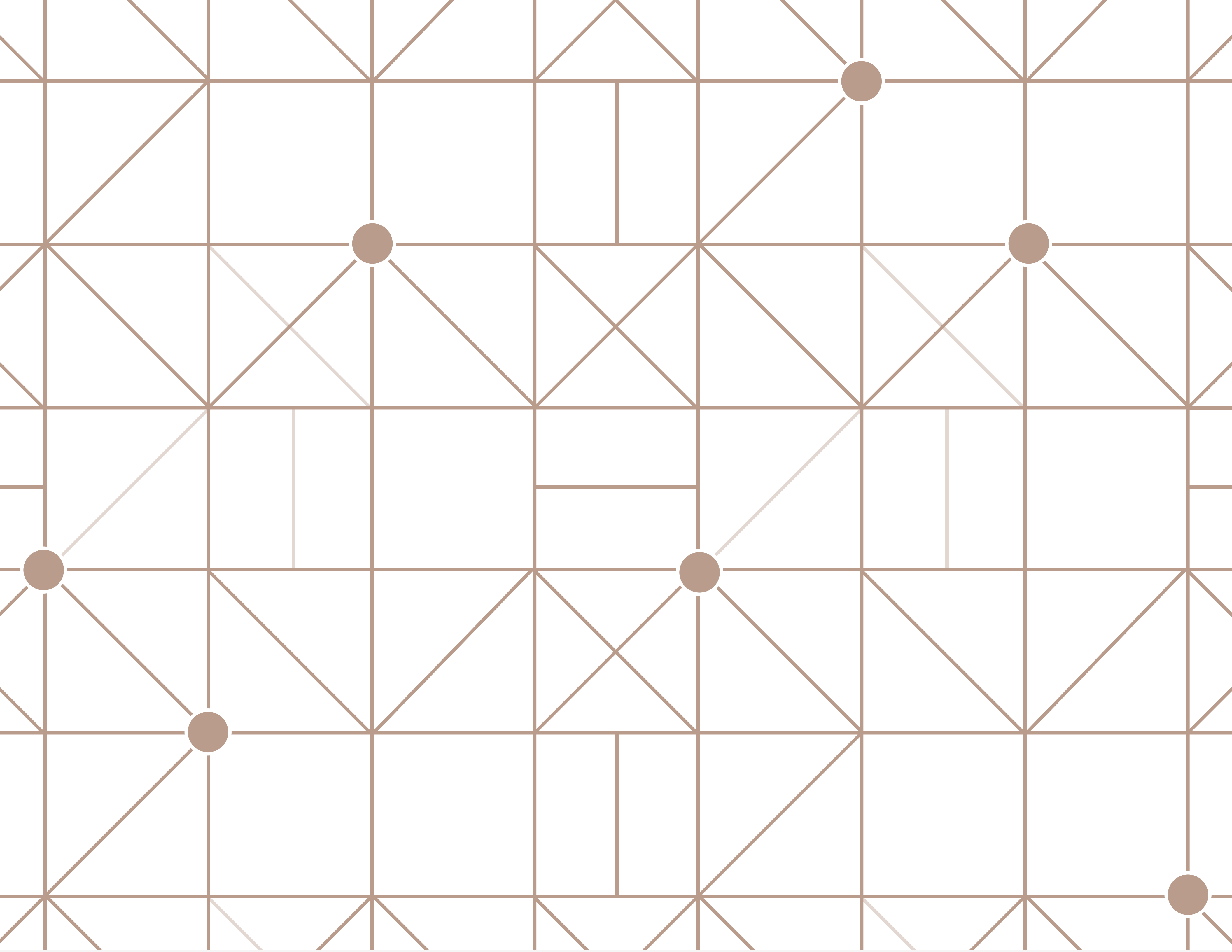
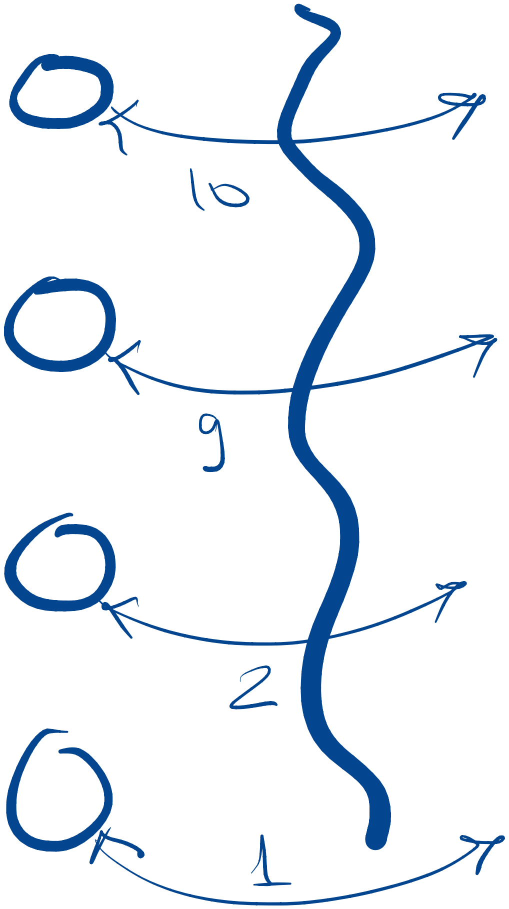




crossing time

23
25
17

2 can cross at a time → time is for slowest only with "flashlight"

Fibonacci Number $F_0 = 0$ $F_n = F_{n-1} + F_{n-2}$

$F_1 = 1$ $\forall n \geq 2$

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}
0	1	1	2	3	5	8	13	21	34	55

Task: Compute F_n = function of n = input

(1) Math following recursive

```
Fib(n)
    if n=0 return 0
           (output)
    if n=1 return 1
    (else) return Fib(n-1) + Fib(n-2)
```

(A) correct? yes

(B) how fast? exponential runtime $\boxed{\Theta(2^n)}$??

tight asymptote

lower bound

upper bound

Ω Θ O

asymp exact asymp

asymptote

ex $\Theta(n^2) = f(n)$

$c_2 \cdot n^2 \leq f(n) \leq c_1 \cdot n^2$

low bound upper bound

② Array computation Fib(n)

F = array F[0]=0 , F[1]=1

for i=2 : n

  F[i] = F[i-1] + F[i-2]

return F[n]

$\boxed{\text{C. storage} \quad \text{A correct} \quad \text{B how fast}}$

LINEAR

$\Theta(n)$

Q: can we store only last 3 values?

③ Matrix-Multiplication-based Fib(n)

$$M = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$ 2×2

claim $$M^n = \begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix}$$

induction proof base case $$M^1 = M = \begin{bmatrix} F_2 & F_1 \\ F_1 & F_0 \end{bmatrix} ?$$

inductive step $n \to n+1$

$$\boxed{M^n = \begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix}} \Longrightarrow M^{n+1} = \begin{bmatrix} F_{n+2} & F_{n+1} \\ F_{n+1} & F_n \end{bmatrix}$$

ind hyp ind conclusion

proof: $$M^{n+1}_1 = M^n \times M_2 \overset{\text{hyp}}{=} \begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

$$= \begin{array}{c} 1 \\ 2 \end{array} \begin{bmatrix} F_{n+1} + F_n & F_{n+1} \\ F_n + F_{n-1} & F_n \end{bmatrix} = \begin{bmatrix} F_{n+2} & F_{n+1} \\ F_{n+1} & F_n \end{bmatrix} \checkmark$$

(2) Matrix Fib (n)   Run Time?

$M = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$ ✓

$\boxed{\text{compute } M^n = A}$ exponential of a matrix

naive $M^n = \underbrace{M \cdot M \cdot M \cdots M}_{n \text{ times}}$

$\Theta(n)$ time

return $M[1,2]$ ✓

$\Theta(\log n)$

Fast exponentiation (example)

M

$M^2 = M \cdot M$

$M^4 = M^2 \cdot M^2$

$M^8 = M^4 \cdot M^4$

$M^{16} = M^8 \cdot M^8$

$M^{32} = M^{16} \cdot M^{16}$

$M^{64} = M^{32} \cdot M^{32}$

Want $M^{100} = M^{64} \cdot M^{32} \cdot M^4$

$M^{171} = M^{128} \cdot M^{32} \cdot M^8 \cdot M^2 \cdot M^1$

④ Generating Function

$$F(n) = \frac{\phi^n - \varphi^n}{\sqrt{5}}$$

$$\phi = \frac{1+\sqrt{5}}{2}$$

$$\varphi = \frac{1-\sqrt{5}}{2}$$

ϕ real number $\in \mathbb{R}$

math-coprocessor estimate

$\simeq$ constant time.

$\simeq \Theta(1)$

idea for ϕ, φ

guess?

$F(n) \simeq a^n$? true

rec. Fib:

$$F_{n+1} = F_n + F_{n-1}$$

$$a^{n+1} = a^n + a^{n-1} \quad \Big| \div a^{n-1}$$

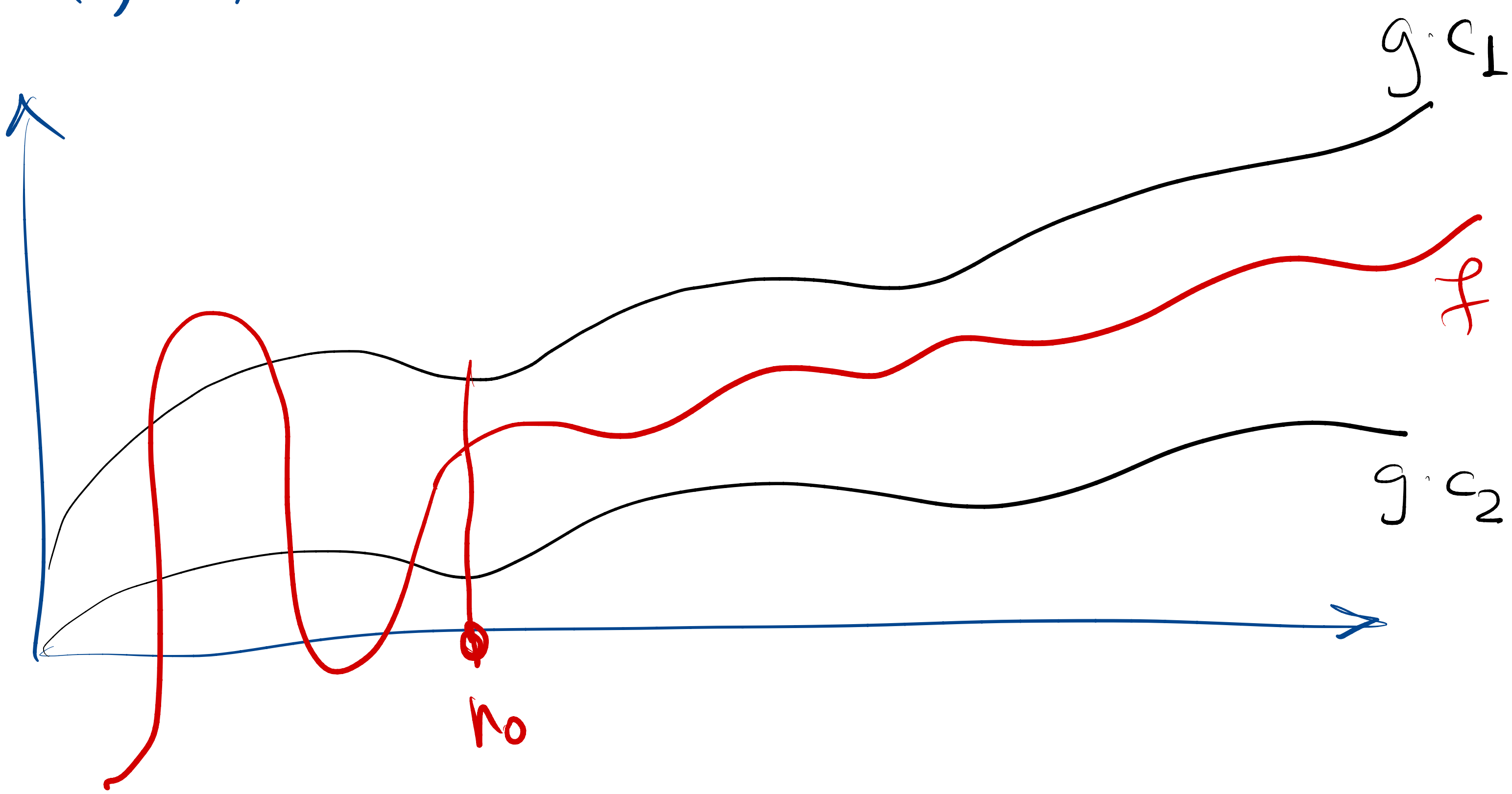
$$a^2 = a + 1$$

quad eq $\Rightarrow \phi, \varphi$.

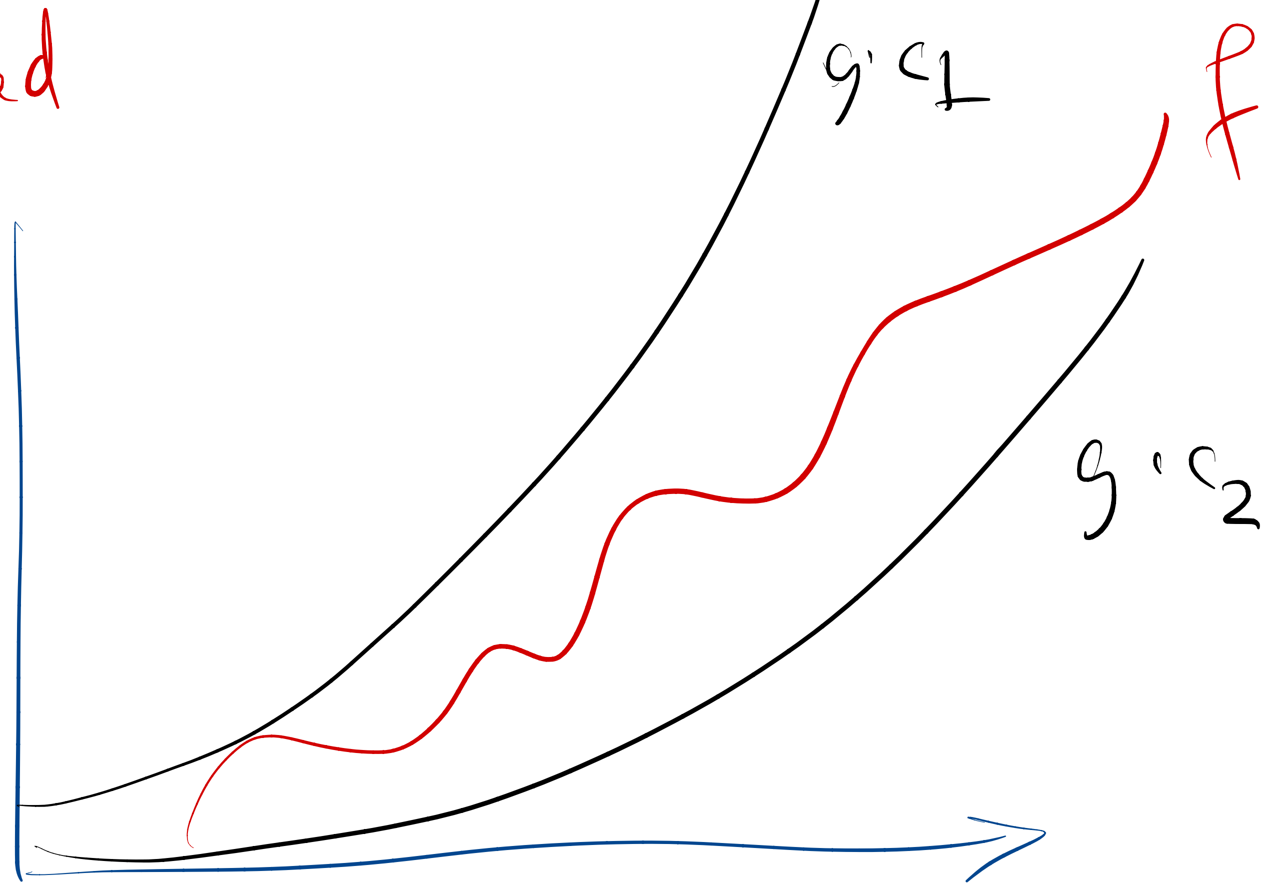
big O notation

$f = \Theta(g)$ $c_1, c_2 > 0$ constants $c_1 > c_2$

$f(n) = \Theta(g(n))$: $g \cdot c_2 \le f \le g \cdot c_1$

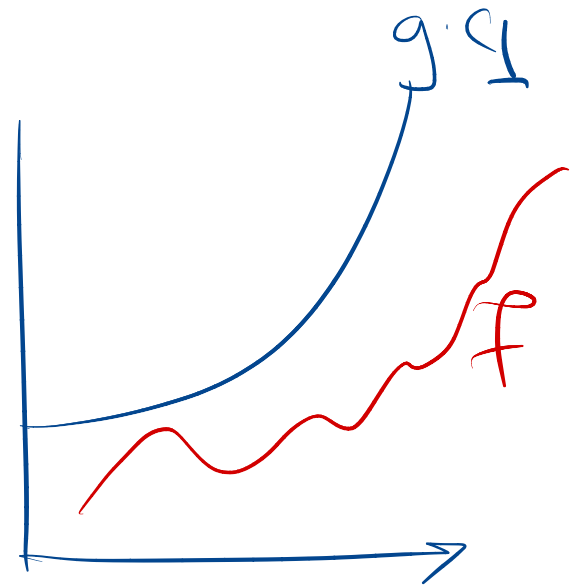


g = quad



Big O = upper bound $f = O(g)$

$\exists c_1 > 0 \quad f \leq g \cdot c_1$



$f(n)$

$2n^2+3n+2 \qquad \Theta(n^2)$

$n^2\log n + n^3 \qquad \Theta(n^3)$

$2^n + 5n^2 - 3 \qquad \Theta(2^n)$

$3^n + 2^n \qquad \Theta(3^n)$

$3^n = \Theta(2^n)$?

$c_2 \cdot 2^n \le 3^n \le \cancel{c_1 \cdot 2^n}$

true

$\frac{3^n}{2^n} \le c_1$

$\left(\frac{3}{2}\right)^n \le c_1$

$\lim (1.5)^n = +\infty$

NO

$\log_a x = b \iff a^b = x$

$f = \log_2 x$

$g = \log_3 x$

$f = \Theta(g)$

$\boxed{c_2 \cdot \log_3(n) \le \log_2(n)} \le c_1 \cdot \log_3(n)$

$\log_3(x) \stackrel{?}{=} \log_2(x) \cdot \log_3 2$?

$3^{\log_3 x} = 3^{\log_2 x \cdot \log_3 2}$

$x = \left(3^{\log_3 2}\right)^{\log_2 x}$

$c_2 = 1$

$$\log_3(n) \leq \boxed{\log_2(n) \leq c_1 \cdot \log_3(n)} \; ??$$

$2^{\log_2 x}$ x

$$\log_2(n) \leq c_1 \cdot \log_2(n) \cdot \log_3 2$$

$$1 \leq c_1 \cdot \log_3 2$$

constant

Binary Search

$A = $

-3	-1	0	5	12	17	21	75	92	103
1	2	3	4	5	6	7	8	9	10

$BS(v, A, begin, end)$

Find value v = given in sorted $A[begin : end]$ (indices)

initially begin = 1, end = 10

- $m = \frac{begin + end}{2}$
- $\boxed{\text{if } v = A[m]}$ ⟹ done, return m, $A[m]$

  if $v < A[m]$ // search $A[begin : m]$
  $BS(v, A, begin, m)$

  //else

  if $v > A[m]$ // search $A[m : end]$
  $BS[v, A, m, end]$.

Fix termination

R.T. $T(n)$ = time to search $|A| = n$ size

$$T(n) = \Theta(1) + T(n/2)$$

recurrence due to recursive calls

$T(n)$: A size; $T(n/2)$: $|A|/2$ size

$k=1 \quad T(n) = 1 + T(n/2) \quad \forall n$

$k=2 \quad = 1 + [1 + T(n/4)] = 2 + T(n/4)$

$k=3 \quad = 2 + [1 + T(n/8)] = 3 + T(n/8)$

$k=4 \quad = 3 + [1 + T(n/16)] = \boxed{4} + T(n/16)$

pattern

$k: \quad k + T(n/2^k)$

optional (if messy) induction proof

step k $\qquad$ step $k+1$

$k + T(n/2^k) \Longrightarrow k+1 + T(n/2^{k+1})$

$$T(n) = K + T\left(\frac{n}{2^k}\right)$$

last K : $\frac{n}{2^k} \simeq 1 \Rightarrow T(n/2^k) = T(1) = \text{base}$

$$n \simeq 2^k$$

$$k \simeq \log_2(n)$$

$$T(n) = \log_2(n) + \underbrace{T(1)}_{\text{const}} = \Theta(\log n)$$

Merge Sort $A(b:e)$: Sort Array A

L	R
sorted	sorted

$m = \frac{b+e}{2}$

Split in 2 $A[b:m)$

$A[m+1:e]$

$\text{MergeSort}(A[b:m])$ $T(n/2)$

$\text{MergeSort}(A[m+1:e])$ $T(n/2)$

Merge/Combine sorted $A[b:m)$ (L) sorted $A[m+1:e]$ (R)

$i=1$ (L) $j=1$ (R) $t=1$

if $L[i] < R[j]$

$C[t] = L[i]$,

$i = i+1$

$\Theta(n)$

else

$C[t] = R[j]$

L $\quad j = j+1$

$t = t+1$

until both arrays finished

R.T $\quad T(n) = \underbrace{T(n/2)}_{\text{L-sort}} + \underbrace{T(n/2)}_{\text{R-sort}} + \underbrace{n}_{\text{Merging}}$

$k=1 \quad 2T(n/2) + n$

$k=2 \quad = 2\left[2T(n/4) + n/2\right] + n = 4T(n/4) + 2n$

$k=3 \quad = 4\left[2T(n/8) + n/4\right] + 2n = 8T(n/8) + 3n$

$k=4 \quad = 8\left[2T(n/16) + n/8\right] + 3n = 16T(n/16) + 4n$

k pattern

$$\boxed{= 2^k T(n/2^k) + kn}$$

last k: $\frac{n}{2^k} \simeq 1$ case $\Longleftrightarrow$ $k \simeq \log_2 n$

$$T(n) = 2^{\log_2 n} T(1) + \log_2(n) \cdot n$$

$$= n \cdot \text{const} + n \log_2 n$$

$$\Theta(n \log_3 n)$$

HW1 3.1-1 $\max(f,g) = \Theta(f+g)$

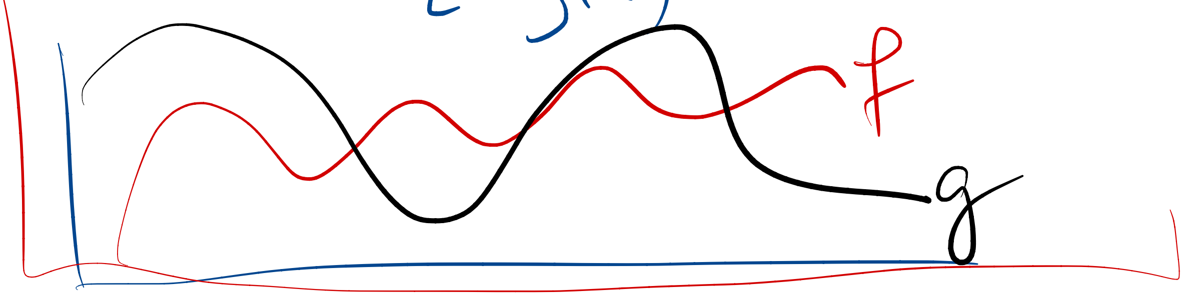
Def Θ $\quad c_2(f+g) \le \max(f,g) \le c_1(f+g)$

$c_1, c_2 > 0$

guess $\left.\begin{array}{l} c_1 = 2 \\ c_2 = 1/3 \end{array}\right\} \Leftrightarrow ?!\ \frac{1}{3}(f+g) \le \max(f,g) \le 2(f+g)$

$c_2 < c_1$

$$\max(f(n), g(n)) = \begin{cases} f(n) & \text{if } f(n) > g(n) \\ g(n) & \text{otherwise} \end{cases}$$



$\boxed{\text{any function}}$ $\min(f,g) \le f \le \max(f,g)$

$\ln \ln (n^2)$ $\qquad$ $2^{\ln^2(n)}$

$2^{\ln(n) \cdot \ln(n)}$

$\ln(\ln(n^2))$

asymptote

$\ln(\ln(n^2)) \ll \boxed{n^{\ln(n)}}$ bigger

explanation

$\ln(\ln(n^2)) \ll$ $\log(n)$ not that ... n^2 easy $\le n^{\ln(n)}$

K-nearest neighbours

Javier Béjar

LSI - FIB

Term 2012/2013

classifier

supervised
(not this class)

f(datapoint_representation) $\simeq$ label

x_i y_i

Parametric vs Non parametric Models

- In the models that we have seen, we select a hypothesis space and adjust a fixed set of parameters with the training data ($h_\alpha(x)$)
- We assume that the parameters α summarize the training and we can forget about it
- This methods are called **parametric** models
- When we have a small amount of data it makes sense to have a small set of parameters and to constraint the complexity of the model (avoiding overfitting)

Parametric vs Non parametric Models

- When we have a large quantity of data, overfitting is less an issue
- If data shows that the hipothesis has to be complex, we can try to adjust to that complexity
- A **non parametric** model is one that can not be characterized by a fixed set of parameters
- A family of non parametric models is **Instance Based Learning**

Instance Based Learning

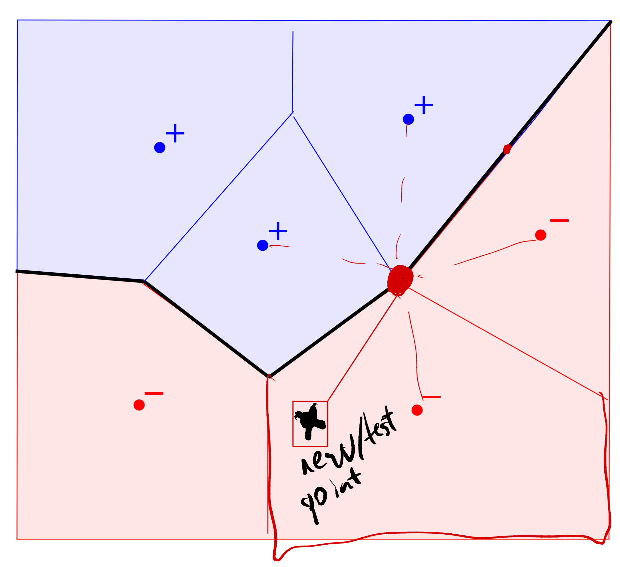
- Instance based learning is based on the memorization of the dataset
- The number of parameters is unbounded and grows with the size of the data
- There is not a model associated to the learned concepts
- The classification is obtained by looking into the memorized examples
- The cost of the learning process is 0, all the cost is in the computation of the prediction
- This kind learning is also known as **lazy learning**

K-nearest neighbours

- **K-nearest neighbours** uses the local neighborhood to obtain a prediction
- The K memorized examples more similar to the one that is being classified are retrieved
- A distance function is needed to compare the examples similarity
  - Euclidean distance ($d(x_j, x_k) = \sqrt{\sum_i (x_{j,i} - x_{k,i})^2}$)
  - Mahnattan distance ($d(x_j, x_k) = \sum_i |x_{j,i} - x_{k,i}|$)
- This means that if we change the distance function, we change how examples are classified

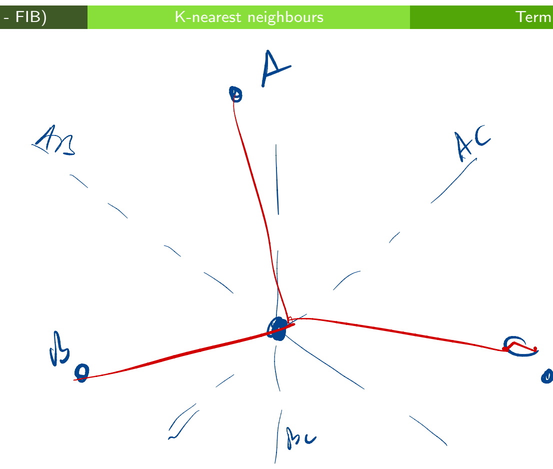
K-nearest neighbours - hypothesis space (1 neighbour)

binary labels = {+,-}



{+ −}
training set (I know labels)

unique partition of plane (the space) s.t. $\forall x$ is in the partition with closest centroid.



K-nearest neighbours - Algorithm

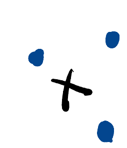
Neighbourhood-k
$N(x_{new}) = \{$ the closest k points in training $\}$

- Training: Store all the examples
- Prediction: $h(x_{new})$
  - $\boxed{\text{Let be } x_1, \ldots, x_k \text{ the } k \text{ more similar examples to } x_{new}}$
  - $h(x_{new})$= combine_predictions($x_1, \ldots, x_k$)
- The parameters of the algorithm are the number k of neighbours and the procedure for combining the predictions of the k examples
- The value of k has to be adjusted (crossvalidation)
  - We can overfit (k too low)
  - We can underfit (k too high)

k=3

Good - close

Bad - far away



- data density (not uniform)
  - easy to find close neighbors (dense / typical)
  - hard to find close neighbors (sparse / anomaly)

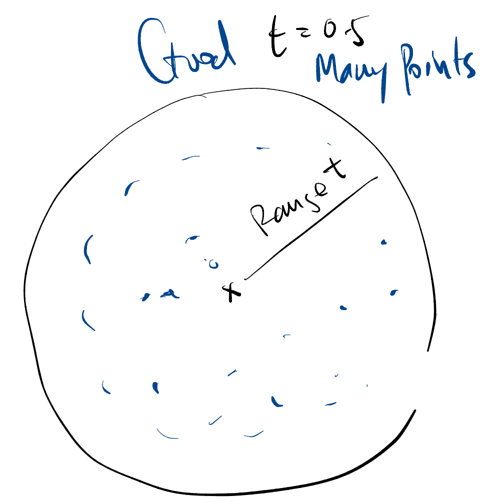
K NN
top/closest K

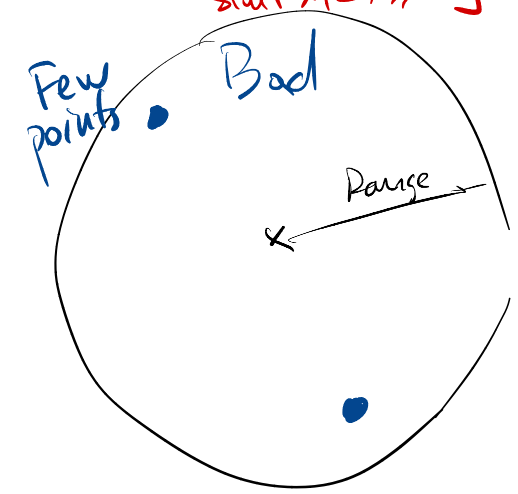
$N_x = \{$ closest k points to x $\}$

Range-NN
"t"

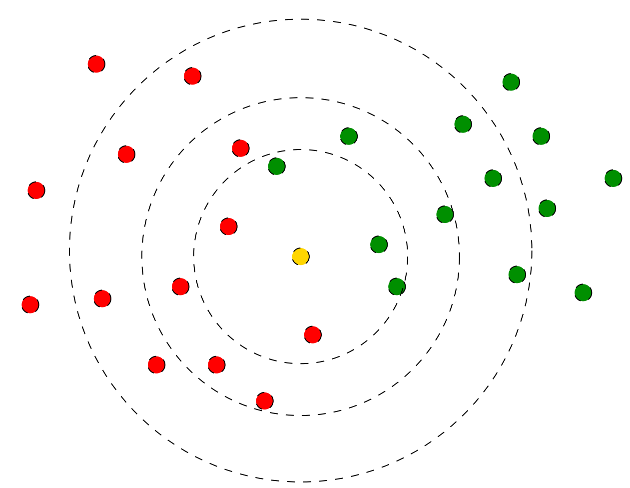
$N_x = \{$ all points within t similarity $\}$

$= \{ z \in \text{training} \mid sim(x,z) \geq t \}$





K-nearest neighbours - Prediction



Weighted version

- $\gamma(x,z) = \text{similarity}(x,z)$
- use all points

$$\text{predict}(\underbrace{x}_{\text{test}}, \underbrace{L}_{\text{label}}) = \text{avg} \underbrace{\sum_{i=1}^{N}}_{\text{over train set}} \overbrace{\underbrace{L(z_i)}_{\text{label } L \text{ for point } z_i} \cdot \gamma(x, z_i)}^{\text{weighted avg}}$$

Looking for neighbours

- Looking for the K-nearest examples for a new example can be expensive
- The straightforward algorithm has a cost $O(n \log(k))$, not good if the dataset is large
- We can use indexing with *k-d trees* (multidimensional binary search trees)
  - They are good only if we have around 2^{dim} examples, so not good for high dimensionality
- We can use *locality sensitive hashing* (approximate k-nn)
  - Examples are inserted in multiple hash tables that use hash functions that with high probability put together examples that are close
  - We retrieve from all the hash tables the examples that are in the bin of the query example
  - We compute the k-nn only with these examples

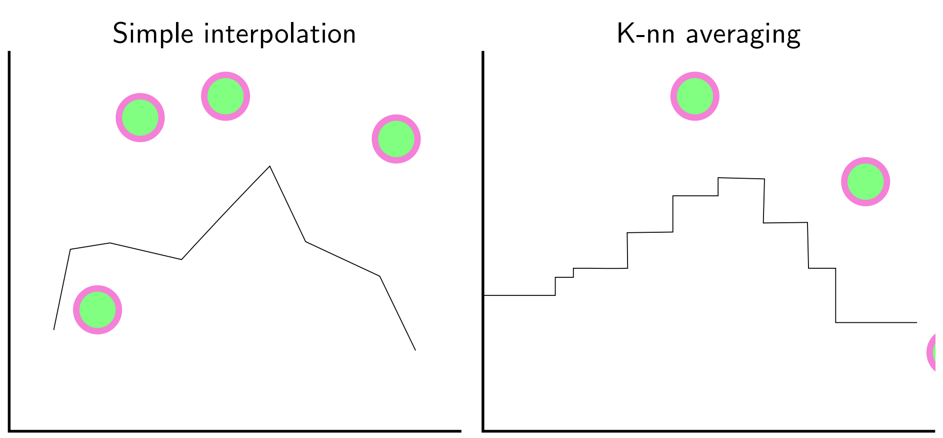
K-nearest neighbours - Variants

- There are different possibilities for computing the class from the k nearest neighbours
  - Majority vote
  - Distance weighted vote
    - Inverse of the distance
    - Inverse of the square of the distance
    - Kernel functions (gaussian kernel, tricube kernel, ...)
- Once we use weights for the prediction we can relax the constraint of using only k neighbours
  1. We can use k examples (local model)
  2. We can use all examples (global model)

K-nearest neighbours - Regression

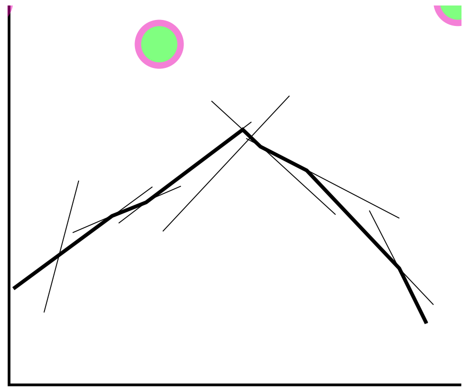
- We can extend this method from classification to regression
- Instead of combining the discrete predictions of k-neighbours we have to combine continuous predictions
- This predictions can be obtained in different ways:
  - Simple interpolation
  - Averaging
  - Local linear regression
  - Local weighted regression
- The time complexity of the prediction will depend on the method

K-nearest neighbours - Regression



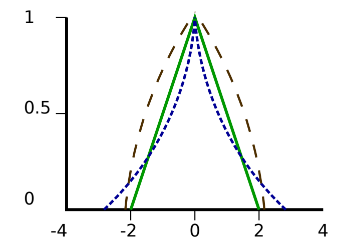
K-nearest neighbours - Regression (linear)

- K-nn linear regression fits the best line between the neighbors
- A linear regression problem has to be solved for each query (least squares regression)



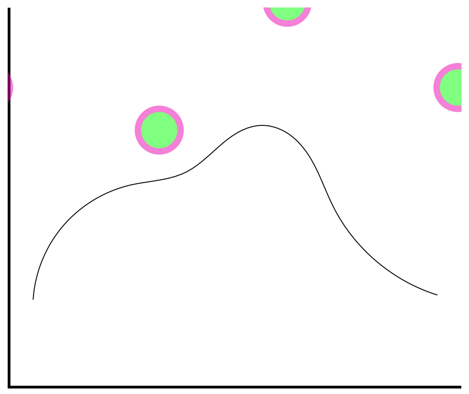
K-nearest neighbours - Regression (LWR)

- Local weighted regression uses a function to weight the contribution of the neighbours depending on the distance, this is done using a **kernel function**



- Kernel functions have a width parameter that determines the decay of the weight (it has to be adjusted)
  - Too narrow $\Longrightarrow$ overfitting
  - Too wide $\Longrightarrow$ underfitting
- A weighted linear regression problem has to be solved for each query (gradient descent search)

K-nearest neighbours - Regression (LWR)



K-nearest neighbours - Advantages

- The cost of the learning process is zero
- No assumptions about the characteristics of the concepts to learn have to be done
- Complex concepts can be learned by local approximation using simple procedures

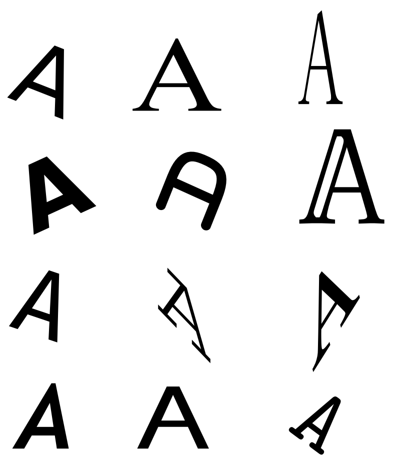
K-nearest neighbours - Drawbacks

- The model can not be interpreted (there is no description of the learned concepts)
- It is computationally expensive to find the k nearest neighbours when the dataset is very large
- Performance depends on the number of dimensions that we have (*curse of dimensionality*) $\Longrightarrow$ *Attribute Selection*

The Curse of dimensionality

- The more dimensions we have, the more examples we need to approximate a hypothesis
- The number of examples that we have in a volume of space decreases exponentially with the number of dimensions
- This is specially bad for k-nearest neighbors
  - If the number of dimensions is very high the nearest neighbours can be very far away

Optical Character Recognition



- OCR capital letters
- 14 Attributes (All continuous)
- Attributes: horizontal position of box, vertical position of box, width of box, height of box, total num on pixels, mean x of on pixels in box, ...
- 20000 instances
- 26 classes (A-Z)
- Validation: 10 fold cross validation

Optical Character Recognition: Models

- K-nn 1 (Euclidean distance, weighted): accuracy 96.0%
- K-nn 5 (Manhattan distance, weighted): accuracy 95.9%
- K-nn 1 (Correlation distance, weighted): accuracy 95.1%

idea: change of variable

$$2^{\ln^2 n}$$

$$(\ln(n))!$$

$$m = \ln(n)$$

$$2^{m \cdot m}$$

$$m!$$