# 1 Approximate K-Nearest Neighbors (kNN) Search for High-Dimensional Vectors

total time limit: 50 seconds
querying time limit per test: 5 seconds
number of test cases: 2
memory limit per test: 500 megabytes (1000 megabytes for Python)

## 1.1 Background

Vector databases store high-dimensional vectors which can be videos or text embedding used for deep learning models. A major challenge for vector databases is to implement a fast algorithm to search nearest neighbors, even with approximation.

## 1.2 Task

The task of this competition is to implement a vector search engine that creates an index over numeric high-dimensional vectors so that given a query vector, the search engine can efficiently return its top-k nearest neighbors, meaning the k closest vectors. To specify, given a set of vectors $S \subset \mathbb{R}^D$ and a point $x \in \mathbb{R}^D$, the top-k nearest neighbors $S_x$ are defined as a set such that

$$|S_x| = k; \quad \min_{p \in S/S_x} d(x,p) \geq \max_{p' \in S_x} d(x,p').$$

The closeness is defined by squared Euclidean distance with formula given below:

$$d(x,y) = ||x-y|| = \sqrt{\sum_{i=1}^{D}(x_i - y_i)^2}$$

pair of vectors with less distance are considered closer to each other.
The data will be provided prior to queries, and should be processed with the intent that a query can be answered with low latency. Queries will be provided one at a time, and it should be answered by ids of the k closest vectors. Vector id is the ordering of data vectors, the first vector is assigned id 0. To emphasize the significance that a query should be responded in low latency, there will be an independent timer for dealing with queries, and responses must finish within a 5-seconds interval in total. Each test case contains 800 queries.

## 1.3 Input

The input consists of two parts: data and queries. The data part has $(2 + N)$ lines, which contains the number of vector $N$, the dimension of vectors $D$, and data as vectors $\{ d_i \}_{0 \leq i < N}$. The query part has at most $(2 + 800)$ lines, which contains the number of vectors asked for each query $k$, queries as vectors

$\{ q_i \}_{0 \le i < 800}$, and a string "end". The details are as follows:
**Data Part**:
The first line contains the number of vector $N$, an integer, $1000 \le N \le 50000$. Vectors are numbered from 0 to $N - 1$.
The second line contains the dimension of vector $D$, an integer, $3 \le D \le 1024$.
The next $N$ lines contain data as vectors $\{ d_i \}_{0 \le i < N}$, each vector $d_i$ takes one line and each line has $D$ entries as floats, separated by space.
**Query Part**:
The next line contains number of vectors asked for each query $k$, integer, $2 \le k \le 10$.
Then, following lines contain queries as vectors $\{ q_i \}_{0 \le i < 800}$, each vector $q_i$ takes one line and each line has $D$ entries as floats, separated by space. Queries are given line by line, the next line can only be received once after outputting the answer of the previous query.
The last line is a string "end", notifying the program to stop, it is possible to receive the "end" early if queries time out.
It is guaranteed that data vectors and query vectors contain no duplicates, and entries of vectors can be parsed by 4-byte floats.

## 1.4 Output

Output should have $(1 + Q)$ lines where $Q$ is number of queries, once the data part is processed, print "ok" in one line **(make sure to flush it)** to signify that the program is ready to answer queries. **Once entering the query part after printing "ok", participants will be given only 5 seconds to answer all queries.**
Then for each query vector, return the vector id of top-k nearest neighbors, vector id is defined as the ordering of input vector data, starting from 0. The response for each query should be in one line with id numbers separated by space, and the order is not considered.
As mentioned before, queries are given interactively, participants must answer the current query in order to get the next query. Note that it's better to flush stdout every time to ensure that the answer is actually delivered.
Please note each answer must include k and only k vector ids, incorrect format will result in a 0 score.

## 1.5 Example

| input | output |
|---|---|
| 4<br>3<br>1.0 1.0 1.0<br>1.0 1.0 2.0<br>1.0 1.0 3.0<br>2.0 2.0 -1.0<br>2<br>5.5 5.5 0<br>end | ok<br>0 3 |

Explanation: the top-2 nearest vector to query $(5.5, 5.5, 0)$ is vector $d_0$ $(1.0, 1.0, 1.0)$ and vector $d_3$ $(2.0, 2.0, -1.0)$, so the expected answer is "0 3". ("3 0" is also correct.)

## 1.6 Scoring

The goal is to maximize the accuracy of k-NN search while responding queries with low latency. The score will be given based on the recall of every output top-k nearest vector ids. The scoring formula is shown as:

$$\text{score} = \sum_{q=0}^{Q-1} r(\text{ans}_q),$$

where the reward function for each output answer is

$$r(\text{ans}_q) = \begin{cases} 1 & |\text{ans}_q \cap \text{ref}_q| = k \\ 0.5 & |\text{ans}_q \cap \text{ref}_q| \geq \lceil 0.8k \rceil \\ 0 & \text{otherwise} \end{cases},$$

where $\text{ref}_q$ is the actual top-k nearest vector ids and always unique.
**The scorer will not accept any answers 5 seconds after providing the first query.** After 5 seconds, even if not all queries are answered, the answered part can still get scored based on the above formula while the unanswered part will be scored 0.
The total score will be the sum of scores on each test. To break a tie, programs that answer all queries in time will be given an additional score based on how fast all queries are answered, less querying time results in higher score, and the total rewarded score will always be strictly less than 0.5.

## 1.7 Note

Test cases are extracted from real world data, and we do not focus on edge cases, such as precision/overflow issues or extremely skewed distributions.