**DOCUMENTATION for Assignment 1 –
Software Development for Algorithmic Problems**
TEAM MEMBERS:
Damianakis Damianos, Lekes Athanasios

The project has been implemented in the C++ programming language using the g++ compiler. The project is divided into three folders (lsh, hypercube, and cluster), each containing a main function, appropriate tools/structures, and a makefile.

For data organization, we have used linked lists and hash tables. These tools are separated into distinct .cpp files, each accompanied by a header file. Regarding input and command line control, we have created a suitable function in the others.c file.

Various functions that are used throughout the project are included in the others.c file, such as calculating Euclidean distance, power, initialization of the k nearest neighbors, etc. Additionally, for better project organization, we have created a separate .h file, includes.h, which includes all the necessary libraries to avoid repetition in the files. Furthermore, the parsingInit.cpp file contains the parsInit function used for reading the input files, the input file, and the query file.

Note: Due to time constraints, memory deallocation has not been addressed, resulting in memory leaks. This is not overlooked or considered negligible; it's just that we didn't have time to address these issues. The README file might appear rushed as it was written just 20 minutes before the deadline.

LSH:
The LSH algorithm can be found in the LSH folder. We have chosen to separate the algorithms into folders both because they perform different tasks and to avoid having too much information in a single folder. One improvement could be to create a separate folder containing common header files for algorithms like others.c, to avoid duplications across folders.

In terms of the LSH algorithm's data structures, the hash table follows the typical logic of such a structure, with differences lying in its attributes. Specifically, you will find an array vAndT representing the v vectors with their corresponding t values. There's also a struct lshCon containing constants for LSH.

Each bucket of the hash table is a linked list of linkedLlistNode* structures, and for each hash function, an associated R vector is added in the form of a vector<int>.

Regarding the functions of a hashTable object, apart from obvious helper methods, we have findKNeighbors, findKNeighborsTrue, and findKNeighborsR, implementing the processes of finding nearest neighbors using KNN, Brute Force, and Range Search, respectively.

Another structure needing clarification is KNearest. This struct represents the array of nearest neighbors returned by a search process (range search, KNN, brute force). To add more information to KNearest, we chose to create a separate structure storing both the

points (with pointers) that constitute the nearest neighbors and their distances from the query.
To run the program:
- Type `make`
- Type `make run`

HYPERCUBE:
The structure of HyperCube, perhaps the most demanding part of the project, naturally follows the same data structures (hashTable, linkedList) as the first algorithm, but adapted for the HyperCube algorithm.

In HyperCube, you'll find everything contained in an LSH algorithm's hashTable, with the addition of a vector containing 0s and 1s representing the hypercube's vertex.

Helper methods are present, as well as the findKNeighbors(x) functions, which execute the same processes. There's also a decToBin function, defined in the context of defining the vector with 0s and 1s determining my bucket.

To run the program:
1. Type `make`
2. Type `make run`

CLUSTER:
There are several obvious changes made to the cluster files. Initially, we defined separate cluster.h and cluster.cpp files, where we define and implement all functions related to clustering (lloyd, lsh reverse, hypercube reverse, etc.), as well as a centroid class representing the centroid.

Notably, inside the cluster folder, you'll find both hashtable.h and hashtablecube.h, along with their corresponding source code files. This is because LSH and hypercube don't use the exact same hash table format.

To run different cluster algorithm variants, use the following commands:
- `make`
  1. Lloyd
- `make runclassic`
  2. Reverse LSH
- `make runlsh`
  3. Reverse Hypercube
- `make runhyper`

The silhouette has also been implemented for the provided files, with values ranging from 0.3 to 0.5.

Thank you,
Athanasios Lekes, Damianos Damianakis.