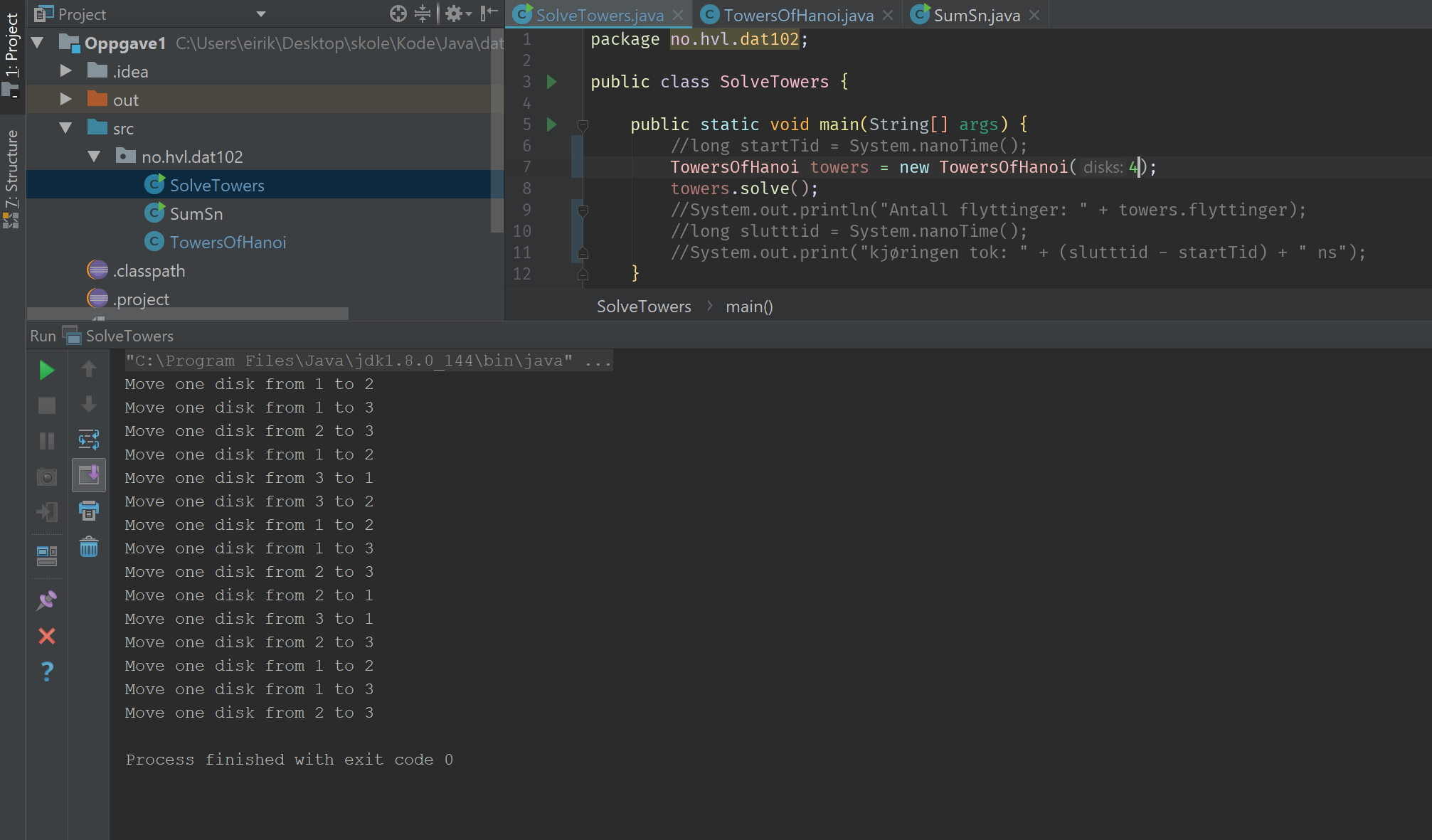
A & B:

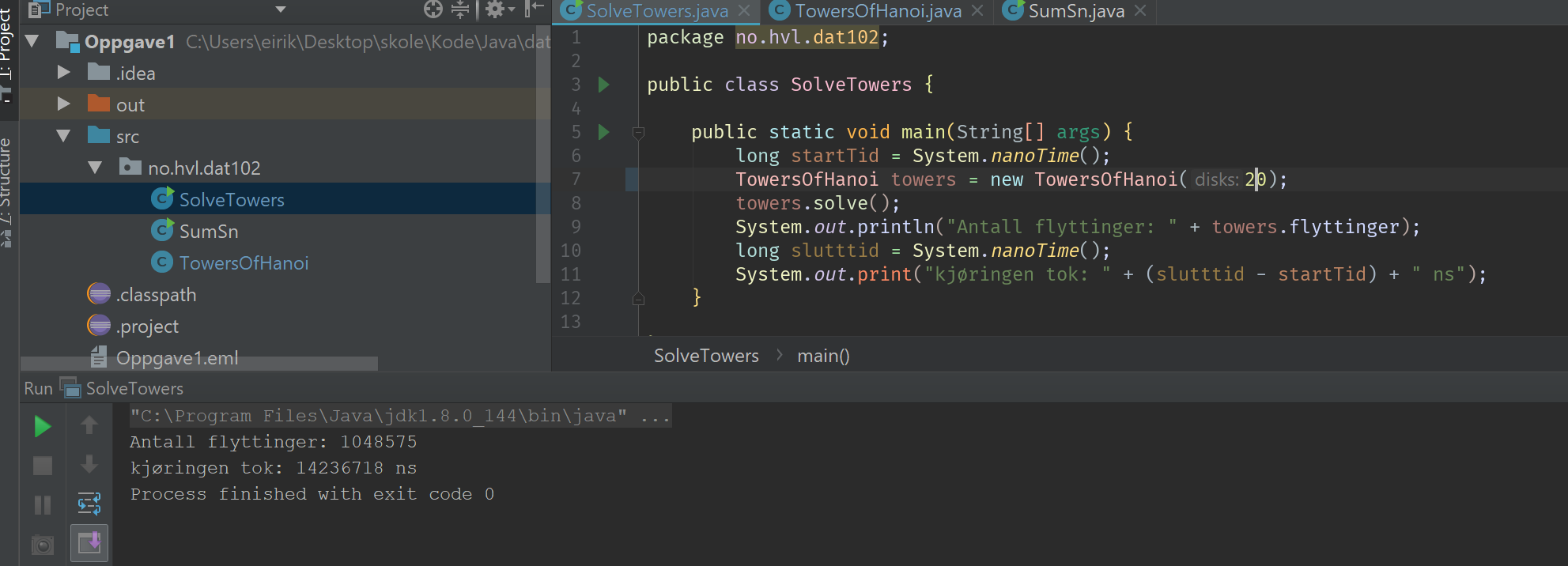


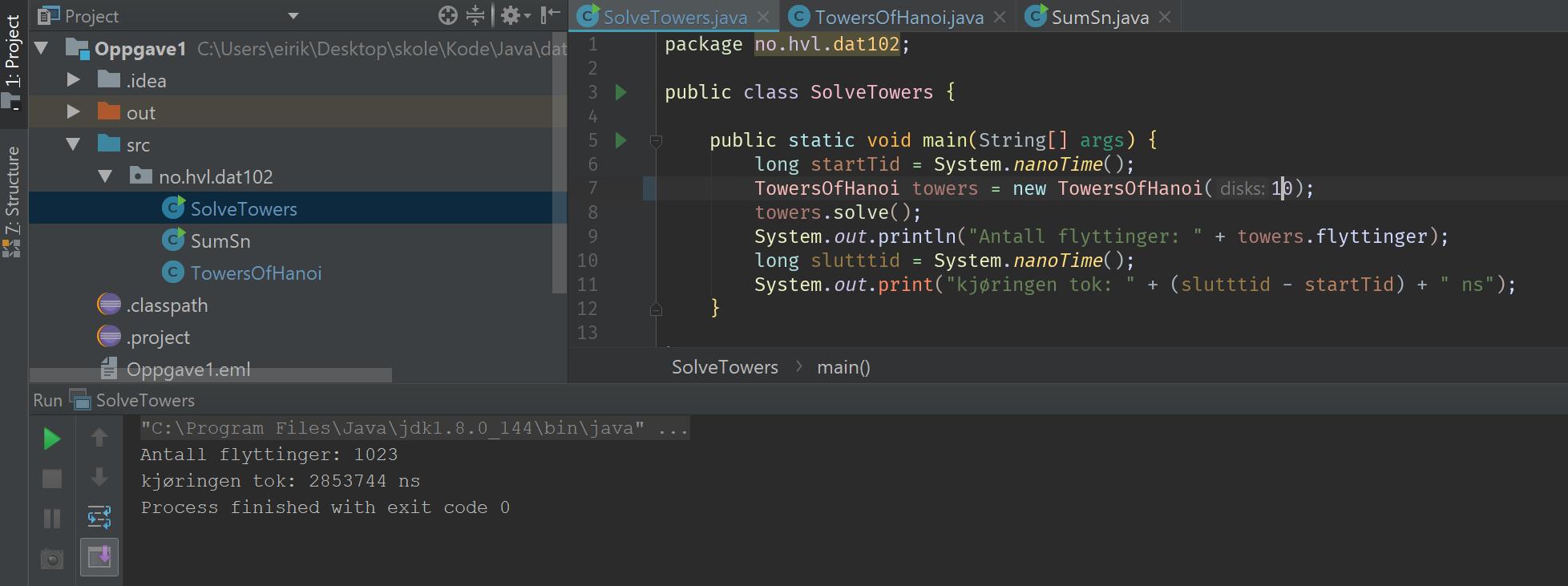
C:

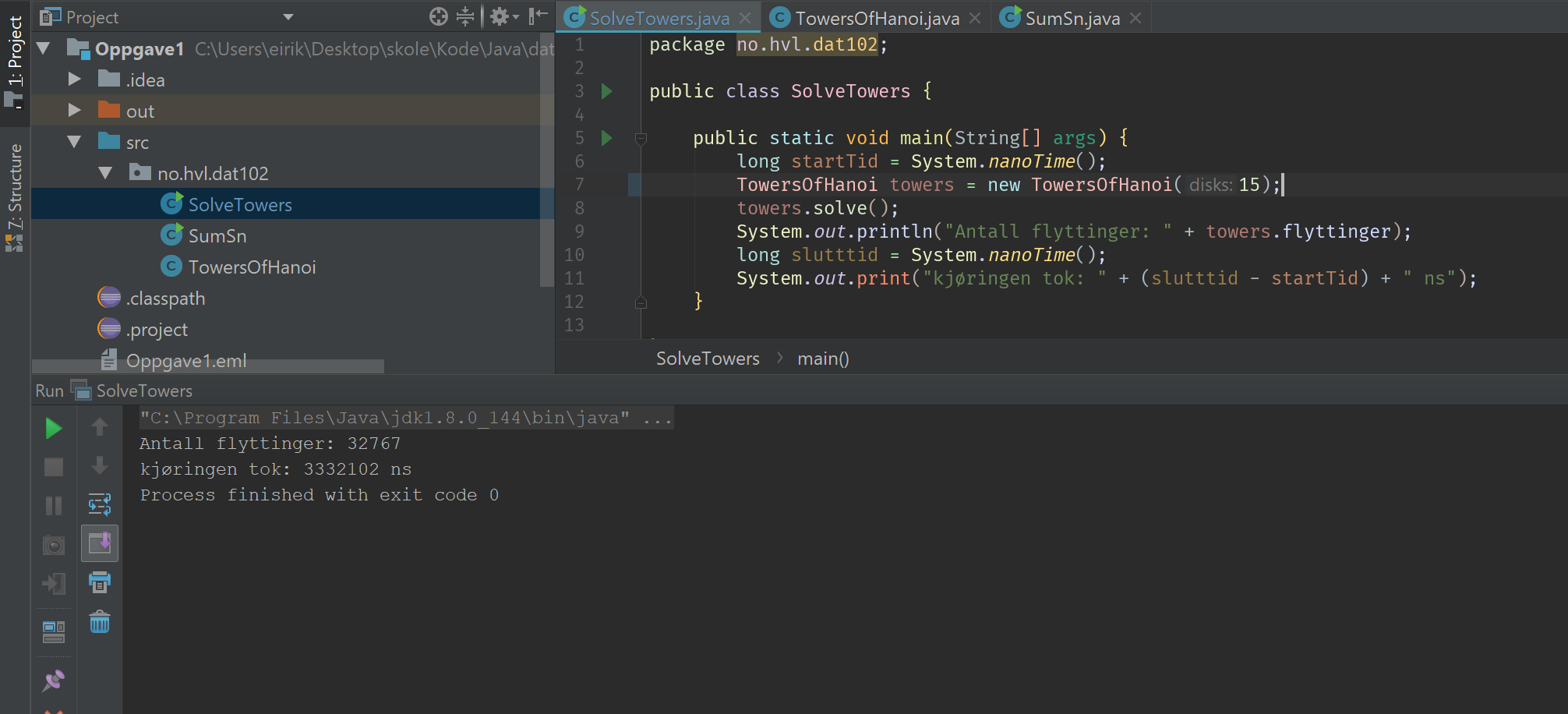


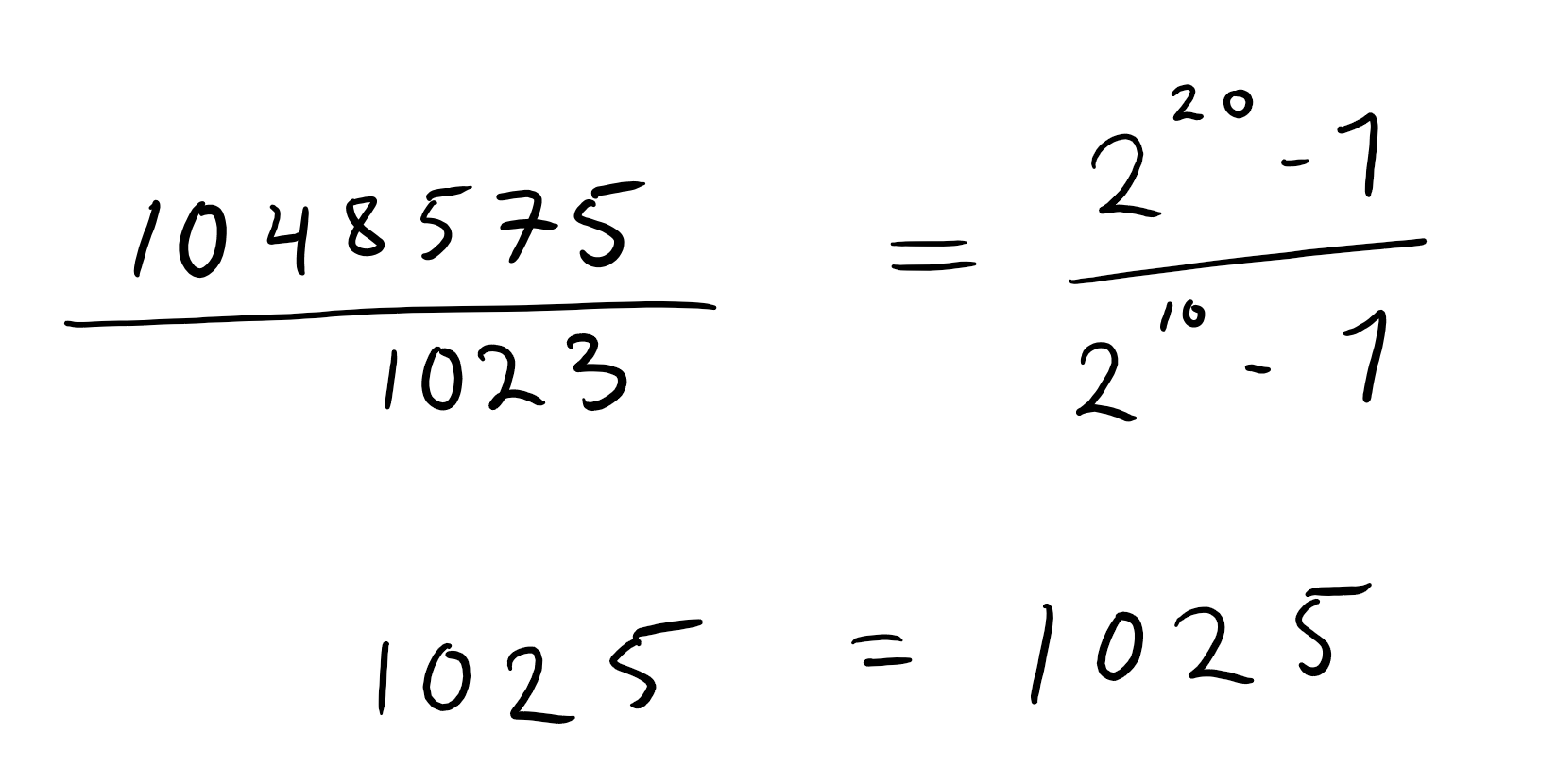












# Oppgave 2

d)

i) gjennomsnitt : n/2 + 0.5: tar alle tilfellene fra 1 sammenliging (finner det på første element), fram til n elementer og deler på antall elementer.

verste tilfellet: n : finner det på siste element (går gjennom alle elementer)

ii) gjennomsnitt : n/4 + 0.75 leter bare gjennom halve tabellen, så blir som i «i)»

verste tilfellet: n/2 blir som i «i» bare med at du søker gjennom halve tabellen.

iii) gjennomsnitt: ca n/8 siden man da halverer samlingen enda en gang

verste tilfellet: n/4 siden den er halvert enda en gang.

# Oppgave 3

b)

Ønsker å søke etter nummer 8 I tabellen og retunere plassen i tabellen.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| 2 | 4 | 5 | 7 | 8 | 10 | 12 | 15 | 18 | 21 | 23 | 27 | 29 | 30 | 31 |

1. Først setter man start = 0 og slutt = tab.length – 1
2. så finner man midten og sjekker om tallet er lik tallet i denne posisjonen
3. om ikke må man sjekke om tallet er større eller mindre en midt pekeren, er den større kutter vi ut alt til venstre + midt pekeren og gjøre et nytt søk. Er tallet vi leitere etter mindre en midt pekeren kutter vi ut alt til høgre + midt pekeren og gjøre et nytt søk på venstre delen.

i dette tilfelle vil neste tabell se slik ut

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 2 | 4 | 5 | 7 | 8 | 10 | 12 |

Prosessen igjen tar seg og vi får i neste runde denne tabellen siden 8 er større en 7 går vi til høgresiden i tabellen

|  |  |  |
| --- | --- | --- |
| 4 | 5 | 6 |
| 8 | 10 | 12 |

Samme skjer igjen og vi går til veneste siden 8 er mindre en 10. vi får da at tall == tab[mid] og vi returnerer posisjonen vi fikk match i

c)

Ønsker å søke etter nummer 16 I tabellen og returnere plassen i tabellen eller -1 om den ikke finnes.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| 2 | 4 | 5 | 7 | 8 | 10 | 12 | 15 | 18 | 21 | 23 | 27 | 29 | 30 | 31 |

Finner at 16 er størren en midt pekeren (15). hele venstresiden og midt pekeren blir kuttet ut.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| 18 | 21 | 23 | 27 | 29 | 30 | 31 |

Finner at midt pekeren ikke er like tallet vi leter etter men mindre så kutter ut høgresiden pluss midt pekeren

|  |  |  |
| --- | --- | --- |
| 8 | 9 | 10 |
| 18 | 21 | 23 |

Finner at midt pekeren ikke er like tallet vi leter etter men mindre så kutter ut høgresiden pluss midt pekeren

|  |
| --- |
| 8 |
| 18 |

Forsatt er 16 midre en 18 så man kommer ut av løkka her og retunerer -1:

While (start<=slutt)

|  |  |  |
| --- | --- | --- |
| 1 | Start = 8 | Slutt = 14 |
| 2 | Start = 8 | Slutt = 10 |
| 3 | Start = 8 | Slutt = 8 |
| 4 | Start = 8 | Slutt = 7 |

Vi får 1 nøkkelsammenligning pr. rekursiv kall. I verste tilfelle blir det log2n. Hvordan stemmer det med søk av 16 som ikke fins? Det stemmer.

# Oppgave 4

a & b)

Finne c:

Ink Drawings
Ink Drawings
￼
￼
￼￼￼
￼
￼￼￼
￼


Flettesortering:

|  |  |  |  |
| --- | --- | --- | --- |
| N | Antall målinger | Målt tid (gjennomsnitt) | Teoretisk tid  C\*f(n) |
| 32000 | 10 | 30 714 830 ns | 368756924.8 ns |
| 64000 | 10 | 106 019 569 ns | 78679384.5 ns |
| 128 000 | 10 | 92 844 102 ns | 1672147699 ns |

QuickSort:

|  |  |  |  |
| --- | --- | --- | --- |
| N | Antall målinger | Målt tid (gjennomsnitt) | Teoretisk tid  C\*f(n) |
| 32000 | 10 | 29 056 123 ns | 368756924.8 ns |
| 64000 | 10 | 63 992 861 ns | 786793849.5 ns |
| 128 000 | 10 | 226 110 810 ns | 1672147699 ns |

Bublesort:

|  |  |  |  |
| --- | --- | --- | --- |
| N | Antall målinger | Målt tid (gjennomsnitt) | Teoretisk tid  C\*f(n) |
| 1000 | 10 | 10 463 630 ns | 770000000 ns |
| 5000 | 10 | 140 497 025 ns | 1.925\*10^10 ns |
| 10 000 | 10 | 520 784 287 ns | 7.7\*10^10 ns |

Sortering ved utvalg: O(n^2)

|  |  |  |  |
| --- | --- | --- | --- |
| N | Antall målinger | Målt tid (gjennomsnitt) | Teoretisk tid  C\*f(n) |
| 1000 | 10 | 10 393 641 ns | 770000000 ns |
| 5000 | 10 | 62 620 307 ns | 1.925\*10^10 ns |
| 10 000 | 10 | 334 246 769 ns | 7.7\*10^10 ns |

Sortering ved Insetting: O(n^2)

|  |  |  |  |
| --- | --- | --- | --- |
| N | Antall målinger | Målt tid (gjennomsnitt) | Teoretisk tid  C\*f(n) |
| 1000 | 10 | 16 090 953 ns | 770000000 ns |
| 5000 | 10 | 39 079 507 ns | 1.925\*10^10 ns |
| 10 000 | 10 | 143 246 810 ns | 7.7\*10^10 ns |

Sortering ved Radix sort: O(n)

|  |  |  |  |
| --- | --- | --- | --- |
| N | Antall målinger | Målt tid (gjennomsnitt) | Teoretisk tid  C\*f(n) |
| 32 000 | 10 | 5 812 596 ns | 24 640 000 ns |
| 64 000 | 10 | 9 337 011 ns | 49 280 000 ns |
| 128 000 | 10 | 13 305 899 ns | 98 560 000 ns |

c)

Radix Sort!

Usortert tabell.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 123 | 398 | 210 | 019 | 528 | 513 | 129 | 294 |

Sorterer inn i midlertidige tabeller med tallet lengst til høgre som utgangspunkt. Starter med tallet først i tabeller (tab[0]).

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 210, |  |  | 123,513 | 294 |  |  |  | 398,528 | 019,129 |

Setter så tilbake i opprinnelige tabell i den rekkefølgen de ble satt inn i de midlertidige tabellene og kjører sortering med hensyn på det midterste tallet

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 210 | 123 | 513 | 294 | 398 | 528 | 019 | 129 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|  | 210,513,019 | 123,129 | 528 |  |  |  |  |  | 294,398 |

Setter tilbake i tabellen og sorterer igjen med hensyn på siste tallet

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 210 | 513 | 019 | 123 | 129 | 528 | 294 | 398 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 019 | 123,129 | 210,294 | 398 |  | 513,528 |  |  |  |  |

Setter tilbake i tabellen og ser at tabellen er sortert

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 019 | 123 | 129 | 210 | 294 | 398 | 513 | 528 |

d)

Når n = 100:

vanlig kvikksort: 483938 ns

Nye kvikksort: 630276 ns

Da vil det si at den nye metoden er raskere en vanlig kvikksort