Sorted Set

HackerRank

At Quora, we rely on a sorted set data structure for many of our ranking systems, such as those powering answerer suggestions. This data structure comes out-of-the-box in the popular open-source system Redis. However, Redis is explicitly designed to be single-threaded in order to keep the code and system simple. Given the amount of data we store and access patterns for retrieving it later, we want to build a faster version that is able to support the same sorted set abstraction.

The goal of this task is to implemented a networked sorted set that handles positive integers. Your program should listen on a Unix domain socket at "./socket", accept new connections from multiple clients, and process commands that are sent over these connections via the binary protocol described below. The program should be running, even after all connections have been disconnected, until it is terminated by the test system. Each `<...>` represents a four byte unsigned integer in network byte order* sent or received on the socket. All client -> server and server -> client commands are prefixed with the number of fields in the command (detailed below). All set *ids*, *keys*, and *scores* will be positive.

Your server should implement the following commands:

1. *Add Score:* Adds member `<key>` to `<set>`, with score `<score>`. If `<set>` doesn't exist, it's created. If `<key>` is already in `<set>`, its score is updated.
   - Client: `<4> <1> <set> <key> <score>`
   - Server: `<0>`
2. *Remove Key:* Removes `<key>` from `<set>` if `<set>` exists and `<key>` is in `<set>`.
   - Client: `<3> <2> <set> <key>`
   - Server: `<0>`
3. *Get Size:* Returns the size of set `<set>`, or `0` if `<set>` doesn't exist.
   - Client: `<2> <3> <set>`
   - Server: `<1> <size>`
4. *Get key-value:* Returns the score of key `<key>` in `<set>`, and `0` if either the set does not exist or does not contain `<key>`.
   - Client: `<3> <4> <set> <key>`
   - Server: `<1> <score>`
5. *Get Range:* Returns all elements in sets `<set1> ... <setM>` with scores in the range `[<lower>, <upper>]`. Elements should be returned sorted by non-decreasing order of key. If two keys match, the elements with matching keys should be sorted by non-decreasing order of value.
   - Client: `<N> <5> <set1> ... <setM> <0> <lower> <upper>`
   - Server: `<K> [<key> <score>]` (repeat for each element of the set returned, where K is the total number of integers returned)
6. *DISCONNECT:*
   - Client: `<1> <6>`
   - Server: No response, then disconnect the client

The program should support up to 8 parallel connections. For multithreading:

- *(C, C++)* - use the POSIX thread library (pthread.h), fork() is disabled.

- *(Python, Java, etc)* - use the language's default threading library.

Input

Input will be sent over a Unix domain socket in the form of four byte unsigned integer in network byte order*. Each query is preceded with a length header which represents number of integers contained in the query. (These are the first integers in each of the queries in the protocol documentation above.)

Output

Output should be sent over the same UDS socket in the form of four byte unsigned integer in network byte order. Output of each operation should be send in the order in which that operation is called in input.

Sample Input

```
4 1 10 5 346248660
5 5 10 0 122145695 1413614583
3 4 10 5
4 1 5 8 1427404471
4 1 2 1 1761359496
4 1 2 4 631790567
3 4 2 4
3 4 2 1
4 1 7 3 1989599602
2 3 2
3 2 2 1
2 3 2
9 5 2 3 5 7 10 0 113558404 1516732979
1 6
```

Sample Output

```
0
2
5
346248660
1
346248660
0
0
0
1
631790567
1
1761359496
0
1
2
0
1
1
6
4
631790567
5
346248660
8
1427404471
```

Explanation

The first line of input has the values

```
4 1 10 5 346248660
```

`4` denotes the number of values in the query, `1` corresponds to the ADD command, `10` is the set ID, `5` is the key to add, and `346248660` is the score for this key.