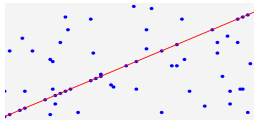
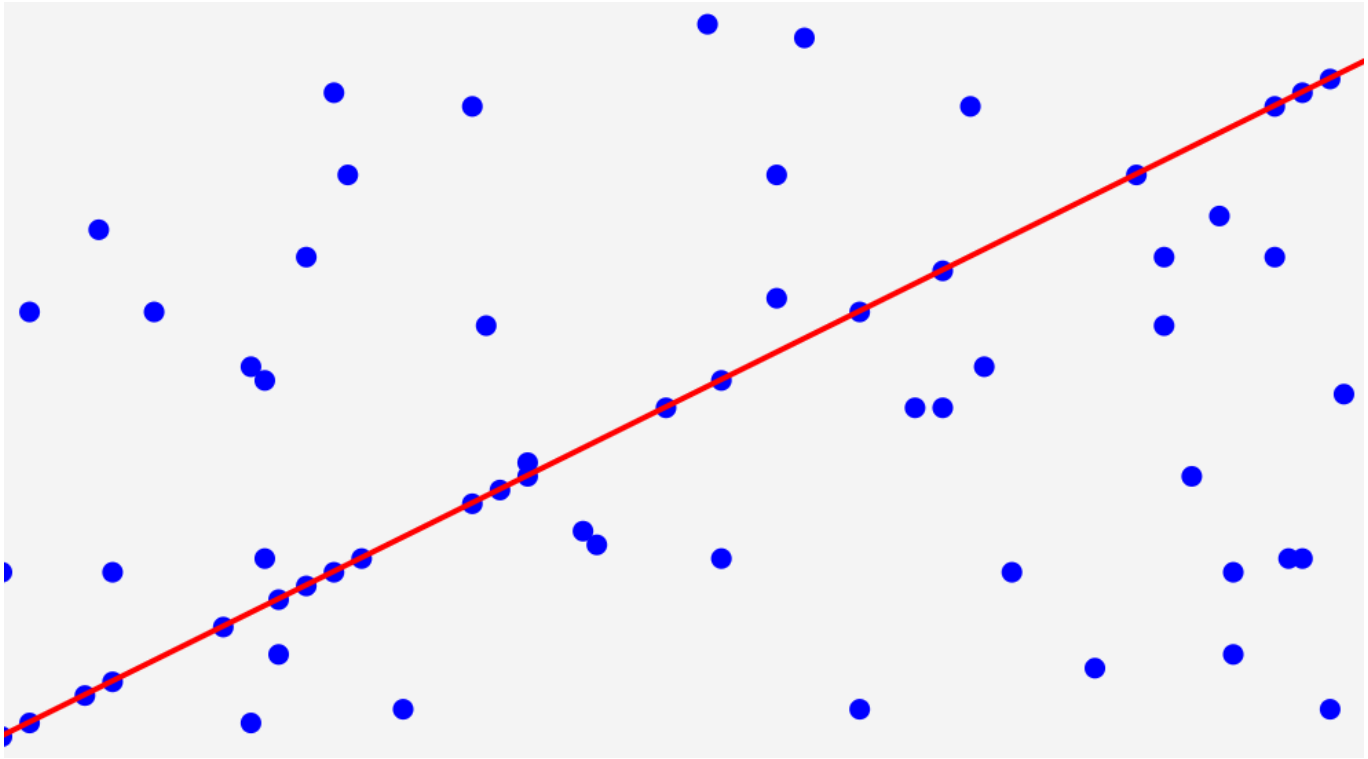
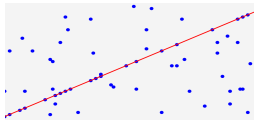


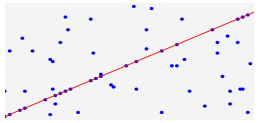


Vous avez dit « online » ?



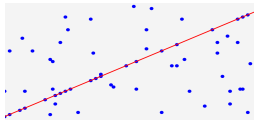
Force brute

- 2 points définissent une droite.
- Pour chaque paire de points (N^2), considérer tous les points (N) pour décider si P% sont sur la droite.
- Complexité: $O(N^3)$
- 1000 points → ~ 1 milliard d'opérations (~ 1 sec)



Force brute

```
// True ssi il y a une droite avec P% des points
boolean online() {
  for(int i = 0; i < N; ++i)
    for(int j = i + 1; j < N; ++j)
      if(containsPPP(new Line(points[i], points[j])))
        return true;
  return false;
}
```



Force brute

```
// La droite line contient-elle P% des points ?
boolean containsPPP(Line line) {
  final int NP = N * P;
  int cpt = 0;
  for(Point p : points)
    if(line.contains(p)) {// 3 points alignés. cf énoncé
      cpt += 100;
      if(cpt >= NP)
        return true;
    }
  return false;
}
```

Force brute

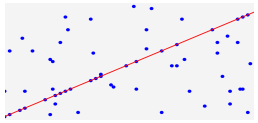
Complexité : $O(N^3)$

- 1.000 → 1,9 sec
- 10.000 → 1800 sec = 30 min
- 100.000 → 21 jours ?!
- 1.000.000 → 58 ans ???!!! (l'âge du professeur :)

Force brute (variante)

```
// Toutes les droites qui couvrent P% des points
Set<Line> setOfLinesFB() {
  final Set<Line> result = new HashSet<>();
  for(int i = 0; i < N; ++i)
    for(int j = i + 1; j < N; ++j) {
      Line line = new Line(points[i], points[j]);
      if(containsPPP(line))
        result.add(line);
    }
  return result;
}

boolean online() { return !setOfLinesFB().isEmpty(); }
```



Diviser pour Régner

- Ne pas résoudre directement pour les N points mais séparer le problème en deux.
- Résoudre récursivement les deux sous-problèmes et …
- … joindre les deux solutions partielles pour donner la solution globale

Diviser pour Régner

```
// Les droites qui couvrent P% des points de [d, f[
Set<Line> setOfLines(int d, int f) {
  if((f - d) <= 10)              // trop petit pour diviser
    return setOfLinesBF(d, f);// force brute
  else {
    final Set<Line> result = new HashSet<>();
    int m = (d + f) / 2;
    Set<Line> both = setOfLines(d, m);
    both.addAll(setOfLines(m, f));
    for(Line line : both)
      if(containsPPP(d, f, line))
        result.add(line);
    return result;
  }
}
```

```
T(N) =
  T(N/2) +
  T(N/2) +
  both.size() * N
```

Diviser pour Régner

$T(N) = 2 * T(N/2) + \text{both.size()} * N$

Si both.size() = N **Alors** $T(N) = 2 * T(N/2) + N^2$

Solution: $T(N) \in N^2 \times \log(N)$

Diviser pour Régner

$T(N) = 2 * T(N/2) + \text{both.size()} * N$

Mais $P \geq 20$!!

Il peut y avoir maximum 5 droites (différentes) avec 20% des points. Donc both.size() ≤ 10 !

$T(N) = 2 * T(N/2) + N$

Complexité : N x log(N)

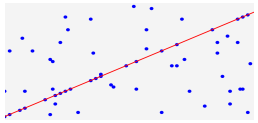
Diviser pour Régner

Force Brute:

- 1.000 → 1,9 sec
- 10.000 → 1800 sec = 30 min
- 100.000 → 21 jours ?!
- 1.000.000 → 58 ans ???!!!

Diviser pour régner:

- 10.000 → 0,04 sec
- 100.000 → 0,17 sec
- 1.000.000 → 2,3 sec

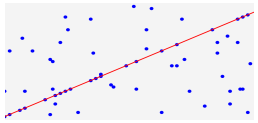


Vous êtes sûrs ?

ou plutôt, avez-vous vraiment besoin d'être sûrs ?

- Un américain moyen au cours de sa vie a une probabilité de 1E-4 d'être frappé par la foudre et 1E-5 d'en mourir!

- Accepteriez-vous que votre algorithme se trompe ? Avec quelle probabilité d'erreur ?



Algorithme probabiliste

S'il y a une droite avec P% des points,

un point choisi au hasard a P% de chance de s'y trouver !

Deux points choisis indépendamment ont P%^{2} de s'y trouver conjointement

```
// Prob[faux négatif] ≤ 1 - P%²
// P = 20 → 96%; P = 50 → 75%; P = 90 → 19%
boolean aligned_or_maybe_not() {
  int i1 = rand.nextInt(N); // Deux points au hasard
  int i2 = (i1 + 1 + rand.nextInt(N - 1)) % N; // diff.
  return containsPPP(new Line(points[i1], points[i2]));
}
```

Algorithme probabiliste

```
// Prob[faux négatif] ≤ (1 - P%²)^NB_ESSAIS
// P = 20 → 1E-9; P = 50 → 1E-63; P = 80 → 1E-222
boolean aligned_or_very_probably_not() {
  final int NB_ESSAIS = 500;      // 200000/P/P
  for(int k = 0; k < NB_ESSAIS; ++k)
    if(aligned_or_maybe_not())
      return true;
  return false;
}

boolean online() {
  return aligned_or_very_probably_not();
}
```

Algorithme probabiliste

Pour N = 10.000.000 l'algorithme probabiliste est de 4 à 400 fois plus rapide que l'algorithme diviser pour régner !