RAD (1) RQS, Min-Cut

Monte Carlo & Las Vegas

Monte Carlo

Some probability of error. Running time may be random or not.
Two-sided error: non-zero probability of failure when it outputs either true or false

One-sided error: probability of failure is 0 for at least one of the two possible outputs (true/false)

Las Vegas

Always returns correct answer.

Running time is a random variable.

A Las Vegas algorithm is by definition a Monte Carlo algorithm with error probability 0.

Random Quicksort

fordel er at den ikke har svaghed overfor input

1: **function** $\textsc{RandQS}(S)$
   ▷ Assumes all elements in S are distinct.
2: **if** $|S| \leq 1$ **then**
3: **return** S
4: **else**
5: Pick pivot $x \in S$, uniformly at random
6: $L \leftarrow \{y \in S \mid y < x\}$
7: $R \leftarrow \{y \in S \mid y > x\}$
8: **return** $\textsc{RandQS}(L)+[x]+\textsc{RandQS}(R)$

(Line 6–7 uses $|L| + |R| = |S| - 1$ comparisons)

What is the expected number of comparisons?

Lad være et sorteret array: $[S(1), \ldots, S(n)]$:= RandQS(S).
Det er efter man har kaldt randomized quicksort.

For $i < j$ lad X_{ij} være antallet af gange $S_{(i)}$
og $S_{(j)}$ er blevet sammenlignet. $X_{ij} \in \{0, 1\}$ den er en indikator variabel fordi 2 elementer bliver kun sammenlignet når det ene element er pivot element og når vi så laver et rekursivt kald vil der blive lavet 2 del-lister fordi vi laver partition, ingen af del-listererne kommer til at indeholde pivot elementet.

Number of comparisons = $X = \sum_{i<j} X_{ij}$
hver X_{ij} er en indikator variabel

$$X = \sum_{i<j} X_{ij} = \sum_{1<i<j<n'} X_{ij} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$$

Vi kigger på alle j der er større end i dvs vi kigger på $j = i + 1$ op til n. vi summer til n-1 i den anden sum ellers ville der ikke være nogen j'er der er større end i.

Linearity of expectation
$\mathbb{E}[\#Comparisons] = \mathbb{E}[X] = \mathbb{E}[\sum_{i<j} X_{ij}] = \sum_{i<j} \mathbb{E}[X_{ij}]$

lad p_{ij} være sandsynligheden for at $S_{(i)}$ og $S_{(j)}$ bliver sammenlignet, i.e., $p_{ij} = Pr[X_{ij} = 1]$. så gælder der

$$\mathbb{E}[X_{ij}] = \sum_{x} xPr[X_{ij} = x] =$$

Det er en indikator variabel! forventningen er sandsynligheden for den er 1.

$$0 \cdot Pr[X_{ij} = 0] + 1 \cdot Pr[X_{ij} = 1] = p_{ij}$$

Derved gælder der
$\mathbb{E}[\#Comparisons] = \mathbb{E}[X] = \mathbb{E}[\sum_{i<j} X_{ij}] = \sum_{i<j} \mathbb{E}[X_{ij}] = \sum_{i<j} p_{ij}$

Bevis

Hvert rekursive kald til RQS returnerer en sorteret liste $[S_{(a)}, \ldots, S_{(b)}]$. lad $x = S_{(c)}$ være pivot i det rekursive kald.

Formod at i og j er i blandt en af indexerne, her har vi en sorteret rækkefølge: $a \leq i < j \leq b$.

$S_{(a)}$	$\cdots$	$S_{(i)}$	$\cdots$	$S_{(j)}$	$\cdots$	$S_{(b)}$

Hvordan kan vi vælge pivot element c:

Case 1: $c < i \; eller \; c > j$: $S_{(i)}$ og $S_{(j)}$ ikke sammenlignet nu, men sammen i rekursions kald. de kan bliver sammenlignet i undertræ

Case 2: $i < c < j$: $S_{(i)}$ og $S_{(j)}$ aldrig sammenlignet. pivot ligger i mellem i og j. så ryger i og j ned i hvert sit undertræ. derfor kan de aldrig sammenlignes.

Case 3: $c = i \; eller \; c = j$: $S_{(i)}$ og $S_{(j)}$ sammenlignet en gang.

derved, er p_{ij} den betinget sandsynlighed for at vælge $S_{(i)}$ eller $S_{(j)}$ som pivot, givet at pivotet bliver valgt uniformt tillfældigt i $\{S_{(i)}, S_{(i+1)}, \ldots, S_{(j)}\}$: altså som vi ser i case 3

sandsynligheden for at vi vælger pivot til at være $S_{(i)}$ eller $S_{(j)}$ givet vi har valgt vores pivot i blandt $\{S_{(i)}, S_{(i+1)}, \ldots, S_{(j)}\}$, hvis vi vælger udfra den liste ved vi at de ikke bliver sammenlignet i den nuværende rekursion:

$$p_{ij} = Pr[c \in i, j | c \in i, i+1, \cdots, j] =$$

$$\frac{|\{i, j\}|}{|\{i, i+1, \cdots, j\}|} =$$

Der er 2 muligheder for c og vi skal dividere med antallet af muligheder for c i denne mængde $\{i, i+1, \cdots, j\}$

$$\frac{2}{j-i+1}$$

Det følger at
$\mathbb{E}[\#Comparisons] = \sum_{i<j} p_{ij} = \sum_{i<j} \frac{2}{j-i+1}$

$$\mathbb{E}[X] = \sum_{i<j} \frac{2}{j-i+1} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} =$$

$$\sum_{i=1}^{n-1} (\frac{2}{2} + \frac{2}{3} + \frac{2}{4} + \cdots + \frac{2}{n-i+1}) <$$

Da vi er intresseret i en øvregrænse på køretiden lader vi brøken starte på 1 og lader den gå op til n. derved får vi flere brøker med da vi både har 2/1 med og $n > n - i + 1$

$$\sum_{i=1}^{n} (\frac{2}{1} + \frac{2}{2} + \frac{2}{3} + \cdots + \frac{2}{n}) =$$

Den inderste sum kan forsimples.

$$2 \sum_{i=1}^{n} \sum_{j=1}^{n} 1/j =$$

$H_n = \sum_{k=1}^{n} \frac{1}{k}$ kaldes det n'te harmoniske nummer.

$$2 \sum_{i=1}^{n} H_n =$$

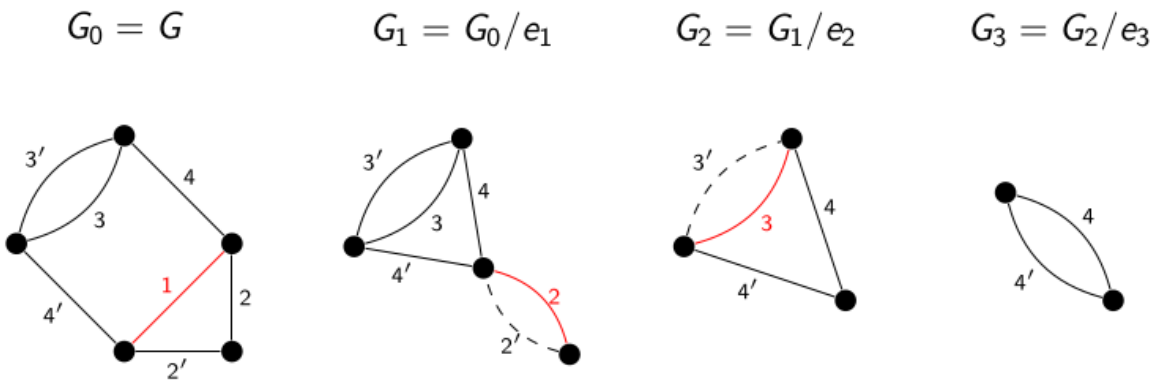
Proposition B.4: $H_n = \ln n + \Theta(1) \in O(\log n)$
Derved får vi

$$2nH_n \in O(n \log n)$$

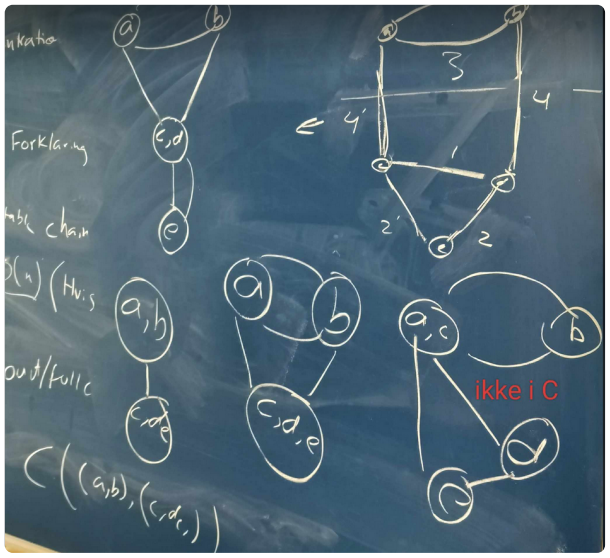
Når $|S| = n$, det forventet antal af sammenligninger i RandQS(S) højst O(n log n) for ethvert input.

Min-cut

Problem: Given a connected graph $G = (V, E)$
Find smallest C ⊆ E that splits G. C is called a min-cut, and λ(G) := |C| is the min-cut size (also called the edge connectivity of G).



C er et set af min cuts. Den her algoritme vælger en tillfældig edge og fjerner den (trækker den sammen), der er derfor en mulighed for at den fjerner en edge hvilket resultere at de vertexes vi ender med ikke er i C, dvs vi har en graf der ikke er i min-cut.



her ser vi at hvis vi trækker 1 sammen får vi nu en knude c,d. vi kan se at c og d skal være i samme knude så vi har stadig min cut, men hvis vi istedet trak 4' sammen får vi a,c som ikke er i min-cut og defor fejler algoritmen.

Da vi vælger tillfældige knuder er der self en chance for at vi f.eks. vælger 4'. Vi er derfor nødt til at køre algoritmen et hvis antal gange. Der gælder sandsynligheden for vores min cut overlever er:

$$\Pr[\text{ some min-cut is returned }] \geq \Pr[C \text{ is returned }] \geq \frac{2}{n(n-1)}$$

hvis vi kalder RandMinCut(G) $t\frac{n(n-1)}{2}$ gange

$$\begin{aligned}\Pr[\text{ not a min-cut }] &\leq \left(1+(-\frac{2}{n(n-1)})\right)^{t\frac{n(n-1)}{2}} \\ &\leq \left(e^{-\frac{2}{n(n-1)}}\right)^{t\frac{n(n-1)}{2}} \\ &= e^{-t}\end{aligned}$$

bruger at $1+x \leq e^x$
bruger at $(x^a)^b = x^{a\cdot b}$
derfor bliver $(e^{-\frac{2}{n(n-1)}})^{t\frac{n(n-1)}{2}} = e^{(-\frac{2}{n(n-1)})\cdot(t\frac{n(n-1)}{2})} = e^{-(t\frac{2n(n-1)}{2n(n-1)})} = e^{-t} = \frac{e}{t}$

I hvert kald, gælder der at sandsynligheden for at min-cut ikke returneres er højst e^{-t}. hvis vi nu vælger e.g. t = 21 reduceres fejl sandsynligheden til 1 ud af en milliard.

vi vælger altså mellem høj sucess sandsynlighed og køretid.

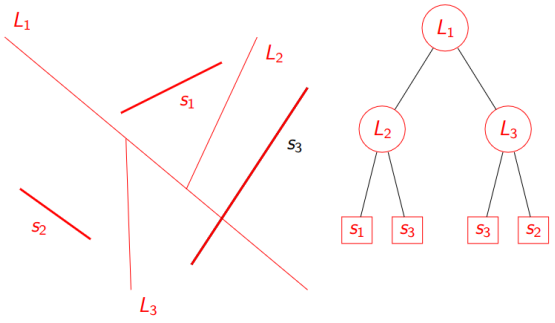
Binary planar partitions (ekstra)

Given a set $S = \{s_1, s_2, \ldots, s_n\}$ of non-intersecting line segments in the plane, we wish to find a binary planar partition such that every region in the partition contains at most one line segment (or a portion of one line segment).

Binary Planar Partitions

Given non-intersecting line segments $S = \{s_1, \ldots, s_n\}$ in the plane, construct a binary tree where:

- Each node v has a region $r(v)$ of the plane.
- Each internal node v is associated with a line $\ell(v)$ that intersects $r(v)$ and partitions it into the regions of its two children.
- The root is the whole plane.
- For every leaf v, $r(v)$ contains at most one s_i.



If every line used in a BPP contains one of the segments, it is called an autopartition.

Algorithm RandAuto:

Input: A set $S = \{s_1, s_2, \ldots, s_n\}$ of non-intersecting line segments.

Output: A binary autopartition P_π of S.

1. Pick a permutation π of $\{1, 2, \ldots, n\}$ uniformly at random from the $n!$ possible permutations.
2. **while** a region contains more than one segment, cut it with $l(s_i)$ where i is first in the ordering π such that s_i cuts that region.

The expected size of the returned autopartition is
O(n log n) for any input of size n.