Probabilistic Analysis

Spørgsmål 5 fra Exam Questions

Kevin Vinther

December 13, 2023

Table of Contents i

Randomized Divide and Conquer: Median-Finding and Quicksort

Problem: Finding the median

- Antag at vi er givet et sæt af n tal: $S = \{a_1, a_2, \ldots, a_n\}$
- Deres median er tallet i midten hvis vi sorterer dem.
- Antag at k er medianens plads.
- Hvis n er lige, så $k = n/2$.
- Hvis n er ulige, så $k = \frac{n+1}{2}$
- I det følgende problem, antager vi at tallene er distinkte.

Køretid

- Hvad ville køretiden være på en algoritme der finder medianen?

Køretid

- Hvad ville køretiden være på en algoritme der finder medianen?
- Intutionen siger nok $O(n \log n)$. Sortér først, og så tag midterværdien.
- Men, hvad hvis jeg siger, at med en divide-and-conquer algoritme, kan man finde medianen på $O(n)$?

Design af Algoritmen i

- Det første skridt er problemet af *selection*.
- Givet et sæt af n tal, S, og et tal k, mellem 1 og n, overvej funkitionen `Select(S,k)` som returnerer det k'e største element i S.
- `Select(S,k)` kan dermed finde medianen ved `Select(S, n/2)`, `Select(S, (n+1)/2)`
- Den basale struktur af algoritmen fungerer således:
- Vi vælger et element $a_i \in S$, "the splitter".
- Vi definerer følgene sæt
  - $S^- = \{a_j : a_j < a_i\}$
  - $S^+ = \{a_j : a_j > a_i\}$.

Design af Algoritmen ii

- Derefter kan vi bestemme om sæt S^- eller S^+ indeholder det k'e største element, og så bliver vi kun ved ved dette sæt.
- Vi er lige nødt til at snakke om pseudokoden (for den er fandeme forvirrende).

Pseudokode

```
Select(S,k):
Choose a splitter a_i ∈ S
For each element a_j of S
   Put a_j in S^- if a_j < a_i
   Put a_j in S^+ if a_j > a_i
Endfor
If |S^-| = k − 1 then
   The splitter a_i was in fact the desired answer
Else if |S^-| ≥ k then
   The k^th largest element lies in S^-
   Recursively call Select(S^-, k)

Else suppose |S^-| = ℓ < k − 1
   The k^th largest element lies in S^+
   Recursively call Select(S^+, k − 1 − ℓ)
Endif
```

Theorem 13.17

- Kig på pseudokoden. Algoritmen bliver altid kaldt på et mindre sæt, så derfor må den terminere. Observér også at hvis $|S| = 1$, så har vi $k = 1$, og dette element bliver så returneret.
- Fra valget af hvilket sæt vi skal kalde `Select(s,k)` på, er det klart at se at det også virker når $|S| > 1$.

Theorem (13.17)

Regardless of how the splitter is chosen, the algorithm above returns the k^{th} largest element of S.

- Af en eller anden grund er følgende ikke et bevis? Jeg forstår ikke den her bog.

Choosing a good splitter i

- Køretiden afhænger af hvordan vi vælger **splitteren**.
- Hvis vi kan vælge en splitter i lineær tid, tager resten af algoritmen også lineær tid, plus tiden for det rekursive kald.
- **Mål**: Vi vil gerne finde en splitter der gør sættet vi skal kigge på så småt som muligt.
- **Eksempel**: Hvis vi altid kunne vælge medianen som splitter[1], så kunne vi vise et lineært bound bå køretiden som følger:
- Lad cn være køretiden for `Select`, uden at tælle tiden for det rekursive kald.
- Med medianen som splitter, ville køretiden $T(n)$ være bounded af recurrencen $T(n) \leq T(n/2) + cn$

Choosing a good splitter ii

- Ifølge bogen har dette løsningen $T(n) = O(n)$. Jeg håber ikke jeg er den eneste der ikke forstår recurrence relations, så forhåbentligt nogen kan hjælpe ♡
- Derfor, hvis vi havde en måde at vælge splittere a_i på, således at der var mindst εn elementer både større og mindre end a_i, for enhver fastsat konstant $\varepsilon > 0$.
- I så fald ville sættet i det reskursive kald skrumpe med en factor af mindst $1 - \varepsilon$ hver gang. (Hvorfor?) Dermed ville køretiden blive bounded af $T(n) \leq T((1-\varepsilon)n) + cn$. Det
- Her får vi

$$T(n) \leq cn + (1-\varepsilon)cn + (1-\varepsilon)^2 cn + \cdots = \left[1 + (1-\varepsilon) + (1-\varepsilon)^2 + \cdots\right.$$

Choosing a good splitter iii

- $$cn \leq \frac{1}{\varepsilon} \cdot cn$$
- Siden vi har en konvergent geometrisk serie.(ok hvad?)
- Det eneste vi rigtigt skal være opmærksomme på er en "off-center" splitter.
- For eksempel, hvis vi altid vælger minimumselementet som splitteren, ender vi måske op med et sæt i det rekursive kal der måske kun er ét element mindre hver gang, end det vi havde før.
- I dette eksempel, ville køretiden blive bounded af $T(n) \leq T(n-1) + cn$. Men det er et problem:
- $T(n) \leq cn + c(n-1) + c(n-2) + \cdots = \frac{cn(n+1)}{2} = \Theta(n^2)$

[1] ville der ikke være nogen grund til at have den her algoritme lol

Tilfældelige Splitters

- Til analysen (som kommer om lidt) vil vi vælge **splitteren** $a_i \in S$ uniformt tilfældigt.

Analyse af algoritmen i

- **Terminologitid!**
- Vi siger at algoritmen er i *fase j* når størrelsen af sættet vi snakker om er højest $n\left(\frac{3}{4}\right)^j$, men større end $n\left(\frac{3}{4}\right)^{j+1}$
- Vi vil forsøge at bounde tiden vi bruger i fase j:
- I en given ieration af algoritmen, siger vi at et element under overvejelse er *centralt* hvis mindst en fjerdedel af elementer er mindre end det, og mindst en fjerdedel af elementerne er større end det.
- Så, hvis a_c er det centrale element, så: $|a_j : a_j < a_c| \geq \frac{1}{4}n, |a_j : a_j > a_c| \geq \frac{1}{4}n$

Analyse af algoritmen ii

- Læg nu mærke til at hvis vi vælger et centralt element, så vil vi fjerne en fjerdedel af sættet, og det vil skrumpe med en faktor af **mindst** $\frac{3}{4}$, eller bedre.
- Og ved du hvad det bedste er? (Udover legetøj fra BR.)
- Halvdelen af tallene i et sæt er centrale! Så sandsynligheden for at vores tilfældelige splitter er central er $\frac{1}{2}$.

Theorem (13.7)

If we repeatedly perform independent trials of an experiments, each of which succeeds with probability $p > 0$, then the expected number of trials we need to perform until the first success is $\frac{1}{p}$.

Analyse af algoritmen iii

- Givet teorem 13.7 (Som, for reference, bare er expected value for bernoulli trials), er det expected antal af iterationer før et centralt element bliver valgt er 2. ($\frac{1}{\frac{1}{2}} = 2$)
- Det er stort set alt analyse vi har brug for.
- Lad X være et random variable, lig med antallet af skridt der skal tages af algoritmen. Vi kan skrive det som summen $X = X_0 + X_1 + \cdots$, hvor X_j er det forventede antal af skridt der bliver brugt af algoritmen i fase j.
- Når algoritmen er i fase j, har sættet højest størrelsen $n\left(\frac{3}{4}\right)^j$.
- Dermed er antallet af skridt nødvendigt for en iteration i fase j højest $cn(\frac{3}{4})^j$ for en konstant c.

Analyse af algoritmen iv

- Vi har lige argumenteret for at den forventede antal af iterationer brugt i fase j er højest to, og derfor har vi: $E[X_j] \leq 2cn(\frac{3}{4})^j$. Dermed kan vi bounde den sammenlagte køretid ved brug af linearity of expectation:
-

$$E[X] = \sum_j E[X_j] \leq \sum_j 2cn\left(\frac{3}{4}\right)^j = 2cn\sum_j \left(\frac{3}{4}\right)^j \leq 8cn$$

Analyse af algoritmen v

$$\sum_{n=1}^{\infty} a_1 (r)^{n-1} \text{ converges if and only if } -1 < r < 1$$

$$\sum_{n=1}^{\infty} a_1 (r)^{n-1} = \frac{a_1}{1-r}$$

Calcworkshop.com

(Geometric Series)

Analyse af algoritmen vi

- Dermed får vi 8cn fordi $\frac{1}{1-\frac{3}{4}} = \frac{1}{\frac{1}{4}} = 4$, dette ganges med 2 ($2cn$) og vi får $8cn$.

Theorem 13.18

Theorem (13.18)

The expected running time of `Select(n,k)` *is* $O(n)$

- Vi gjorde det! ☺
- Ifølge Jørgen er dette en Las Vegas algoritme. Den vil **altid** returnere det korrekte element, men med tilfældig køretid.
- I.e., en Monte Carlo algoritme er ikke altid korrekt, men har et bound på køretid. En Las Vegas algoritme er altid korrekt, men køretiden er en random variable.

Reflektion

- Hvad er en Las Vegas algoritme?
- Hvorfor er det vigtigt at alle elementer er distinkte i vores analyse?
- Hvilke aspekter af algoritmen var sværest at forstå? Hvorfor?
- Diskutér forskellet mellem valg af en dårlig og en god splitter

Quicksort

- Hurtig recap (Husker I DM507? Gode tider):
- Essensen af quicksort er: Pivots, Partitioning og Rekursion!
- Pivotelement: elementet som jeg gerne vil finde en position til.
- Vi tager alle elementer, og lægger dem i 2 partitions, alle mindre end pivotens værdi i én, alle større end i en anden (ligesom `Select(s,k)`).

Pseudokode

```
Quicksort(S):
If |S| ≤ 3 then
   Sort S
   Output the sorted list
Else
   Choose a splitter a_i ∈ S uniformly at random
   For each element a_j of S
      Put a_j in S⁻ if a_j < a_i
      Put a_j in S⁺ if a_j > a_i
   Endfor
   Recursively call Quicksort(S⁻) and Quicksort(S⁺)
   Output the sorted set S⁻, then a_i, then the sorted set S⁺
Endif
```

Quicksort Køretid i

- Præcis som ved median-findings-algoritmen, er worst-case køretiden dårlig.
- Hvis vi bliver ved med at vælge det mindste element som splitter, så er køretiden $T(n) \leq T(n-1) + cn = \Theta(n^2)$
- Til gengæld, hvis splitteren altid var medianen ville køretiden være:

$$T(n) \leq 2T(n/2) + cn = O(n \log n)$$

- Vi vil gerne vise at den *expected running time* (forventede køretid, i guess?) er bounded $O(n \log n)$, næsten lige så godt som hvis splitterne er perfekte (i.e., hver splitter er median).

Quicksort Køretid ii

- Vi bruger samme idé som tidligere, med den centrale splitter (hvor vi deler op i fire fjerdedele).
- Den centrale idé af en randomiseret `Quicksort` er at der er $\frac{1}{2}$ chance for at få en *central splitter*, som gør problemet signifikant nemmere, og hurtigere.

Modified Quicksort Pseudokode

```
Modified Quicksort(S):
If |S| ≤ 3 then
   Sort S
   Output the sorted list
Endif
Else
   While no central splitter has been found
      Choose a splitter a_i ∈ S uniformly at random
      For each element a_j of S
         Put a_j in S⁻ if a_j < a_i
         Put a_j in S⁺ if a_j > a_i
      Endfor
      If |S⁻| ≥ |S|/4 and |S⁺| ≥ |S|/4 then
         a_i is a central splitter
      Endif
   Endwhile
   Recursively call Quicksort(S⁻) and Quicksort(S⁺)
   Output the sorted set S⁻, then a_i, then the sorted set S⁺
Endif
```

Forskel i

- Forskellen her er måden hvorpå vi finder en splitter. (pivot)
- Hvis splittern er dårlig, i.e., hvis det ikke er den type splitter vi definerede tidligere (en central splitter), så smider vi den ud og finder en ny.
- Som vi så tidligere, er gennemsnittet af gange du skal gøre dette for at få en central splitter 2. (fordi $p(a_i$ er en central splitter$) = \frac{1}{2}$.
- Hver iteration af `While` loopet vælger en mulig splitter a_i, og bruger $O(|S|)$ tid på at splitte sættet og derefter vælge om a_i er central.

Forskel ii

Theorem (13.19)

The expected running time for the algorithn on a set S, excluding the time spent on recursive calls, is $O(|S|) = O(n)$

- Vi grupperer delproblemerne baseret på størrelse.
- Et delproblem er af type j hvis størrelsen af sættet under overvejse er højest $n\left(\frac{3}{4}\right)^j$ men større end $n\left(\frac{3}{4}\right)^{j+1}$.
- Ved (13.19) er den forventede køretid på et delproblem af type j, udelukket rekursive kald $O(n\left(\frac{3}{4}\right)^j)$
- For at bounde køretiden, skal vi først bounde antallet af delproblemer af type j.
- Når vi splitter, får vi to disjunkte sæt.

Theorem (13.20)

The number of type j subproblems created by the algorithm is at most $\left(\frac{4}{3}\right)^{j+1}$

- Der er højest $(\frac{4}{3})^{j+1}$ delproblemer af type j, og den forventede tid brugt på hver er $O(n\left(\frac{3}{4}\right)^{j})$
- Jeg forstår ikke helt hvorfor, men ifølge Jørgen:
- As subproblems of type j are disjoint, and each have size at least $\left(\frac{3}{4}\right)^{j+1} \cdot n$, then at most $\frac{n}{n \cdot \left(\frac{3}{4}\right)^{j+1}} = \left(\frac{4}{3}\right)^{j+1}$ subproblems of type j.

Forskel iv

- Af linearity of expectation, er den forventede tid brugt på delproblem af type j $O(n)$, fordi $O\left(n \cdot \left(\frac{3}{4}\right)^j \cdot \left(\frac{4}{3}\right)^{j+1}\right) = O\left(\frac{4}{3}n\right) = O(n)$
- Antallet af forskellige typer er derfor bounded af $\log_{\frac{4}{3}} n = O(\log n)$, hvilket giver os det bound vi ledte efter.

Theorem (13.21)

The expected running time of `Modified Quicksort` *is* $O(n \log n)$

Reflektion

- Hvad har vi lige snakket om?
- Hvorfor $O(|S|)$ per delproblem?
- Hvorfor får vi $O(n \log n)$?
- Hvorfor ikke $O(n^2)$?

Cormen Randomized Quicksort i

- Let X be the number of comparisons made by `RandomizedQuicksort` on a set S of n distinct numbers.
- Let $z_1 < z_2 < \cdots < z_n$ be the sorted order of the elements in S and let the random variable $X_{ij} = 1$ if z_i and z_j are compared in the algorithm and $X_{ij} = 0$ otherwise.
- Bid mærke i at Z_i og Z_j kun bliver sammenlignet hvis én af dem er pivoten, da kun pivoten bliver sammenlignet.
- Dermed:

$$X = \sum_{i \leq i < j \leq n} X_{ij}$$

- Lad $Z_{ij} = \{z_i, z_{i+1}, \cdots, z_j\}$

Cormen Randomized Quicksort ii

- I denne notation er Z_i og Z_j sammenlignet udelukkende hvis enten Z_j eller Z_i er valgt som pivot.
- Sandsynligheden for at dette sker er:

$$\frac{2}{|Z_{ij}|} = \frac{2}{j - i + 1}$$

- Siden X_{ij} er en indicator random variable, så: $E[X_{ij}] = \frac{2}{j-i+1}$ og dermed:

Cormen Randomized Quicksort iii

$$
\begin{aligned}
E[X] &= E[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} x_{ij}] \\
&= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}] \\
&= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\
&= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} \quad \text{take } k = j - i \\
&< 2 \cdot \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{1}{k} \quad \text{this is the harmonic number}
\end{aligned} \tag{1}
$$

Cormen Randomized Quicksort iv

$$
\begin{aligned}
&= 2n \cdot \sum_{i=1}^{n-1} H(n) \\
&< 2 \cdot \sum_{i=1}^{n-1} H(n) \qquad (2) \\
&= O(n \log n)
\end{aligned}
$$

Collecting Coupons

Coupon Collector i

- Brug af linearity of expectation.
- Antag at et mærke af morgenmadsprodukter har en gratis kupon i hvert boks.
- Der er n forskellige typer af kuponer.
- Som en loyal kunde, hvor mange bokse regner du med at du skal købe før du får en kupon af hver type?
- Klart, lower bound er n. Hvis du køber n bokse, og hver har en forskellige kupon, så er vi færdige.
- Lad X være en random variable lig med antallet af bokse du køber indtil du først har en kupon af hver type.

Coupon Collector ii

- Vi tænker på det som en process, der går et skridt tættere på målet, når du får en kupon du ikke har fået før.
- Målet er dermed at gå et skridt tættere på målet n gange (så du dermed er i mål)
- Givet, at du allerede har j typer af kuponer. I så fald er chancen for at du får den næste kupon $\frac{n-j}{n}$.
- Vi kan inddele processen i faser. Du er i fase j når du har fået j unikke kuponer.
- Lad X_j være det random variable der er lig med antallet af skridt du skal bruge i fase j. Så er $X = X_0 + X_1 + \cdots + X_{n-1}$.
- Så skal vi finde ud af $E[X_j]$ for hvert j.
- $E[X_j] = \frac{n}{n-j}$, siden det bare er en indicator random variable.

Coupon Collector iii

- Dermed, af linearity of expectation:

$$
\begin{aligned}
E[X] &= \sum_{j=0}^{n-1} E[X_j] \\
&= \sum_{j=0}^{n-1} \frac{n}{n-j} \\
&= n \sum_{j=0}^{n-1} \frac{1}{n-j} \\
&= n \sum_{i=1}^{n} \frac{1}{i} \\
&= nH(n) \\
&= \Theta(n \log n)
\end{aligned}
\tag{3}
$$

K-sat præ-viden

- Et **boolean variable** x er et variabel som kan tage to værdier: 0 og 1.
- $x_1 + x_2 + \cdots + x_k$ er 1 hvis mindst én $x_i = 1$, og 0 ellers.
- $\overline{x} = 1 - x, \overline{\overline{x}} = x$
- For hvert $x \in X$ er der to **literals**, x og $\overline{X}$
- En **clause** C over et sæt af boolean variabler X er summan af literals over variablerne fra X.
  - Størrelsen af clausen er antallet af literals den indeholder
- e.g. $C = \{u + \overline{v} + \overline{w}\}$, $u = 0, v = 0, w = 1$. $|C| = 3, C = 1$
- En **truth assignment** er en assignment af værdierne i sættet af variable X.

Satisfiability Problem (SAT) i

- Lad $X = \{x_1, \dots, x_n\}$ være et sæt af boolean variabler, og lad $C_1, \dots, C_m$ være en kollektion af clauses for hvert literal er over X.
- Bestem om der findes en truth assignment $t = (t_1, \dots, t_n)$ til variablerne i X således at værdien af hver clause er 1.
- Altså, kan en truth assignment findes hvorledes at $\mathcal{F} = C_1 * \cdots * C_m$ kan tage værdien 1.
- Afhængigt af om dette er muligt eller ej, siger vi at $\mathcal{F}$ er **satisfiable** eller **unsatisfiable**.
- Her står $*$ for **boolean multiplication**. (Kun $1 * 1 = 1$ alt andet er 0).

- For en given truth assignment $t = (t_1, \ldots, t_n)$ og en literal q, kalder vi $q(t)$ værdien af q når vi bruger truth assignment t. (I.e., hvis $q = \overline{x_3}$ og $t_3 = 1$, så $q(t) = 1 - 1 = 0$)
- Hvis alle clauses har det samme antal af k literals, så har vi et $k-$**SAT**. Generalt er $k-$**SAT** et meget svært problem at løse.

Theorem (A)

For every natural number k, every k-SAT formula with less than 2^k *clauses is satisfiable.*

Satisfiability Problem (SAT) iii

- Overvej en tilfældig truth assignment, hvilket sætter variablen x_i til 1 med sandsynlighed $\frac{1}{2}$ og 0 med sandsynlighed $\frac{1}{2}$ for $i = 1, 2, \ldots, n$. Med dette er hver truth assignment lige sandsynlig.
- Lad X være den random variable defineret på sættet af alle truth assignments hvilket, til en given truth assignment $t = (t_1, \ldots, t_n)$ giver værdien $X(t) =$ antallet af clauses iblandt $C_1, C_2, \ldots, C_m$ hvilke ikke er satisfied af t.
- Ligeledes, for hver clause C_i, lader vi den tilfældige varibel X_i tage værdien $X_i(t) = 1$ hvis t **ikke** satisfier C_i og $X_i(t) = 0$ hvis t satisfier C_i.
- Dermed $X(t) = X_1(t) + X_2(t) + X_3(t) + \cdots + X_m(t)$.

Satisfiability Problem (SAT) iv

- $E[X] = \sum_{i=1}^{m} E[X_i] = m2^{-k}$
- $m2^{-k} < 1$, siden $m < 2^k$.
- Af Markov's Inequality, $p(X \geq 1) \leq \frac{E(X)}{1} = E(X) < 1$, så $p(X = 0) > 0$.
- Det viser at der må være mindst én af de 2^ntruth assignments som satisfier alle m clauses.

Theorem (B)

Let $\mathcal{F} = C_1 * C_2 * \ldots * C_m$ be an instance of SAT. If we have $\sum_{i=1}^{m} 2^{-|C_i|} < 1$, then F is satisfiable.

Satisfiability Problem (SAT) v

Corollary

For all $\epsilon > 0$ there exists a polynomial algorithm for solving any instance of SAT over n variables $x_1, x_2, \ldots, x_n$ in which all clauses have size at least ϵn.

- Lad $\epsilon > 0$ være givet, og lad $\mathcal{F} = C_1 * C_2 * \cdots * C_m$ gå over variablerne $x_1, x_2, \ldots, x_n$ satisfy at $|C_i| \geq \epsilon n$ for hver $i \in \{1, 2, \ldots, m\}$. Antag først at $m < 2^{\epsilon n}$. Så har vi

$$\sum_{i=1}^{m} 2^{-|C_i|} \leq \sum_{i=1}^{m} 2^{-\epsilon n} < 1$$

- Den her er svær:(bliver nok sat over i misc bunken undtagen hvis en af jer forstår

A Randomized Approximation Algorithm for MAX 3-SAT

Max 3-SAT

- Det er ikke umuligt at du ikke kan finde en truth assignment der er satisfiable.
- I stedet kan du lede efter en der er så satisfiable som muligt.
- Det er *Maximum 3-Satisfiability Problem* eller (MAX 3-SAT).

- Antag at vi sætter hver variabel $x_1, \ldots, x_n$ uafhængigt til 0 eller 1 med sandsynlighed $\frac{1}{2}$ hver.
- Hvad er det forventede antal af clauses der bliver satisfied af en sådan assignment?
- Lad Z være et random variable lig med antallet af satisfied clauses.
- Lad Z_i være 0 eller 1. 1 hvis C_i er satisfied, 0 ellers. Dermed er $Z = Z_1 + Z_2 + \cdots + Z_k$.
- $E[Z_i]$ er lig med sandsynligheden at C_i er satisfied, og dette kan nemt udregnes.

- Sandsynligheden for at alle 3 variabler bliver sat på en måde der er forkert er $\left(\frac{1}{2}\right)^3 = \frac{1}{8}$.
- C_i er dermed satisfied med sandsynlighed $1 - \frac{1}{8} = \frac{7}{8}$, og dermed $E[Z_i] = \frac{7}{8}$.
- Ved brug af linearity of expectation, ser vi at $E[Z] = E[Z_1] + E[Z_2] + \cdots + E[Z_k] = \frac{7}{8}k$.
- Ifølge bogen: Since no assignment can satisfy more than k clauses we have the following guarantee. (Theorem 13.14). Jeg forstår det ikke.

Theorem (13.14)

Consider a 3-SAT formula, where each clause has three different variables. The expected number of clauses satisfied by a random assignment is within an approximation factor $\frac{7}{8}$ of optimal.

- For en random variable, må der være et punkt hvor den antager en eller anden værdi mindst lige så stor som forventningen.
- Vi har vist at for hver instance af 3-SAT, er der en tilfældig truth assignment som satisfier en $\frac{7}{8}$ brøk af alle clauses i forventingen, så der **må** eksistere en truth assignment der staisfier et antal af clauses der er mindst lige så stor som denne expectation.

- Det forstår jeg heller ikke:(

Theorem (13.15)

For every instance of $3-SAT$ there is a truth assignment that satisfies at least a $\frac{7}{8}$ fraction of all clauses.

- Jeg forstår *slet* ikke resten. Misc ting?