Randomized Algorithms

Hafsteinn Einarsson
CADMO - ETH Zürich

Randomized algorithms

Las Vegas: Always returns ‘failed’ or correct answer.

Monte Carlo: Can return a wrong answer but with small probability.

p = q?

$$p(x) = (x-7)(x-3)(x-1)(x+2)(2\,x+5)$$
$$q(x) = 2\,x^5 - 13\,x^4 - 21\,x^3 + 127\,x^2 + 121\,x - 210$$

Multiplying out factors of $p(x)$ can be done in $O(d^2)$ (where d is the degree of the polynomial and we assume integer multiplication is in constant time). Evaluating $p(x)$ requires only $O(d)$ operations. Can we compare if $p(x) = q(x)$ in $O(d)$ time?

Is your solution Monte Carlo or Las Vegas?

Given a number *n*,
how can we determine
if it's prime?

Simple intuitive solution:
Try all relevant divisors

```cpp
bool is_prime(int n) {
    if (n < 2) return false;
    if (n < 4) return true;
    if (n % 2 == 0 || n % 3 == 0) return false;
    if (n < 25) return true;
    int s = static_cast<int>(sqrt(static_cast<double>(n)));
    for (int i = 5; i <= s; i += 6)
        if (n % i == 0 || n % (i + 2) == 0) return false;
    return true; }
```

This implementation uses the fact that all primes are either of the form 6*k*+1 or 6*k*-1 for some integer *k* (except for 2 and 3).

Runtime? O(√n)

Other ways?

Wilson's theorem

p is prime if and only if
$(p-1)! = -1 \pmod{p}$

Theoretically practical
not computationally

How can we be faster than $O(\sqrt{n})$?

Miller Rabin Primality Test

Theorem (Miller, Rabin)

If p is a prime, let s be the maximal power of 2 dividing p-1, so that $p\text{-}1=2^s d$ and d is odd. Then for any $1 \leq n \leq p\text{-}1$, one of two things happen:

$n^d = 1 \bmod p$ or
$n^{2^j d} = -1 \bmod p$ for some $0 \leq j < s$.

But if p is not prime then for any $1 \leq n \leq p\text{-}1$ the conditions fails with probability at least 3/4.

Let's try

$n = 91 \quad n-1 = 90 = 2*45$

Pick a number between 1 and 90 (exclusive)

Ok, so you picked 84:

Now, $84^{45} = 70 \pmod{91} \neq 1$

and $84^{2*45} = 77 \pmod{91} \neq -1$

So 84 is a witness that 91 is composite

What are the prime factors of 91?

Let's try again

$n = 91 \quad n\text{-}1 = 90 = 2*45$

Pick a number between 1 and 90 (exclusive)

Ok, so you picked 53:

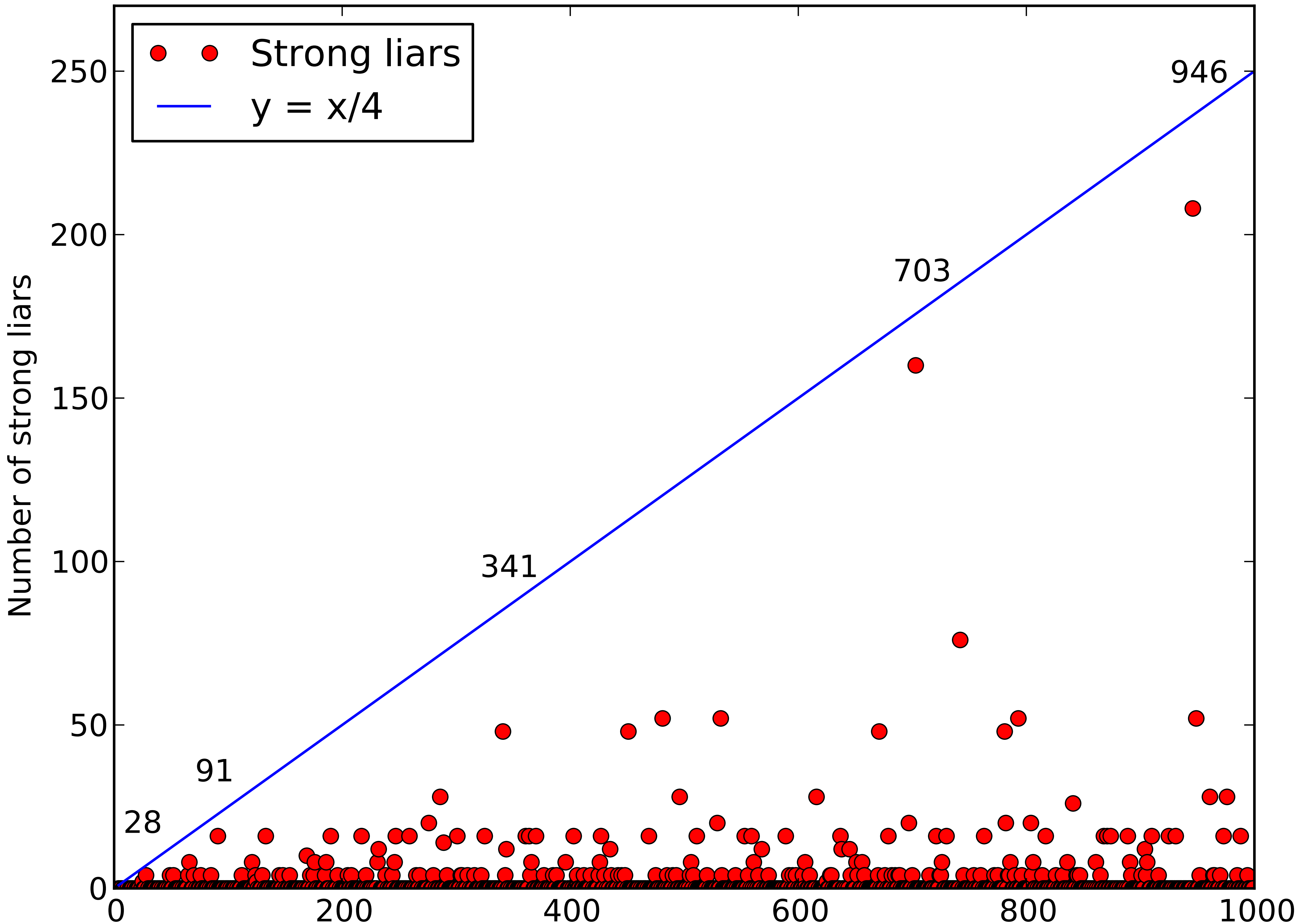
Now, $53^{45} = 1 \pmod{91}$

and $53^{2*45} = 1 \pmod{91} \neq -1$

So 53 is an example of a strong liar

But fortunately there are not many strong liars

Number of strong liars



Theorem (Miller, Rabin)

If p is a prime, let s be the maximal power of 2 dividing p-1, so that $p-1=2^s d$ and d is odd. Then for any $1 \leq n \leq p-1$, one of two things happen:

$n^d = 1 \bmod p$ or
$n^{2^j d} = -1 \bmod p$ for some $0 \leq j < s$.

But if p is not prime then for any $1 \leq n \leq p-1$ the conditions fails with probability at least 3/4.

Proof

First, if

$$x^2 \equiv 1 \mod p$$

then

$$(x-1)(x+1) \equiv 0 \mod p.$$

Now by Fermat's little theorem

$$a^{p-1} \equiv 1 \mod p.$$

By taking a square root repeatedly we either end up with $a^d \equiv 1$ or at some point we have $a^{2^i d} \equiv -1$ and we're done.

```python
import random

def decompose(n):
   exponentOfTwo = 0

   while n % 2 == 0:
      n = n/2
      exponentOfTwo += 1

   return exponentOfTwo, n

def isWitness(possibleWitness, p, exponent, remainder):
   possibleWitness = pow(possibleWitness, remainder, p)

   if possibleWitness == 1 or possibleWitness == p - 1:
      return False

   for _ in range(exponent):
      possibleWitness = pow(possibleWitness, 2, p)

      if possibleWitness == p - 1:
         return False

   return True

def probablyPrime(p, accuracy=100):
   if p == 2 or p == 3: return True
   if p < 2: return False

   exponent, remainder = decompose(p - 1)

   for _ in range(accuracy):
      possibleWitness = random.randint(2, p - 2)
      if isWitness(possibleWitness, p, exponent, remainder):
         return False

   return True
```

Runtime of Miller-Rabin with *k* repetitions?

Naive: $O(k \cdot \log(n)^3)$

With FFT multiplication: $O(k \cdot \log(n)^2)$

For efficiency

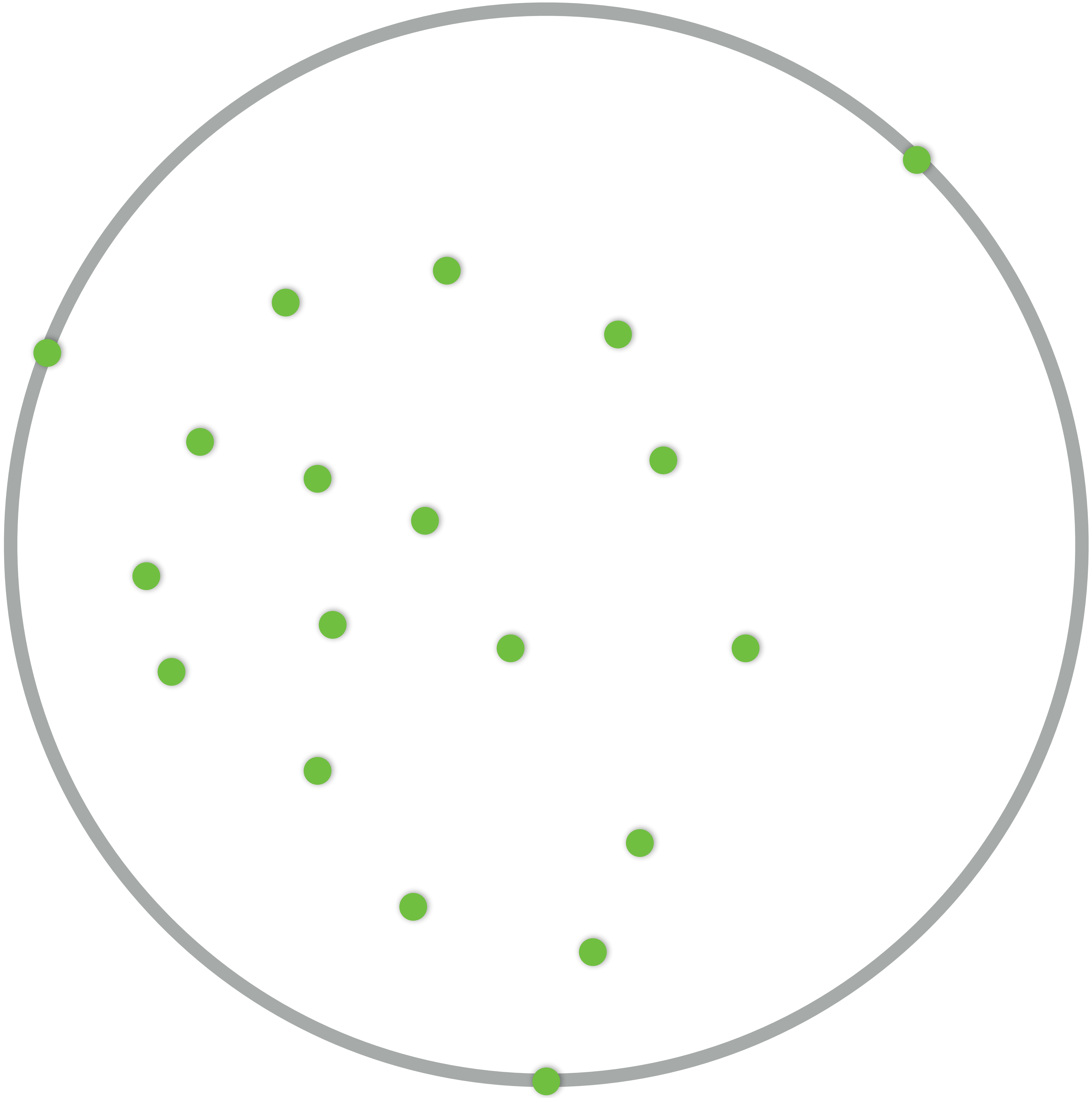
It has been verified that

- if $n < 1{,}373{,}653$, it is enough to test 2 and 3;
- if $n < 9{,}080{,}191$, it is enough to test 31 and 73;
- if $n < 4{,}759{,}123{,}141$, it is enough to test 2, 7, and 61;
- if $n < 1{,}122{,}004{,}669{,}633$, it is enough to test 2, 13, 23, and 1662803;
- if $n < 2{,}152{,}302{,}898{,}747$, it is enough to test 2, 3, 5, 7, and 11;
- if $n < 3{,}474{,}749{,}660{,}383$, it is enough to test 2, 3, 5, 7, 11, and 13;
- if $n < 341{,}550{,}071{,}728{,}321$, it is enough to test 2, 3, 5, 7, 11, 13, and 17.

So that's it for Miller Rabin, questions?

FORODWAITH
ERED MITHRIN
Withered Heath
IRON HILLS
RHÛN
Erebor
The Lonely Mountain
Esgaroth
the Long Lake
MIRKWOOD
Icebay of Forochel
The Lost
Himling
ERED LUIN
Forlindon
Forlond
Gulf of Lhûn
Mithlond
Harlond
Harlindon
Blue Mountains
Fornost
North Downs
Bree
East-West Road
South Downs
Weathertop
Last Bridge
Rivendell
High Pass
Old Ford
Carrock
Old Forest Road
Sarn Ford
Baranduin
Tharbad
MINHIRIATH
CARDOLAN
Nîn-in-Eilph
(Swanfleet)
Moria Gate
Eregion
(Hollin)
Mitheithel
East Bight
FANGORN
ROHAN
West Emnet
East Emnet
WESTFOLD
Helm's Deep
Sarn Gebir
EASTFOLD
Dagorlad (Battle Plain)
Morannon
ERED LITHUI (Ash Mountains)
ERED
NIMRAIS
Erech
Minas Tirith
Anórien
Pinnath Gelin
Anfalas (Langstrand)
Andrast (Ras Morthil)
Edhellond
GONDOR
Lebennin
Tolfalas
Ethir Anduin
South Gondor
(now a debatable and desert land)
Harad Road
Harnen
NEAR HARAD
Núrnen
The Realm of
Middle Earth
Miles
50
100
150
200

Smallest enclosing circle problem



Unique solution defined by 2 or 3 points.

Naive method: $O(n^4)$
How?

Welzl's Algorithm

Algorithm MINIDISC(P)

Input. A set P of n points in the plane.

Output. The smallest enclosing disc for P.

Compute a random permutation $p_1, \ldots, p_n$ of P.
Let D_2 be the smallest enclosing disc for $\{p_1, p_2\}$.
for $i \leftarrow 3$ **to** n
    do if $p_i \in D_{i-1}$
        then $D_i \leftarrow D_{i-1}$
        else $D_i \leftarrow$ MINIDISCWITHPOINT($\{p_1, \ldots, p_{i-1}\}, p_i$)
return D_n

MINIDISCWITHPOINT(P, q)

Input. A set P of n points in the plane, and a point q such that there exists an enclosing disc for P with q on its boundary.

Output. The smallest enclosing disc for P with q on its boundary.

1. Compute a random permutation $p_1, \dots, p_n$ of P.
2. Let D_1 be the smallest disc with q and p_1 on its boundary.
3. **for** $j \leftarrow 2$ **to** n
4. **do if** $p_j \in D_{j-1}$
5. **then** $D_j \leftarrow D_{j-1}$
6. **else** $D_j \leftarrow$ MINIDISCWITH2POINTS($\{p_1, \dots, p_{j-1}\}, p_j, q$)
7. **return** D_n

More efficient: Use the old permutation

MINIDISCWITH2POINTS(P, q_1, q_2)

Input. A set P of n points in the plane, and two points q_1 and q_2 such that there exists an enclosing disc for P with q_1 and q_2 on its boundary.

Output. The smallest enclosing disc for P with q_1 and q_2 on its boundary.

1. Let D_0 be the smallest disc with q_1 and q_2 on its boundary.
2. **for** $k \leftarrow 1$ **to** n
3. **do if** $p_k \in D_{k-1}$
4. **then** $D_k \leftarrow D_{k-1}$
5. **else** $D_k \leftarrow$ the disc with q_1, q_2, and p_k on its boundary
6. **return** D_n

Algorithm MINIDISC(P)

Input. A set P of n points in the plane.

Output. The smallest enclosing disc for P.

Compute a random permutation $p_1, \dots, p_n$ of P.
Let D_2 be the smallest enclosing disc for $\{p_1, p_2\}$.
for $i \leftarrow 3$ **to** n
  do if $p_i \in D_{i-1}$
    then $D_i \leftarrow D_{i-1}$
    else $D_i \leftarrow$ MINIDISCWITHPOINT($\{p_1, \dots, p_{i-1}\}, p_i$)
return D_n

MINIDISCWITHPOINT(P, q)

Input. A set P of n points in the plane, and a point q such that there exists an enclosing disc for P with q on its boundary.

Output. The smallest enclosing disc for P with q on its boundary.

Compute a random permutation $p_1, \dots, p_n$ of P.
Let D_1 be the smallest disc with q and p_1 on its boundary.
for $j \leftarrow 2$ **to** n
  do if $p_j \in D_{j-1}$
    then $D_j \leftarrow D_{j-1}$
    else $D_j \leftarrow$ MINIDISCWITH2POINTS($\{p_1, \dots, p_{j-1}\}, p_j, q$)
return D_n

MINIDISCWITH2POINTS(P, q_1, q_2)

Input. A set P of n points in the plane, and two points q_1 and q_2 such that there exists an enclosing disc for P with q_1 and q_2 on its boundary.

Output. The smallest enclosing disc for P with q_1 and q_2 on its boundary.

Let D_0 be the smallest disc with q_1 and q_2 on its boundary.
for $k \leftarrow 1$ **to** n
  do if $p_k \in D_{k-1}$
    then $D_k \leftarrow D_{k-1}$
    else $D_k \leftarrow$ the disc with q_1, q_2, and p_k on its boundary
return D_n

Expected runtime?

$O(n)$

Algorithm MINIDISC(P)

Input. A set P of n points in the plane.

Output. The smallest enclosing disc for P.

Compute a random permutation $p_1, \ldots, p_n$ of P.
Let D_2 be the smallest enclosing disc for $\{p_1, p_2\}$.
for $i \leftarrow 3$ **to** n
  do if $p_i \in D_{i-1}$
    then $D_i \leftarrow D_{i-1}$
    else $D_i \leftarrow$ MINIDISCWITHPOINT($\{p_1, \ldots, p_{i-1}\}, p_i$)
return D_n

MINIDISCWITHPOINT(P, q)

Input. A set P of n points in the plane, and a point q such that there exists an enclosing disc for P with q on its boundary.

Output. The smallest enclosing disc for P with q on its boundary.

Compute a random permutation $p_1, \ldots, p_n$ of P.
Let D_1 be the smallest disc with q and p_1 on its boundary.
for $j \leftarrow 2$ **to** n
  do if $p_j \in D_{j-1}$
    then $D_j \leftarrow D_{j-1}$
    else $D_j \leftarrow$ MINIDISCWITH2POINTS($\{p_1, \ldots, p_{j-1}\}, p_j, q$)
return D_n

MINIDISCWITH2POINTS(P, q_1, q_2)

Input. A set P of n points in the plane, and two points q_1 and q_2 such that there exists an enclosing disc for P with q_1 and q_2 on its boundary.

Output. The smallest enclosing disc for P with q_1 and q_2 on its boundary.

Let D_0 be the smallest disc with q_1 and q_2 on its boundary.
for $k \leftarrow 1$ **to** n
  do if $p_k \in D_{k-1}$
    then $D_k \leftarrow D_{k-1}$
    else $D_k \leftarrow$ the disc with q_1, q_2, and p_k on its boundary
return D_n

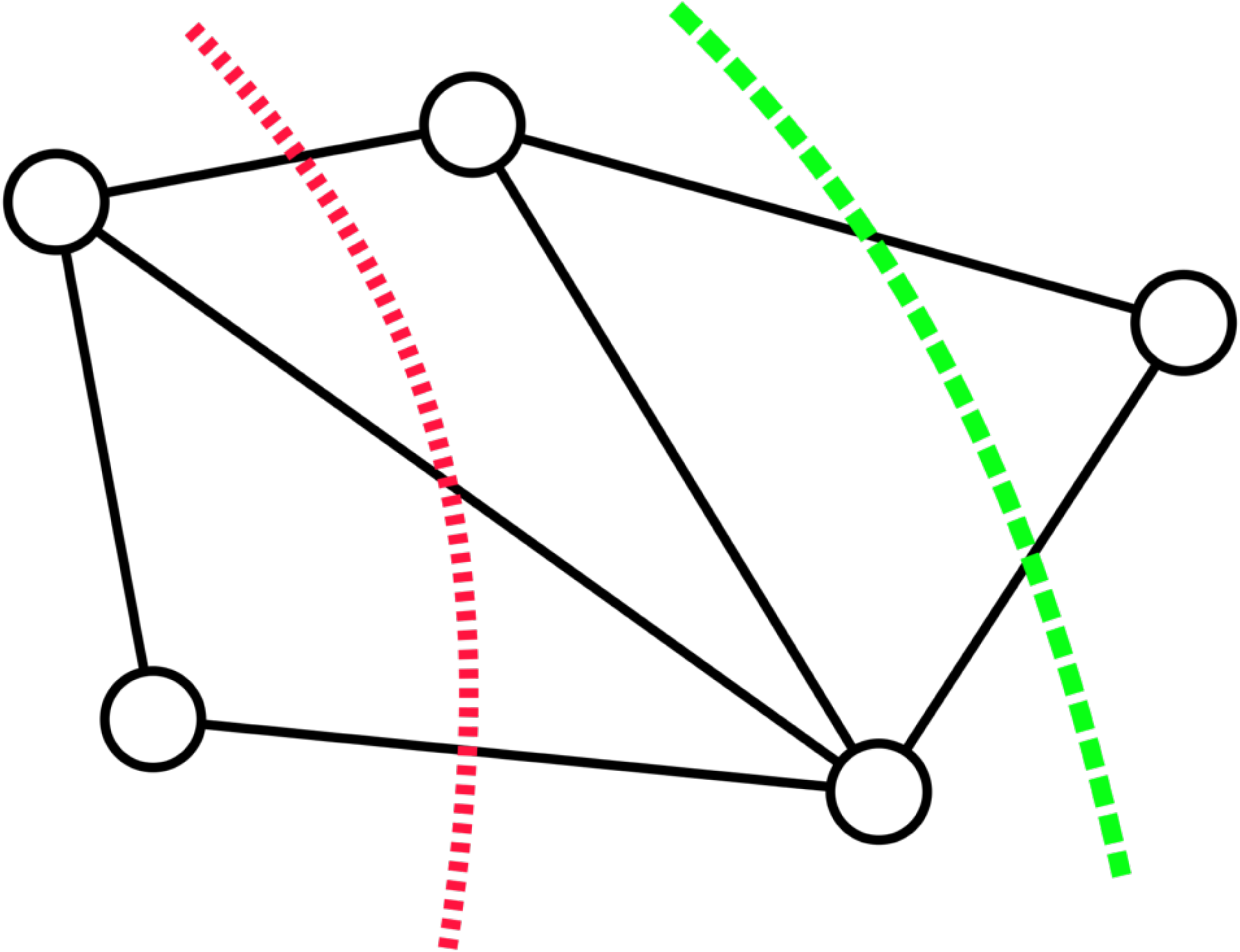
The probability of needing another function call in iteration *i* is only O(1/i).

So if you can show that MiniDiscWithPoint runs in expected time O(*n*) you are done.

This algorithm can be further optimized by moving vertices on the circle to the beginning of the permutation.

Questions regarding Welzl's algorithm?

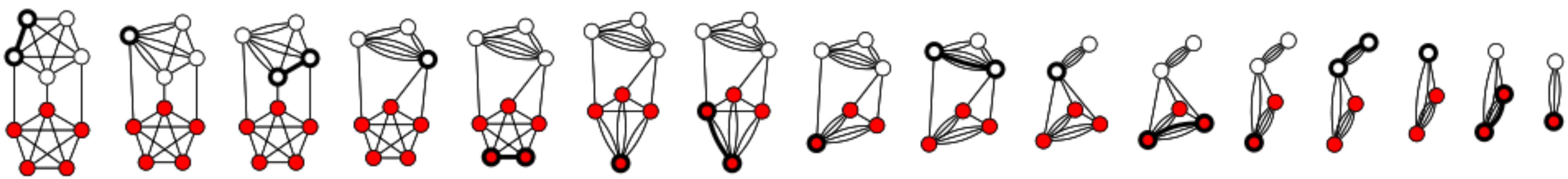
How do you partition the vertices of a graph into two non-empty subsets S and T such that the number of edges between S and T is minimized?

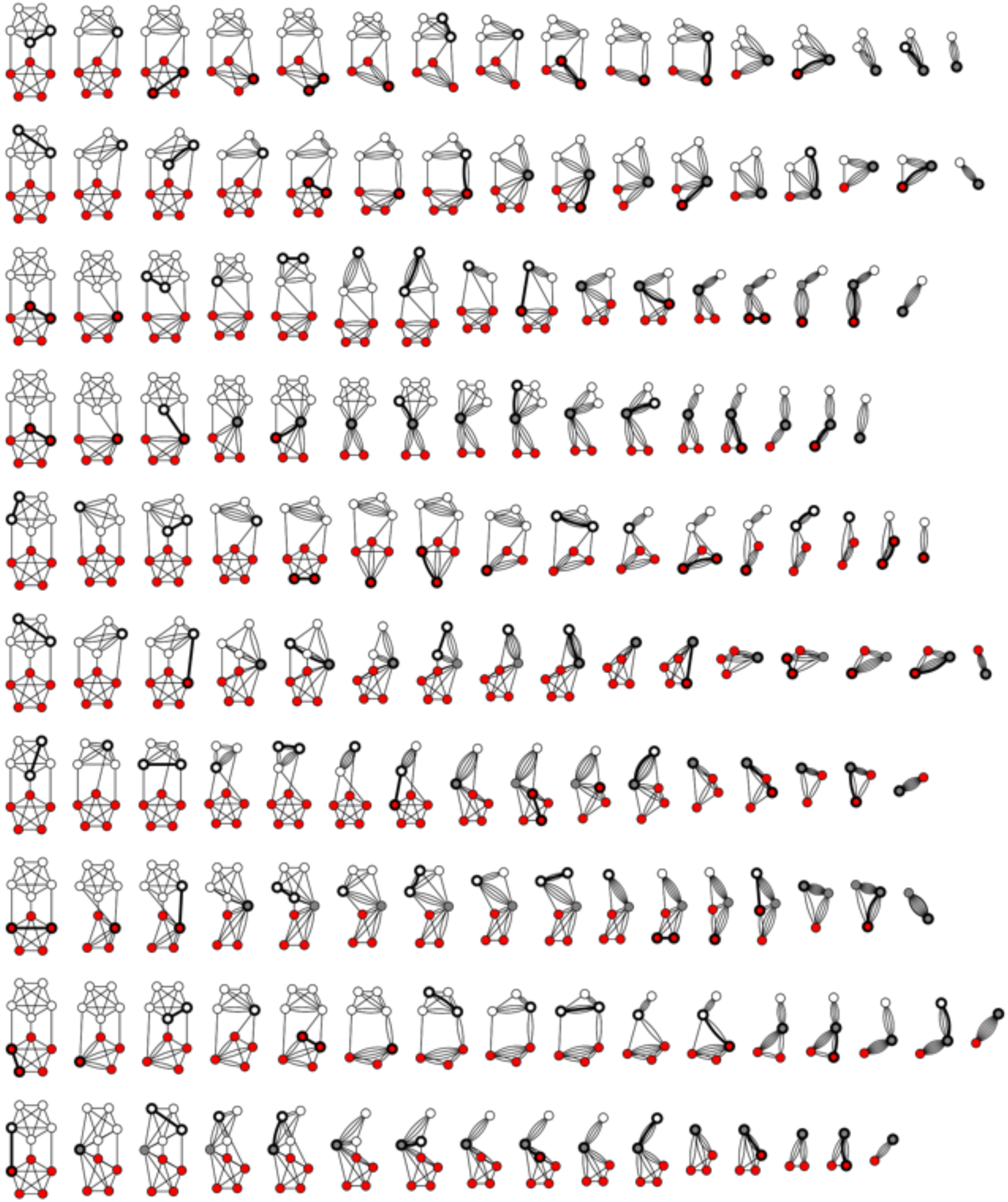


Karger's algorithm

Starting from the input graph $G = (V, E)$, repeat the following process until only two vertices remain:

1. Choose an edge $e = (u, v)$ uniformly at random from E.
2. Set $G = G/e$.





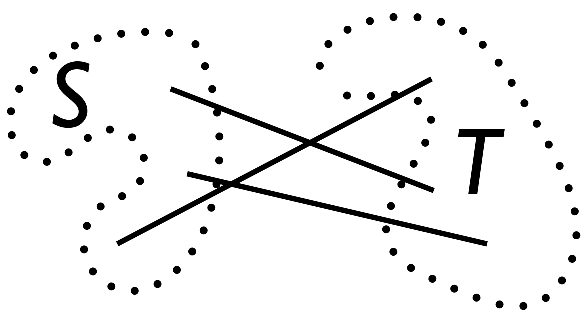
What's the probability of success?

Given any min cut (S, T) of a graph G on n vertices, Karger's algorithm outputs (S, T) with probability at least $\binom{n}{2}^{-1}$.

Given any min cut (S, T) of a graph G on n vertices, Karger's algorithm outputs (S, T) with probability at least $\binom{n}{2}^{-1}$.

Since there are $2^{n-1} - 1$ cuts in every graph (why?) the algorithm is much more efficient than selecting a cut at random which has success probability at most $\frac{\binom{n}{2}}{2^{n-1}-1}$ (why?).

Given any min cut (S, T) of a graph G on n vertices, Karger's algorithm outputs (S, T) with probability at least $\binom{n}{2}^{-1}$.



Let's assume the min cut (S, T) has k edges.

Then the minimum degree of the graph is at least k (why?) and thus $|E| \geq nk/2$.

$$\frac{k}{|E|} \leq \frac{k}{nk/2} = \frac{2}{n}$$

Now an edge from the cut is not chosen with probability at least

$$(1 - k/|E|) \geq (1 - 2/n).$$

Now if p_n is the probability that the algorithm avoids the cut in an n vertex graph then it satisfies the recurrence $p_n \geq \left(1 - \frac{2}{n}\right) p_{n-1}$ with $p_2 = 1$, let's expand:

$$\begin{aligned} p_n &\geq \prod_{i=0}^{n-3} \left(1 - \frac{2}{n-i}\right) \\ &= \prod_{i=0}^{n-3} \left(\frac{n-i-2}{n-i}\right) \\ &= \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdots \frac{2}{4} \cdot \frac{1}{3} \\ &= \binom{n}{2}^{-1}. \end{aligned}$$

Now the algorithm is successful with probability at least $\binom{n}{2}^{-1}$, how many times should we repeat it?

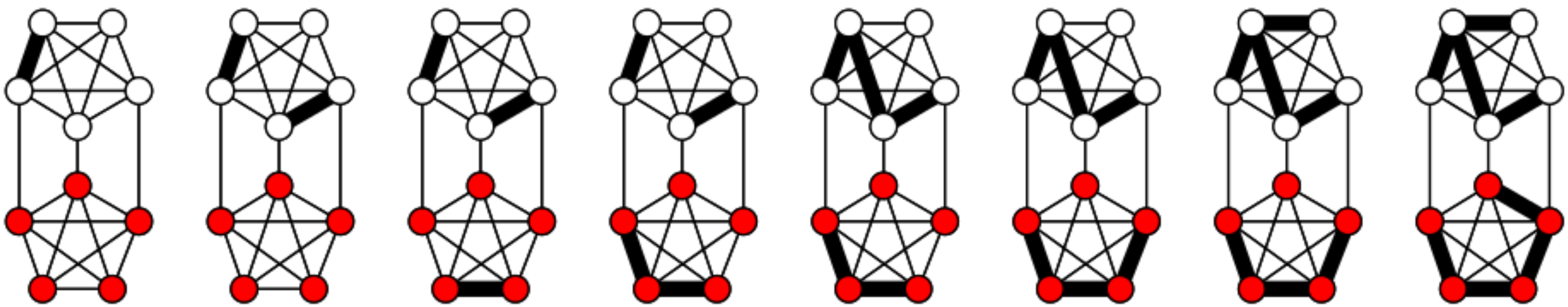
We want a low failure probability. Let's assume that we repeat it T times, then the probability of failure is at most

$$\left(1 - \binom{n}{2}^{-1}\right)^{T} \leq e^{-T\cdot\binom{n}{2}^{-1}} \leq \frac{1}{n}$$

if $T = \binom{n}{2} \ln n$.

What’s the runtime?

Doing an edge contraction can require O(n) operations for an adjacency list or an adjacency matrix leading to a total running time of $O(n^2)$. So here's another way to see it:



What algorithm does this remind you of?

Now using Kruskal and random edge weights we can run the algorithm in $O(|E|\log|E|)$. But we still need at least $O(n^2\log n)$ repetitions leading to a total running time of $O(n^4\log n^2)$. Can we do better?

Karger Stein

Improved algorithm: From a multigraph G, if G has at least 6 vertices, repeat twice:

1. run the original algorithm down to $n/\sqrt{2}+1$ vertices.
2. recurse on the resulting graph.

For a specific minimum cut (S, T) denote by $p_{n,t}$ the probability that the algorithm has not ruined (S, T) after t edge contractions. Like before we can solve $p_{n,t} \geq \binom{t}{2} / \binom{n}{2}$ which for $t = n/\sqrt{2} + 1$ is the point where it becomes less than $1/2$.

The running time now satisfies

$$T(n) = 2T(n/\sqrt{2}) + O(n^2) = O(n^2 \log n)$$

and the success probability satisfies

$$P(n) \geq 1 - \left(1 - \frac{1}{2}P(n/\sqrt{2} + 1)\right)^2 = \Omega\left(\frac{1}{\log n}\right).$$

Now since the success probability is $\Omega\left(\frac{1}{\log n}\right)$ we can simply repeat the algorithm $c \cdot (\log n)^2$ times to get a success probability of at least $1 - \frac{1}{n^c}$.

Thank you for your attention!
Questions?