Recursion + Dynamic Programming

Graphs: Max Flow + min cut

Optimization → Linear Programing

Randomized Algorithms → Streaming/Filtering

Approximation Algorithms

Tower of Hanoi

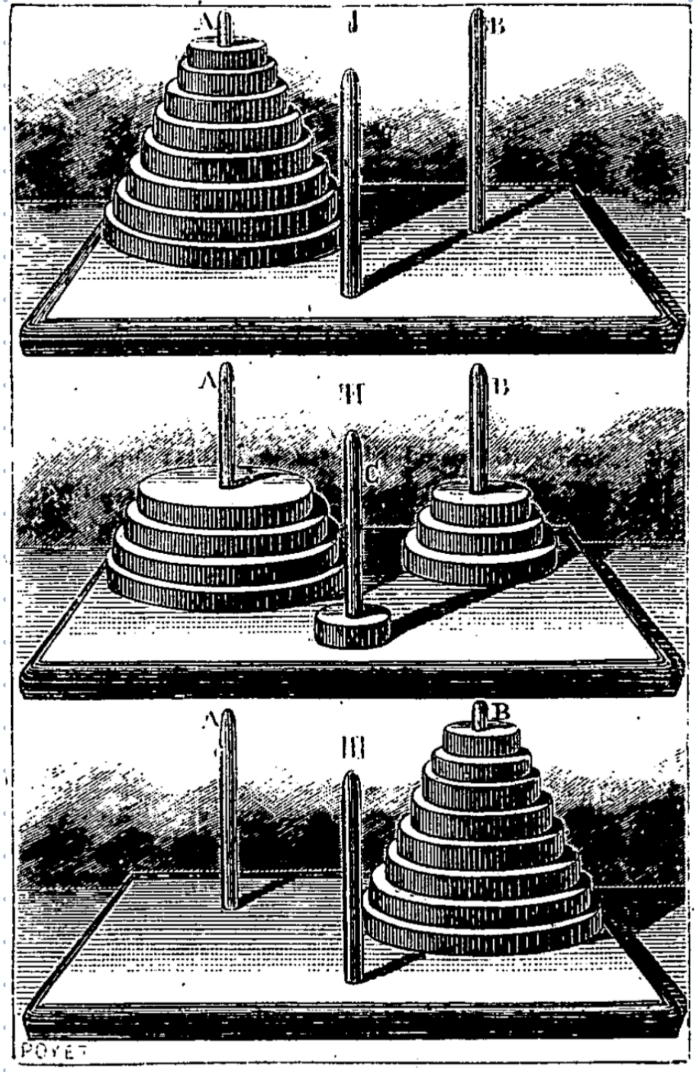
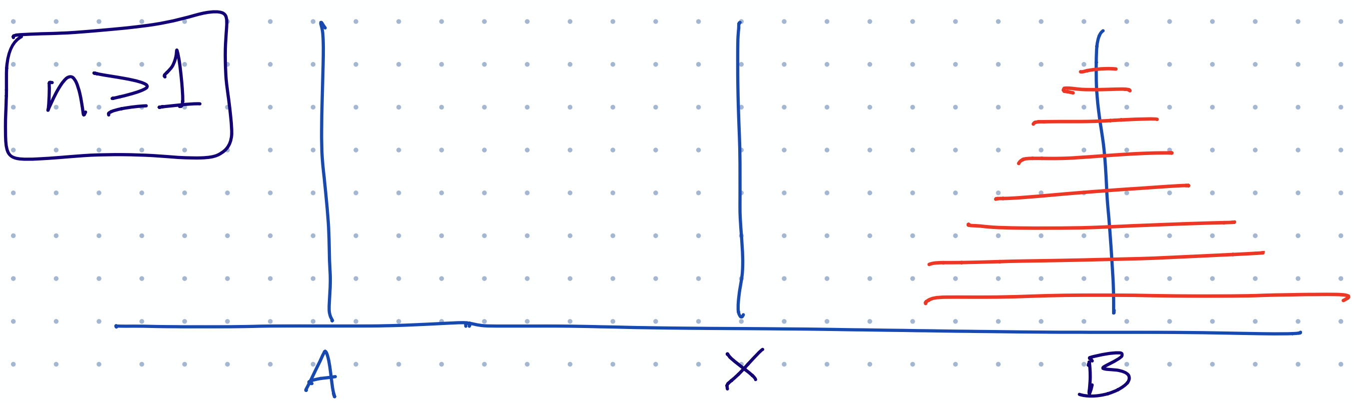
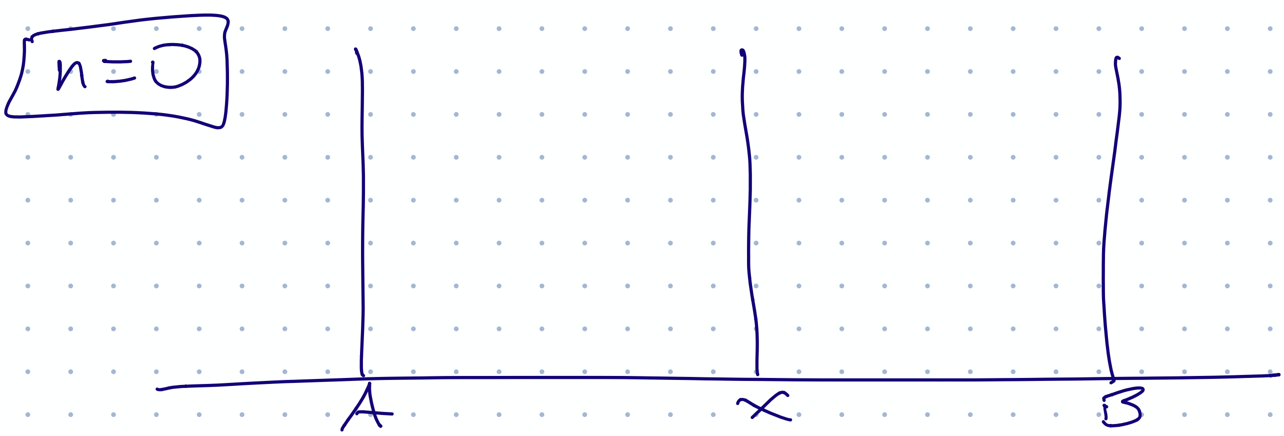


Fig. 18. — La tour d'Hanoï.

Le mandarin N. Claus (de Siam) nous raconte qu'il a vu, dans ses voyages pour la publication des écrits de l'illustre Fer-Fer-Tam-Tam, dans le grand temple de Bénarès, au-dessous du dôme qui marque le centre du monde, trois aiguilles de diamant, plantées dans une dalle d'airain, hautes d'une coudée et grosses comme le corps d'une abeille. Sur une de ces aiguilles Dieu enfila, au commencement des siècles, soixante-quatre disques d'or pur, le plus large reposant sur l'airain, et les autres, de plus en plus étroits, superposés jusqu'au sommet. C'est la tour sacrée de Brahma. Nuit et jour, les prêtres se succèdent sur les marches de l'autel, occupés à transporter la tour de la première aiguille de diamant sur la troisième, sans s'écarter des règles fixes que nous venons d'indiquer, et qui ont été imposées par Brahma. Quand tout sera fini, la tour et les brahmes tomberont, et ce sera la fin des mondes !

L'industrie étrangère s'est emparée depuis peu du jeu de notre ami et de sa légende ; mais nous pouvons affirmer que le tout a été imaginé, il y a quelque temps déjà, au nº 56 de la rue Monge, à Paris, dans la maison bâtie sur l'emplacement de celle où mourut Pascal, le 19 août 1662.





Moves n disks from src to dst using tmp as a placeholder as necessary.

```
HANOI(n, src, dst, tmp):
  if n > 0
    HANOI(n − 1, src, tmp, dst)   ⟨⟨Recurse!⟩⟩
    move disk n from src to dst
    HANOI(n − 1, tmp, dst, src)   ⟨⟨Recurse!⟩⟩
```

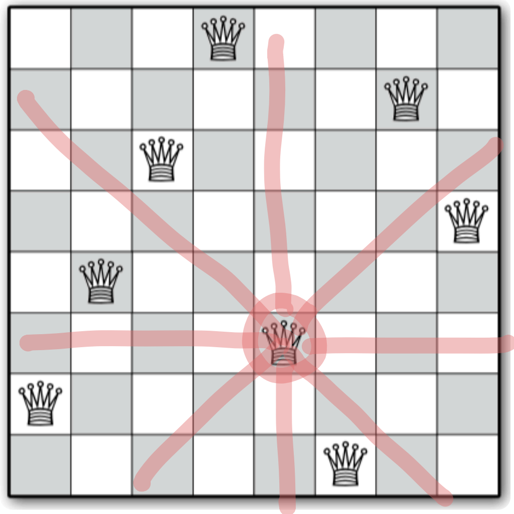
HANOI(n, A, B, X)

$2^n - 1$ moves

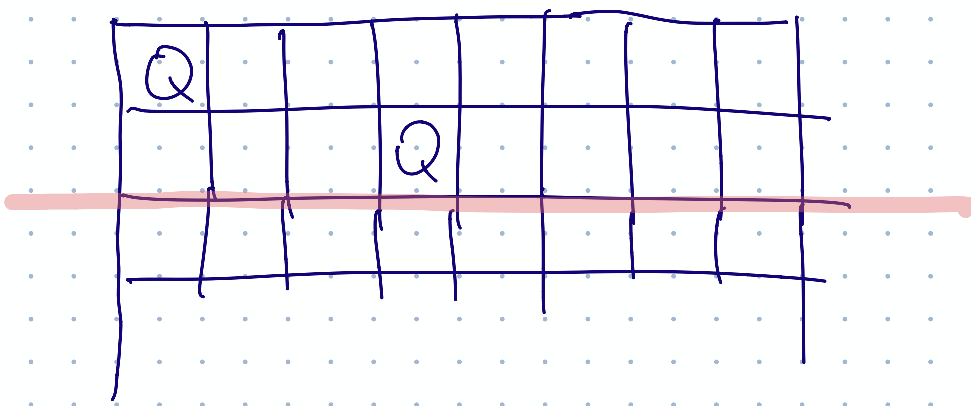
A un coup par seconde, il faut plus de quatre minutes pour déplacer la tour de huit étages. Pour exécuter le transport de la tour d'Hanoï à soixante-quatre étages, conformément aux règles du jeu, il faudrait faire un nombre de déplacements égal à

18 446 744 073 709 551 615;

ce qui exigerait plus de *cinq milliards de siècles !*



"methodisches Tattonieren"

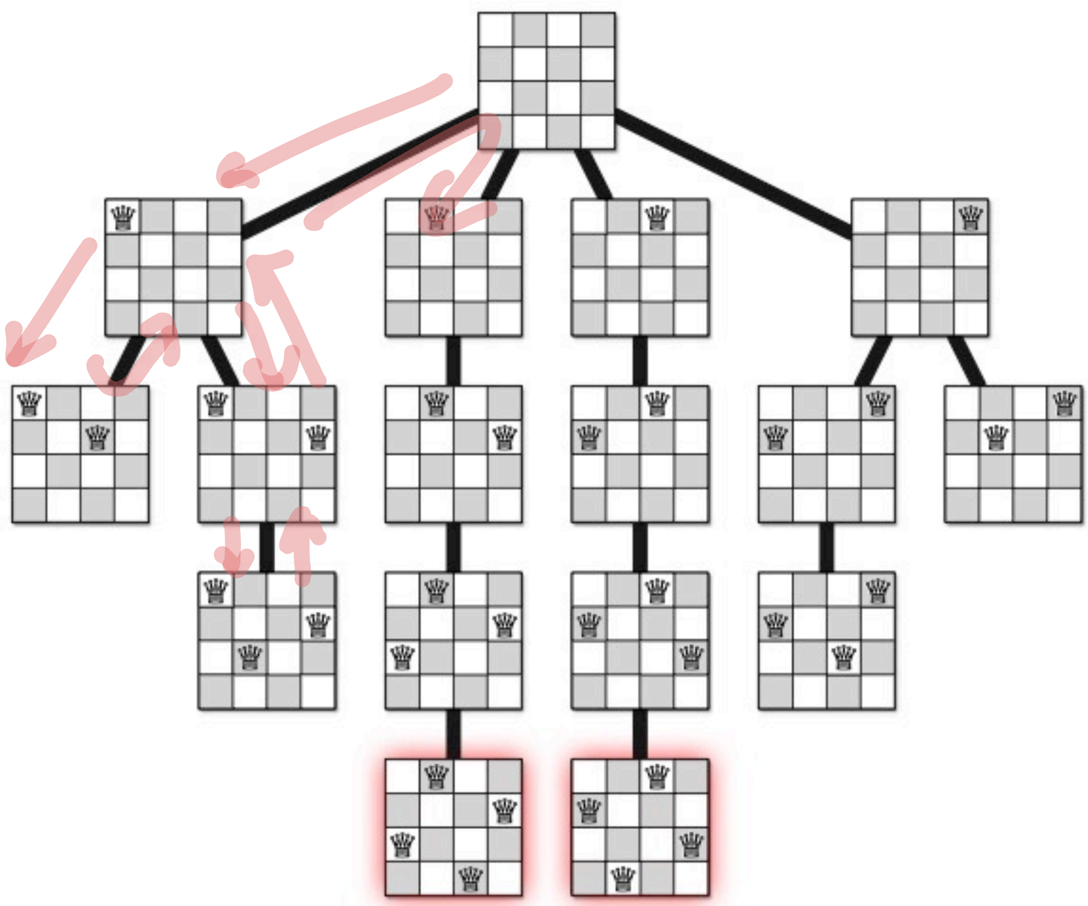


```
PlaceQueens(Q[1..n], r):
    if r = n + 1
        print Q[1..n]
    else
        for j ← 1 to n
            legal ← True
            for i ← 1 to r − 1
                if (Q[i] = j) or (Q[i] = j + r − i) or (Q[i] = j − r + i)
                    legal ← False
            if legal
                Q[r] ← j
                PlaceQueens(Q[1..n], r + 1)        ⟨⟨Recursion!⟩⟩
```

Figure 2.2. Gauss and Laquière's backtracking algorithm for the n queens problem.

Place queens in rows $r..n$

consistent with queens already in rows $1..r-1$



3	**1**	**4**	1	**5**	9	2	**6**	5	3	**5**?	8	9	7	9	3	2	3	8	4	6	2	6

Longest increasing subsequence of A[j..n]
where everything is bigger than A[i]

							i				j											n
3	**1**	**4**	1	**5**	9	2	**6**	5	3	5	**8**?	9	7	9	3	2	3	8	4	6	2	6

$$
LISbigger(i,j) = \begin{cases} 0 & \text{if } j > n \\ LISbigger(i, j+1) & \text{if } A[i] \geq A[j] \\ \max \left\{ \begin{array}{c} LISbigger(i, j+1) \\ 1 + LISbigger(j, j+1) \end{array} \right\} & \text{otherwise} \end{cases}
$$

```
LISBIGGER(i, j):
  if j > n
      return 0
  else if A[i] ≥ A[j]
      return LISBIGGER(i, j + 1)
  else
      skip ← LISBIGGER(i, j + 1)
      take ← LISBIGGER(j, j + 1) + 1
      return max{skip, take}
```

```
LIS(A[1..n]):
  A[0] ← −∞
  return LISBIGGER(0, 1)
```