cse 521: design and analysis of algorithms

Time & place

T, Th 1200-120 pm in **CSE 203**

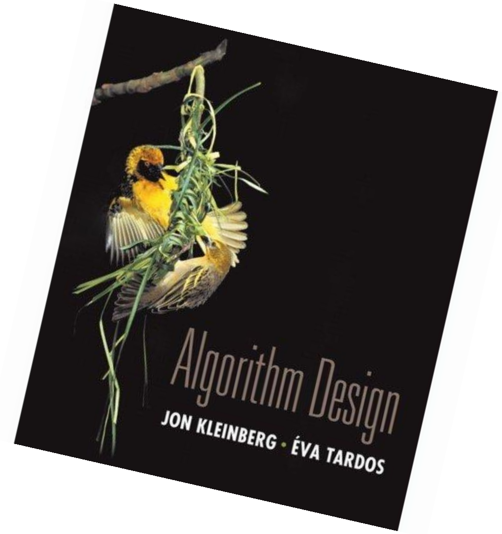
People

Prof: James Lee `(jrl@cs)`

TA: Thach Nguyen `(ncthach@cs)`

Book

Algorithm Design by Kleinberg and Tardos



Grading

50% homework (approx. bi-weekly problem sets)

20% take-home midterm

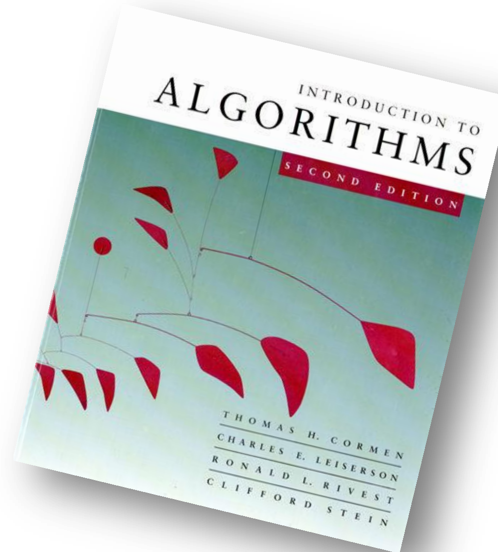
30% in-class final exam

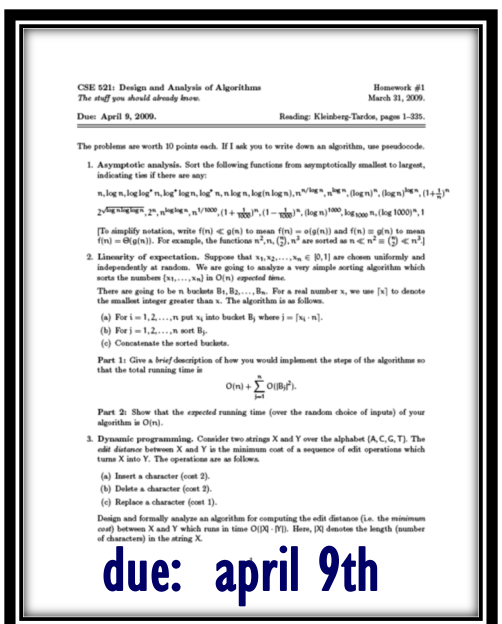
Website: **http://www.cs.washington.edu/521/**

something a little bit different

assume you know:

- asymptotic analysis
- basic probability
- basic linear algebra
- dynamic programming
- recursion / divide-and-conquer
- graph traversal (BFS, DFS, shortest paths)





CSE 521: Design and Analysis of Algorithms — Homework #1

The stuff you should already know. — March 31, 2009.

Due: April 9, 2009. — Reading: Kleinberg-Tardos, pages 1–335.

The problems are worth 10 points each. If I ask you to write down an algorithm, use pseudocode.

1. **Asymptotic analysis.** Sort the following functions from asymptotically smallest to largest, indicating ties if there are any:

   $n, \log n, \log\log^* n, \log^* \log n, \log^* n, n\log n, \log(n\log n), n^{n/\log n}, n^{\log n}, (\log n)^n, (\log n)^{\log n}, (1+\frac{1}{n})^n$

   $2^{\sqrt{\log n \log\log n}}, 2^n, n^{\log\log n}, n^{1/1000}, (1+\frac{1}{1000})^n, (1-\frac{1}{1000})^n, (\log n)^{1000}, \log_{1000} n, (\log 1000)^n, 1$

   [To simplify notation, write $f(n) \ll g(n)$ to mean $f(n) = o(g(n))$ and $f(n) \equiv g(n)$ to mean $f(n) = \Theta(g(n))$. For example, the functions $n^2, n, \binom{n}{2}, n^3$ are sorted as $n \ll n^2 \equiv \binom{n}{2} \ll n^3$.]

2. **Linearity of expectation.** Suppose that $x_1, x_2, \ldots, x_n \in [0,1]$ are chosen uniformly and independently at random. We are going to analyze a very simple sorting algorithm which sorts the numbers $\{x_1, \ldots, x_n\}$ in $O(n)$ *expected time*.

   There are going to be n buckets $B_1, B_2, \ldots, B_n$. For a real number x, we use $\lceil x \rceil$ to denote the smallest integer greater than x. The algorithm is as follows.

   (a) For $i = 1, 2, \ldots, n$ put x_i into bucket B_j where $j = \lceil x_i \cdot n \rceil$.
   (b) For $j = 1, 2, \ldots, n$ sort B_j.
   (c) Concatenate the sorted buckets.

   Part 1: Give a *brief* description of how you would implement the steps of the algorithms so that the total running time is

   $$O(n) + \sum_{j=1}^{n} O(|B_j|^2).$$

   Part 2: Show that the *expected* running time (over the random choice of inputs) of your algorithm is $O(n)$.

3. **Dynamic programming.** Consider two strings X and Y over the alphabet $\{A, C, G, T\}$. The *edit distance* between X and Y is the minimum cost of a sequence of edit operations which turns X into Y. The operations are as follows.

   (a) Insert a character (cost 2).
   (b) Delete a character (cost 2).
   (c) Replace a character (cost 1).

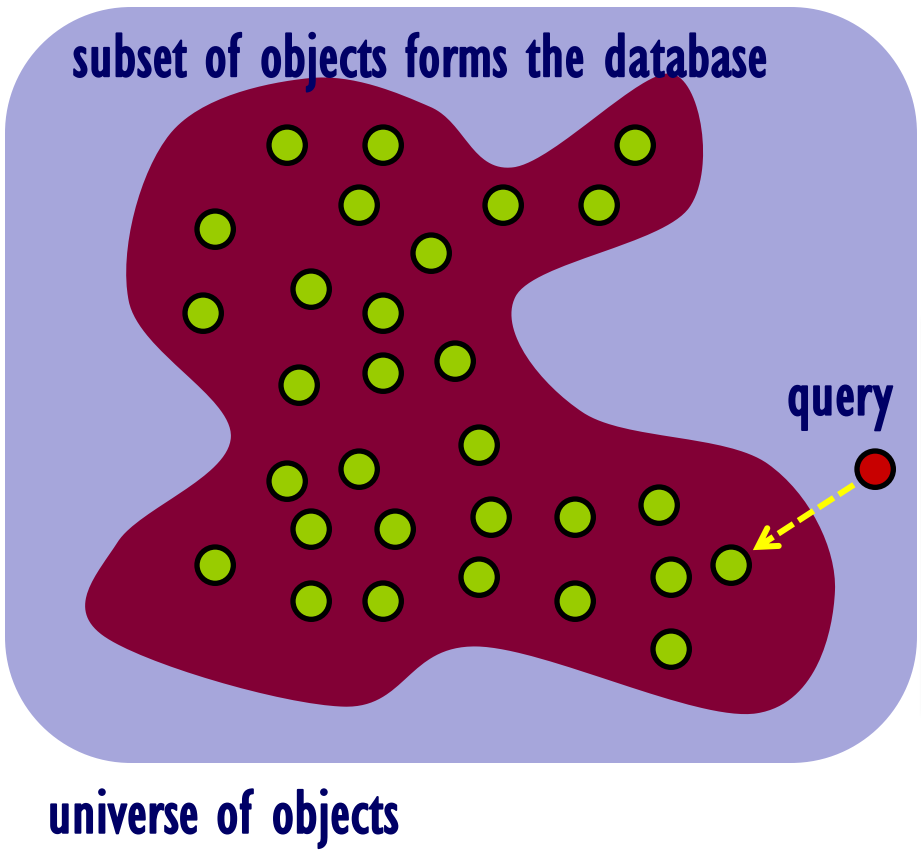
   Design and formally analyze an algorithm for computing the edit distance (i.e. the *minimum cost*) between X and Y which runs in time $O(|X| \cdot |Y|)$. Here, $|X|$ denotes the length (number of characters) in the string X.

due: april 9th

so that we can cover:

- nearest-neighbor search
- spectral algorithms (e.g. pagerank)
- online algorithms (multiplicative update)
- geometric hashing
- \+ graph algorithms, data structures, network flow, hashing, NP-completeness, linear programming, approx. algorithms

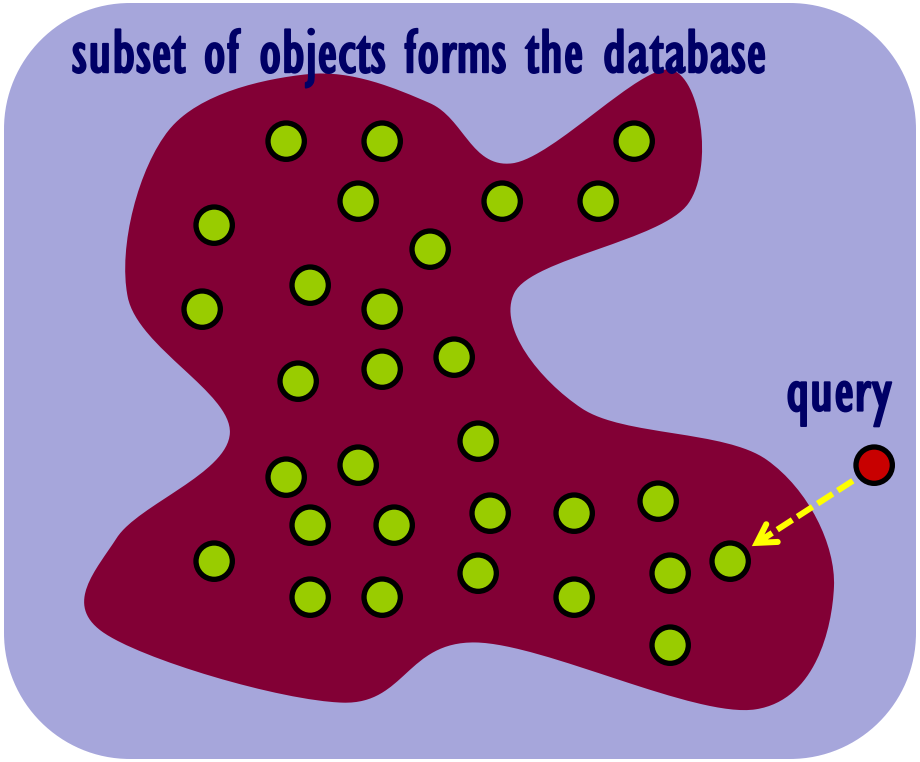
case study: nearest-neighbor search



Goal:

Quickly respond with the database object most similar to the query.

formal model



Goal:

Quickly respond with the database object most similar to the query.

U = universe (set of objects)

$d(x,y)$ = distance between two objects

Assumptions:

$d(x,x) = 0$ for all $x \in U$

$d(x,y) = d(y,x)$ for all $x,y \in U$

(symmetry)

$d(x,y) \leq d(x,z) + d(z,y)$

for all $x,y,z \in U$

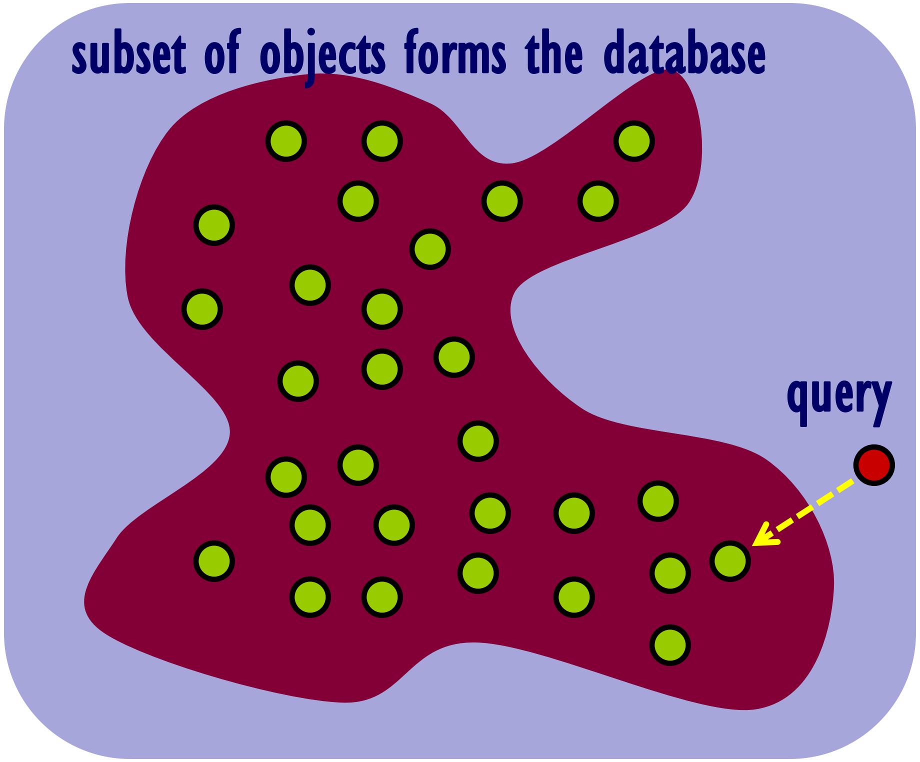
(triangle inequality)

Problem:

Given an input database $D \subseteq U$:

preprocess D (fast, space efficiently) so that queries $q \in U$ can be answered very quickly, i.e. return $a^* \in D$ such that $d(q,a^*) = \min \{ d(q, x) : x \in D \}$

other considerations

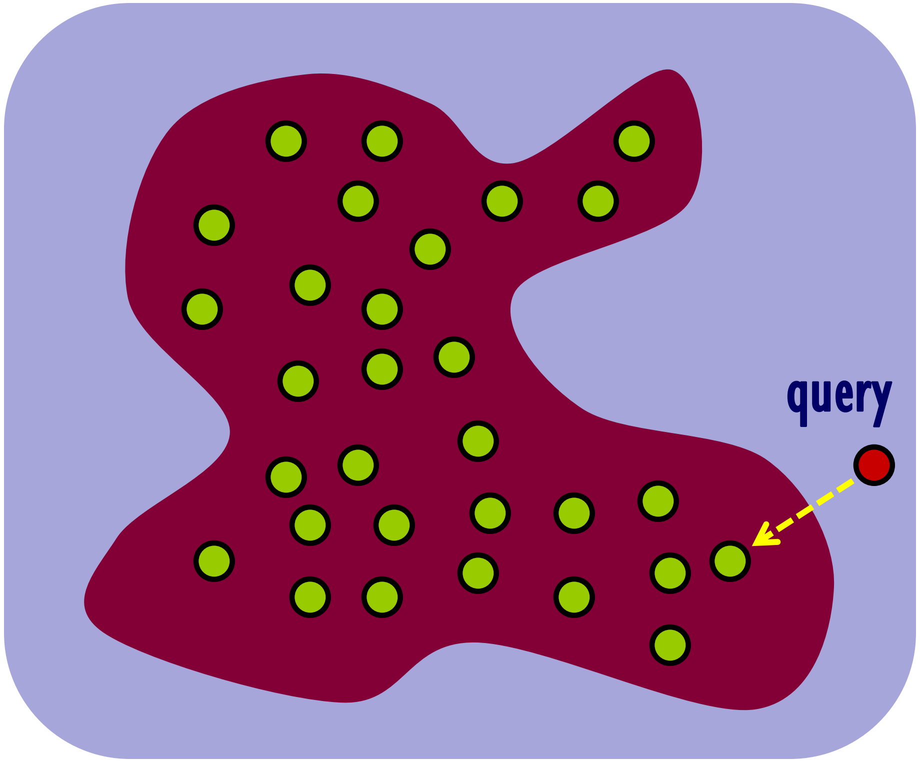


Is it expensive to compute d(x,y)?

Problem:

Given an input database $D \subseteq U$:
preprocess D (fast, space efficiently) so that queries
$q \in U$ can be answered very quickly, i.e. return
$a^* \in D$ such that $d(q,a^*) = \min \{ d(q,x) : x \in D \}$

Should we treat the distance function and objects as a black box?



[Brute force: Time]
Compute **d(query, x)** for every object **x** $\in$ **D**, and return the closest.

Takes **time** $\approx$
|D| · (distance comp. time)

[Brute force: Space]
Pre-compute best response to every possible query **q** $\in$ **U**.

Takes **space** $\approx$
|U| · (object size)

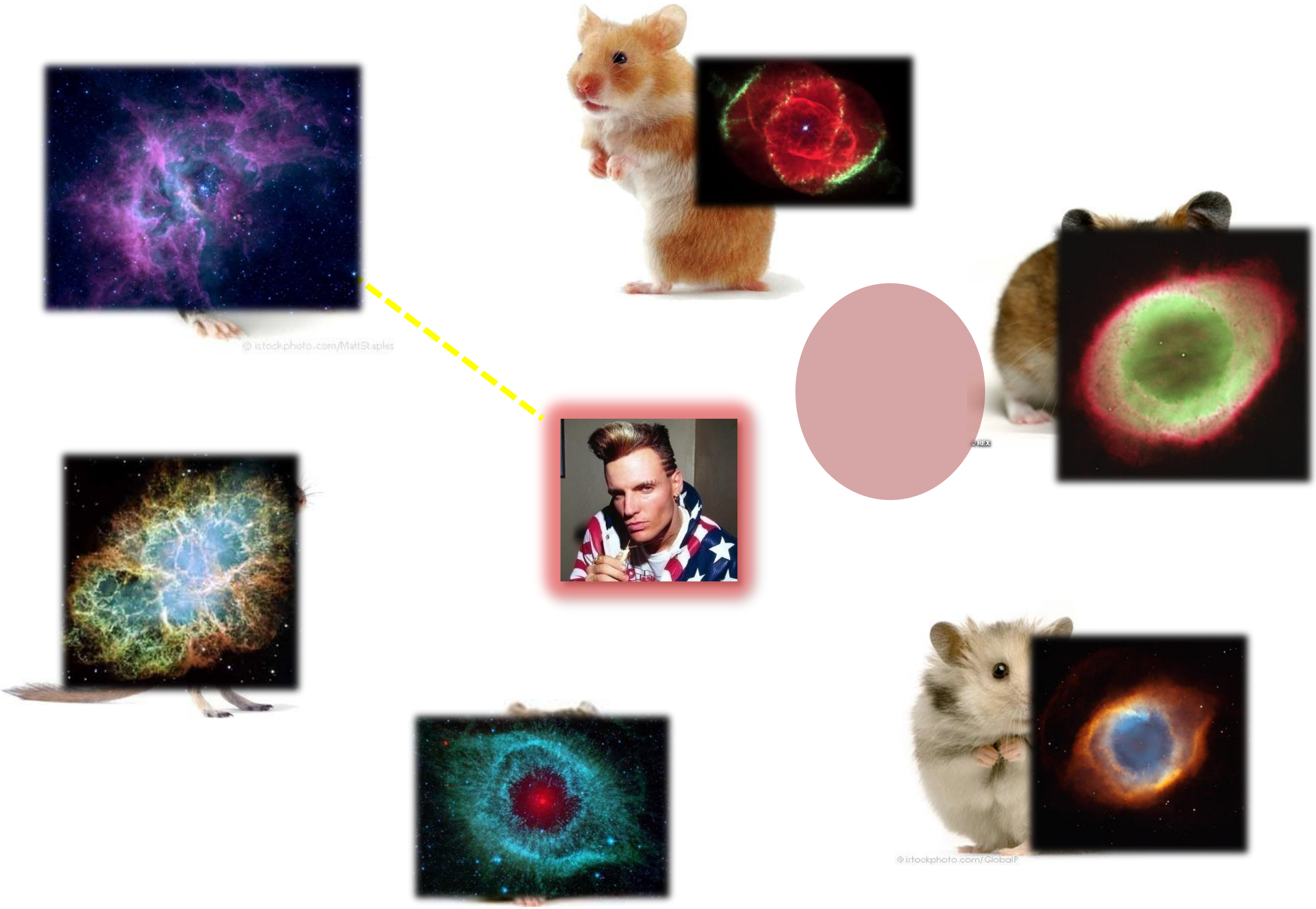
Dream performance:

$O(\log |D|)$ **query time**

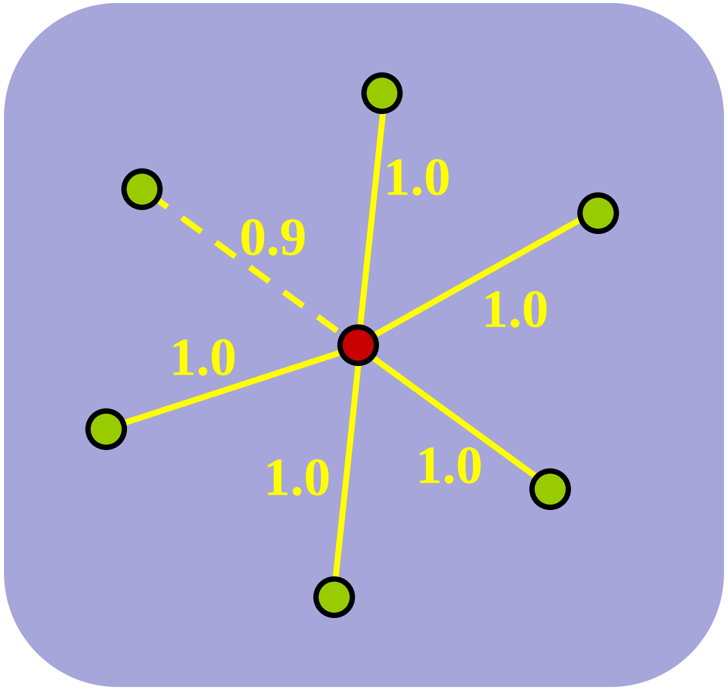
$O(|D|)$ **space**

something hard, something easy

something hard, something easy



something hard, something easy

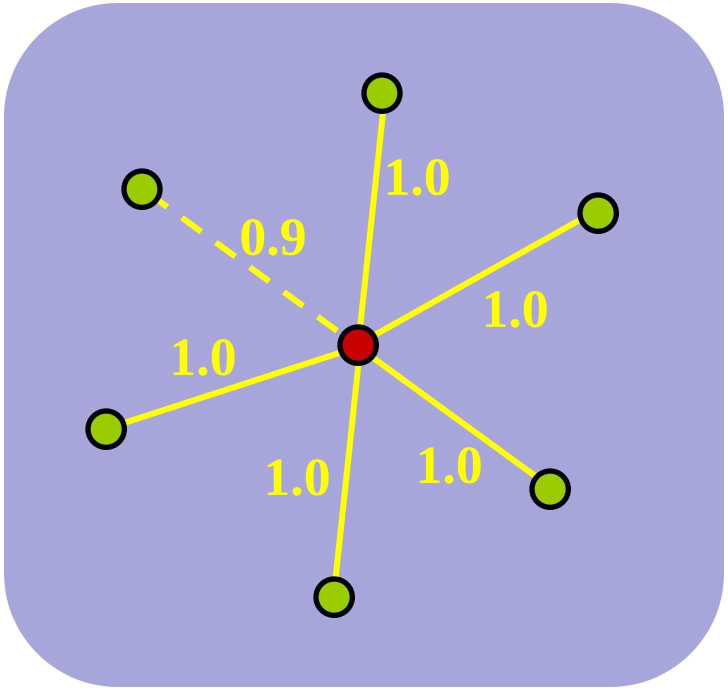


"CURSE OF DIMENSIONALITY"

All pairwise distances are equal:
$d(x,y) = 1$ for all $x,y \in D$

Problem:

… so that queries $q \in U$ can be answerd quickly, i.e. return
$a^* \in D$ such that $d(q,a^*) = \min \{ d(q,x) : x \in D \}$



All pairwise distances are equal:
$d(x,y) = 1$ for all $x,y \in D$

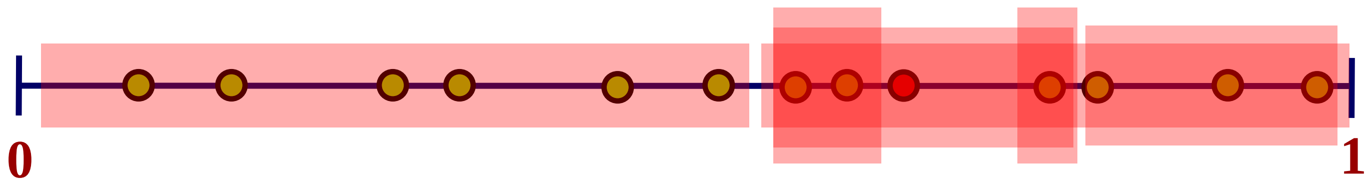
"CURSE OF DIMENSIONALITY"

Can sometimes solve exact NNS by first finding a good approximation

ϵ-Problem:

… so that queries $q \in U$ can be answerd quickly, i.e. return
$a \in D$ such that $d(q,a) \leq (1+\epsilon)\, d(q,D)$

Let's suppose that **U = [0,1]** (real numbers between 0 and 1).



Answer: Sort the points **D ⊆ U** in the preprocessing stage.
To answer a query **q ∈ U**, we can just do binary search.

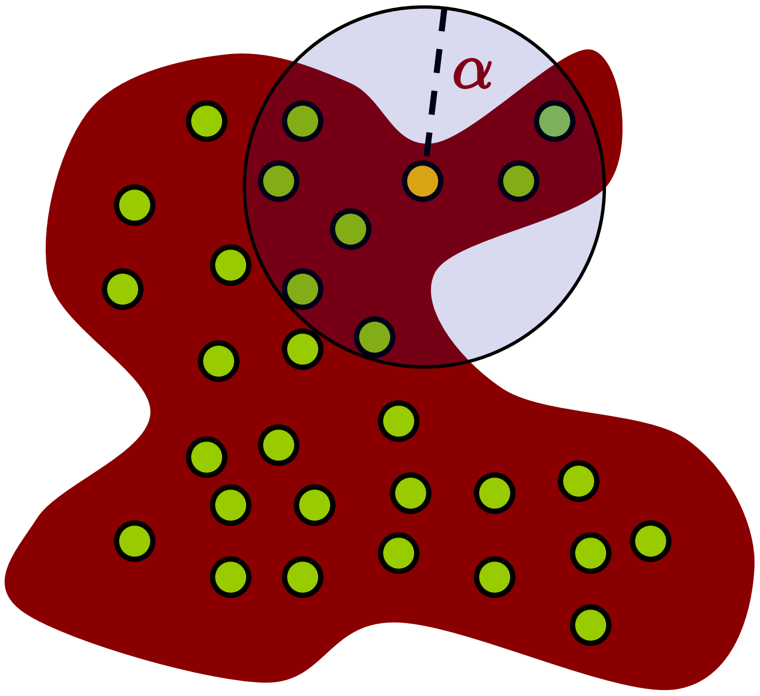
To support insertions/deletions in $O(\log |D|)$ time, can use a **BST**.
(balanced search tree)

How much power did we need?

Can we do this just using distance computations **d(x,y)**? (for **x,y ∈ D**)

Basic idea: Make progress by throwing "a lot" of stuff away.

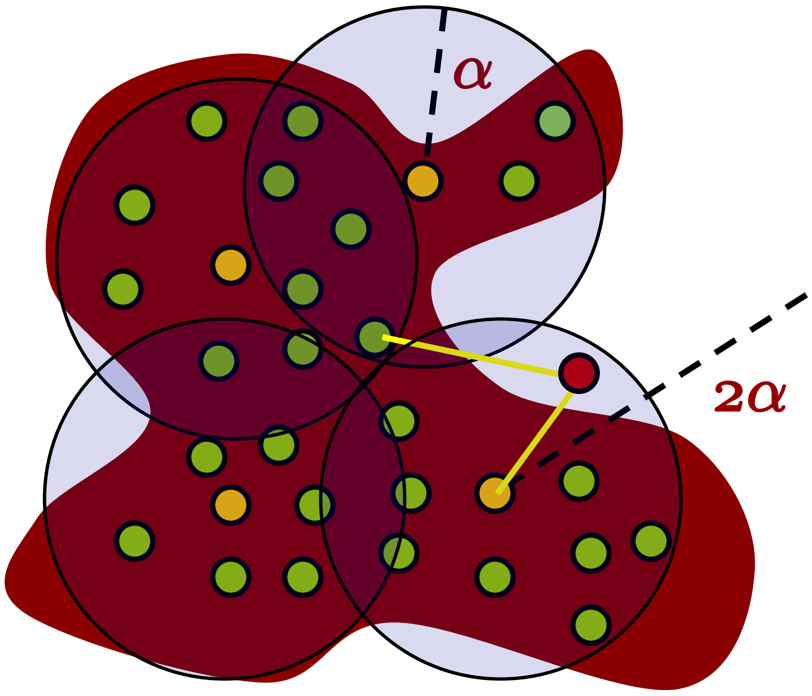
extending the basic idea



Definition: The **ball of radius** α **around** $\mathbf{x} \in \mathbf{D}$ is

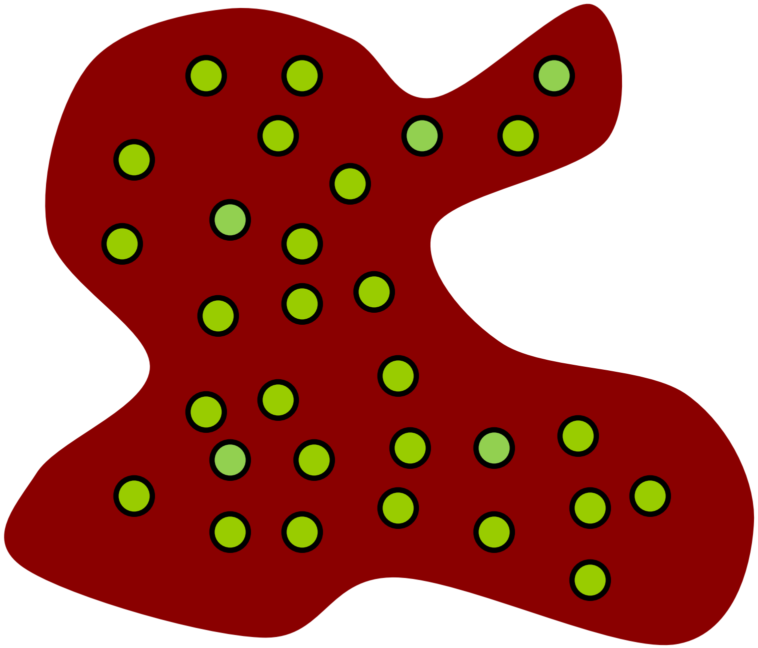
$$B(x, \alpha) = \{y \in D : d(x, y) \leq \alpha\}$$

extending the basic idea



Definition: The **ball of radius** α **around** $\mathbf{x} \in \mathbf{D}$ is

$$B(x, \alpha) = \{y \in D : d(x, y) \leq \alpha\}$$



Greedy construction algorithm:

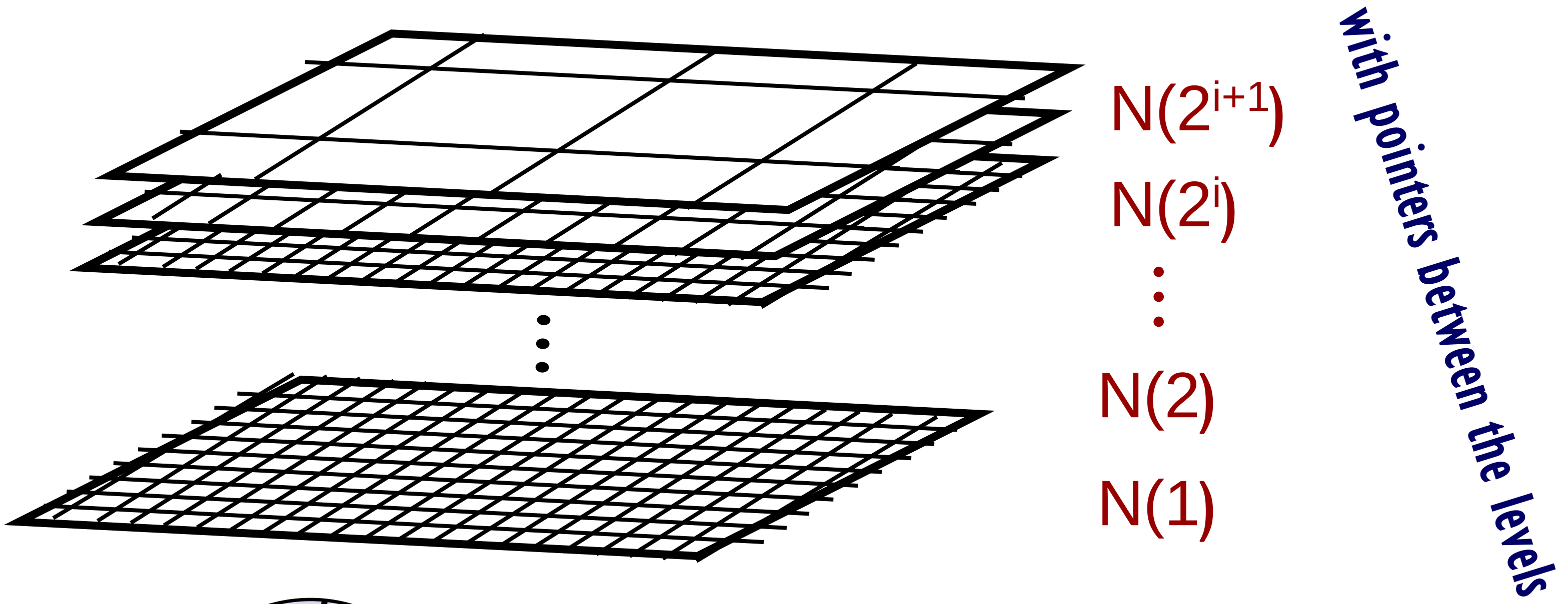
Start with $N = \emptyset$.
As long as there exists an $x \in D$ with $d(x,N) > \alpha$, add x to N.

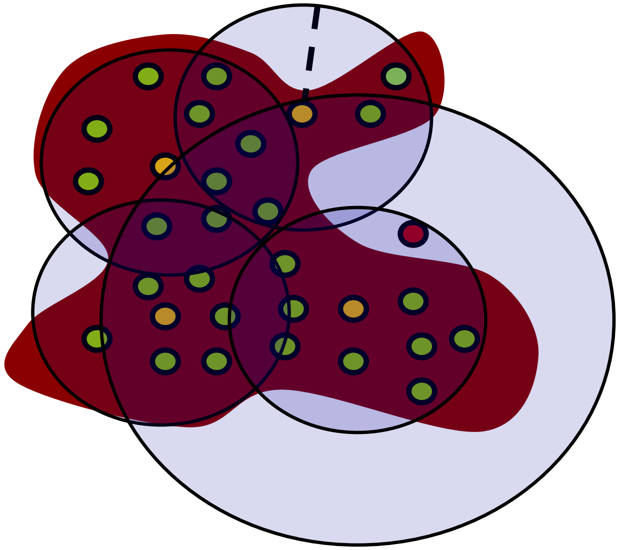
So for every $\alpha > 0$, we can construct an α-net $N(\alpha)$ in $O(n^2)$ time, where $n = |D|$.

Definition: An **α-net in** D is a subset $N \subseteq D$ such that

1) **Separation:** For all $x,y \in N$, $d(x,y) \geq \alpha$
2) **Covering:** For all $x \in D$, $d(x,N) \leq \alpha$

basic data structure: hierarchical nets





Search algorithm:

Use the α-net to find a radius 2α ball.

Use the $\alpha/2$-net to find a radius α ball.

Use the $\alpha/4$-net to find a radius $\alpha/2$ ball.

basic data structure: hierarchical nets

Data structure:

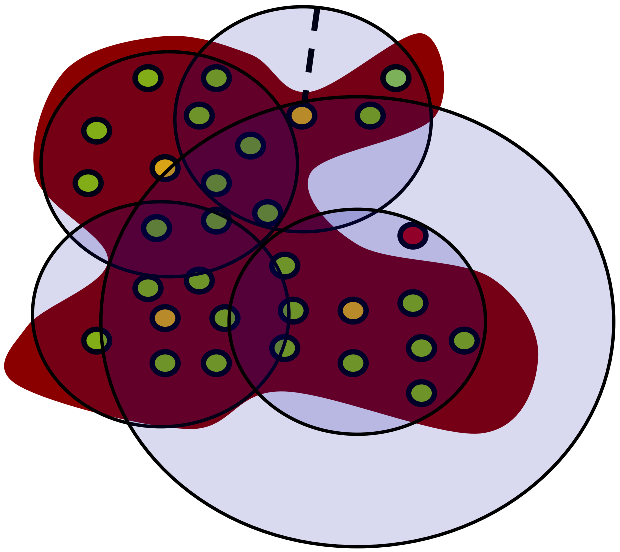
$$d_{\mathsf{max}} = \mathsf{max}\left\{d(x,y) : x,y \in D\right\}$$

$$d_{\mathsf{min}} = \mathsf{min}\left\{d(x,y) : x \neq y \in D\right\}$$

For $i = \log(d_{\mathsf{min}}), \log(d_{\mathsf{min}}) + 1, \ldots, \log(d_{\mathsf{max}})$,

let N_i be a 2^i-net.

For each $x \in N_i$, $L_{x,i} = B(x, 2^{i+1}) \cap N_{i-1}$.



algorithm: traverse the nets

Data structure:

$$d_{\max} = \max\{d(x,y) : x, y \in D\}$$

$$d_{\min} = \min\{d(x,y) : x \neq y \in D\}$$

For $i = \log(d_{\min}), \log(d_{\min}) + 1, \ldots, \log(d_{\max})$,

let N_i be a 2^i-net.

For each $x \in N_i$, $L_{x,i} = B(x, 2^{i+1}) \cap N_{i-1}$.

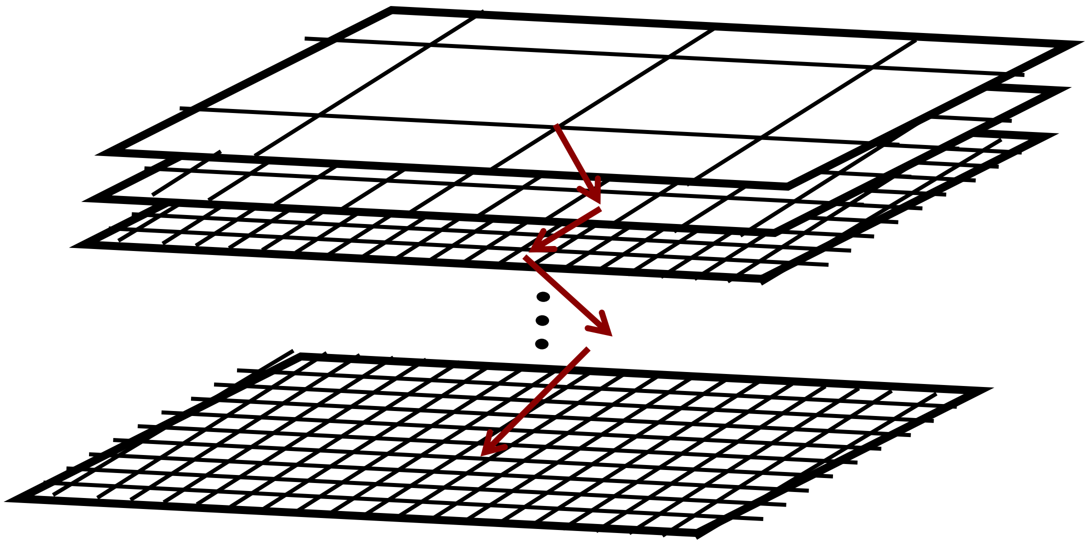
Algorithm: Given input query q ∈ U,

Let CurrentPoint = only point of $N_{\log(d_{\max})}$.

For $i = \log(d_{\max})-1, \log(d_{\max})-2, \ldots, \log(d_{\min})$,

CurrentPoint = closest point to q in $L_{\text{CurrentPoint},i}$

algorithm: traverse the nets



Algorithm: **Given input query q ∈ U,**

Let CurrentPoint = only point of $N_{\log(d_{\max})}$.

For $i = \log(d_{\max})-1, \log(d_{\max})-2, \ldots, \log(d_{\min})$,

CurrentPoint = closest point to q in $L_{\text{CurrentPoint},i}$

running time analysis?

Query time = $O\left(\log\left(\frac{d_{\max}}{d_{\min}}\right)\right) \max\left\{|L_{x,i}| : x \in D, i\right\}$

$$L_{x,i} = B(x, 2^{i+1}) \cap N_{i-1}$$

Nearly uniform point set:

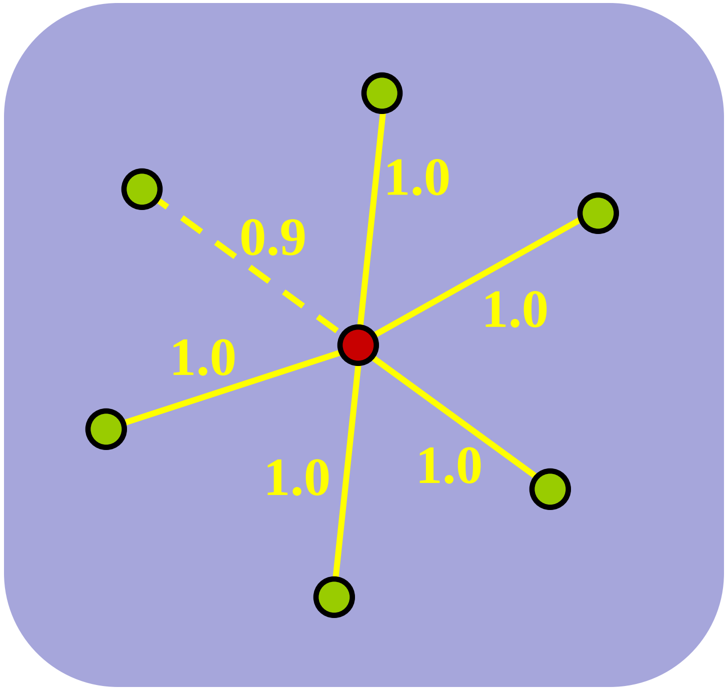
For $u,v \in L_{x,i}$, $d(u,v) \in [2^{i-1}, 2^{i+2}]$

Algorithm: Given input query $q \in U$,

Let CurrentPoint $=$ only point of $N_{\log(d_{\max})}$.

For $i = \log(d_{\max})-1, \log(d_{\max})-2, \ldots, \log(d_{\min})$,

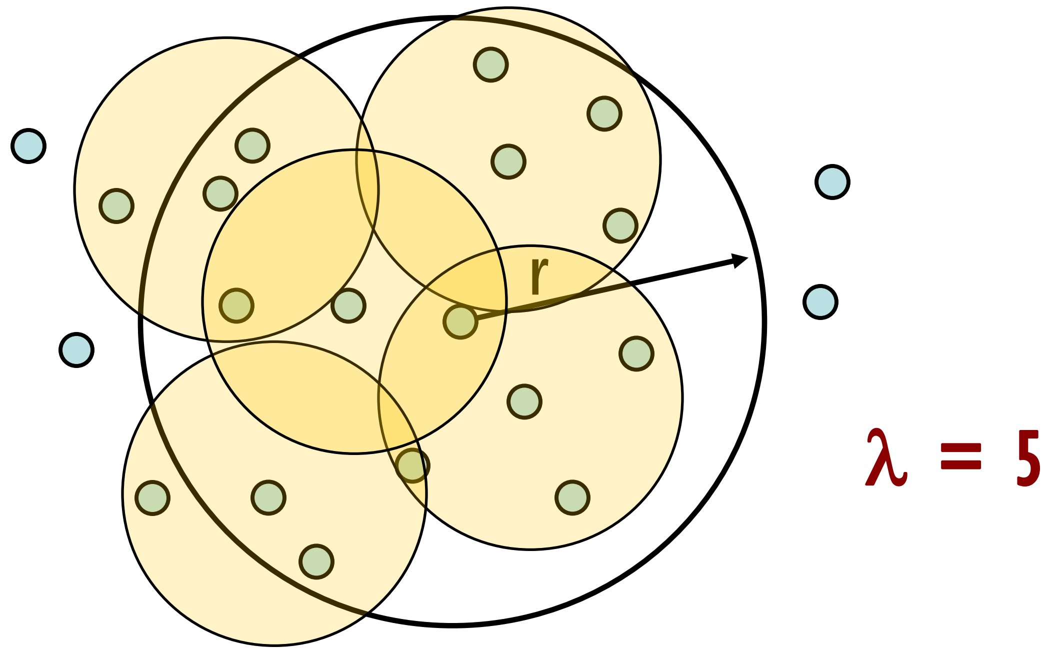
CurrentPoint $=$ closest point to q in $L_{\text{CurrentPoint},i}$



All pairwise distances are equal:
$d(x,y) = 1$ for all $x,y \in D$

"CURSE OF DIMENSIONALITY"

Given a metric space **(X,d)**, let λ**(X,d)** be the smallest constant λ such that every ball in **X** can be covered by λ balls of half the radius.



The **intrinsic dimension** of **(X,d)** is the value

$$\dim(X, d) = \log_2 \lambda(X, d)$$

We can bound the query time of our algorithm in terms of the **intrinsic dimension** of the data…

Query time = $O\left(\log\left(\frac{d_{\max}}{d_{\min}}\right)\right) \max\left\{|L_{x,i}| : x \in D, i\right\}$

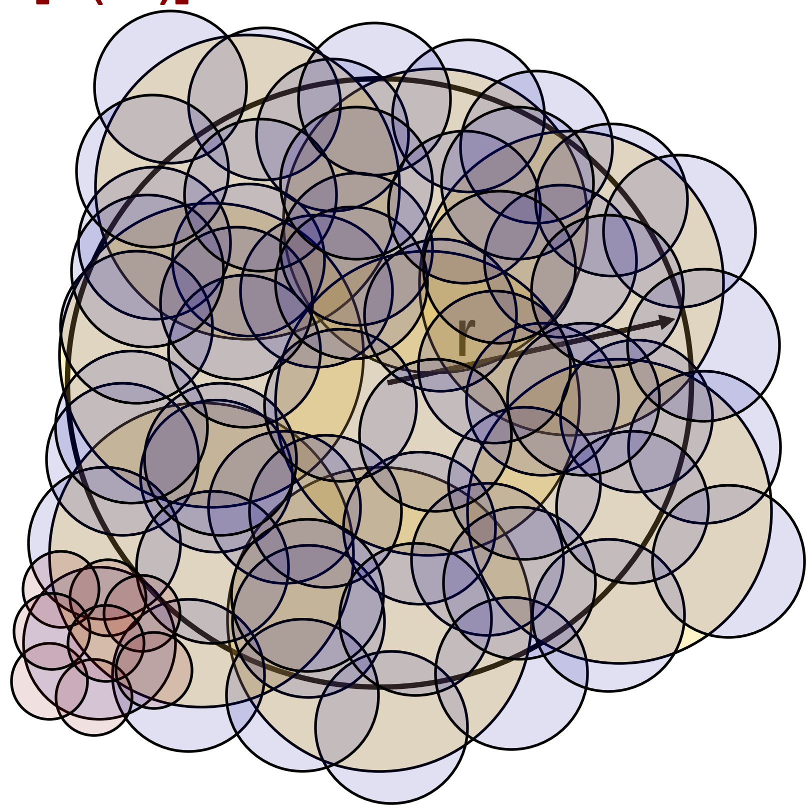
$$L_{x,i} = B(x, 2^{i+1}) \cap N_{i-1}$$

Claim: $|L_{x,i}| \leq [\lambda(X,d)]^3$

Proof: Suppose that $k = |L_{x,i}|$. Then we need at least k balls of radius 2^{i-2} to cover $B(x,2^{i+1})$, because a ball of radius 2^{i-2} can cover **at most one** point of N_{i-1}.

But now we claim that (for any r) every ball of radius r in X can be covered by at most $[\lambda(X,d)]^3$ balls of radius $r/8$, hence $k \leq [\lambda(X,d)]^3$.

But now we claim that (for any **r**) every ball of radius **r** in **X** can be covered by at most $[\lambda(X,d)]^3$ balls of radius **r/8**, hence $k \leq [\lambda(X,d)]^3$.



A ball of radius **r** can be covered λ balls of radius **r/2**, hence by

λ^2 balls of radius **r/4**, hence by

λ^3 balls of radius **r/8**.

We can bound the query time of our algorithm in terms of the **intrinsic dimension** of the data…

$$\text{Query time} = O\left(\log\left(\frac{d_{\max}}{d_{\min}}\right)\right) \max\left\{|L_{x,i}| : x \in D, i\right\}$$

$$L_{x,i} = B(x, 2^{i+1}) \cap N_{i-1}$$

Claim: $|L_{x,i}| \leq [\lambda(X,d)]^3$

Proof: Suppose that $k = |L_{x,i}|$. Then we need at least k balls of radius 2^{i-2} to cover $B(x,2^{i+1})$, because a ball of radius 2^{i-2} can cover **at most one** point of N_{i-1}.

But now we claim that (for any r) every ball of radius r in X can be covered by at most $[\lambda(X,d)]^3$ balls of radius $r/8$, hence $k \leq [\lambda(X,d)]^3$.

We can bound the query time of our algorithm in terms of the **intrinsic dimension** of the data…

Query time = $O\left(\log\left(\frac{d_{\max}}{d_{\min}}\right)\right)[\lambda(X,d)]^3$

$$L_{x,i} = B(x, 2^{i+1}) \cap N_{i-1}$$

Claim: $|L_{x,i}| \leq [\lambda(X,d)]^3$

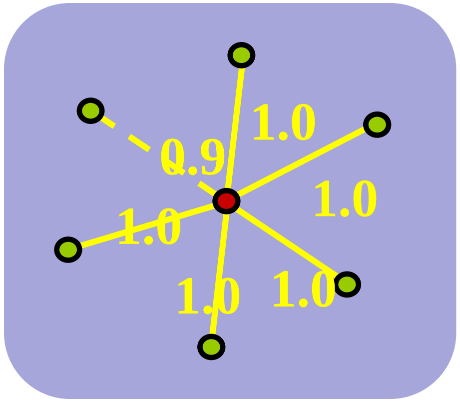
Proof: Suppose that $k = |L_{x,i}|$. Then we need at least k balls of radius 2^{i-2} to cover $B(x,2^{i+1})$, because a ball of radius 2^{i-2} can cover **at most one** point of N_{i-1}.

But now we claim that (for any r) every ball of radius r in X can be covered by at most $[\lambda(X,d)]^3$ balls of radius $r/8$, hence $k \leq [\lambda(X,d)]^3$.

We can bound the query time of our algorithm in terms of the **intrinsic dimension** of the data…

Query time = $O\left(\log\left(\frac{d_{\max}}{d_{\min}}\right)\right)[\lambda(X,d)]^3$

- Generalization of binary search (where dimension **= 1**)
- Works in arbitrary metric spaces with small intrinsic dimension
- Didn't have to think in order to "index" our database
- Shows that the hardest part of nearest-neighbor search is



wrap up

- **Only gives approximation to the nearest neighbor**
- **Next time:** Fix this; fix time, fix space + data structure prowess
- **In the future:** Opening the black box; **NNS in high dimensional spaces**
- **Bonus: Algorithm is completely intrinsic** (e.g. isomap)