Programming Test - Points on a line

In this exercise, the candidate will implement a software system that allows the user to manage a set of points on a one-dimensional line. Users of this software can add points to the line, list all points, delete all points and determine which points are the closest to a given point.

The line is one-dimensional, thus points are represented by their offset with respect to an arbitrarily chosen point on the line, known as the *origin*. When the software starts, there are no points on the line.

Problem to solve

Implement a REST API that exposes the following endpoints:

Add a point to the line

`POST /point` with body `{ "offset": <NUMBER> }`

Specifications This endpoint adds a points to the line.

The point offset with respect to the *origin* is a **real number**. The user can add the same point more than once, even though adding a point many times is the same as adding it once.

Examples The following requests:

```
POST /point with body { "offset": -1 }
POST /point with body { "offset": 5 }
POST /point with body { "offset": 2.5 }
POST /point with body { "offset": 5 }
```

Would produce the following set of points:

```
{ { "offset": -1 }, { "offset": 2.5 }, { "offset": 5 } }
```

Retrieve all points

`GET /points?order=<ASC|DESC>`

Specifications This endpoint returns all points added to the line so far.

It accepts a query parameter **order** (the value of which can be either *ASC* or *DESC*), and sorts the returned points accordingly. If the query parameter **order** is not specified it defaults to *ASC*. If the line contains no points this endpoint returns the empty list.

Examples Given that the line contains the following points:

```
[
    { "offset": 1 },
    { "offset": 3 },
    { "offset": 4 }
]
```

The request `GET /points?order=DESC` would return:

```
[
    { "offset": 4 },
    { "offset": 3 },
    { "offset": 1 }
]
```

While the request `GET /points` or `GET /points?order=ASC` would return:

```
[
    { "offset": 1 },
    { "offset": 3 },
    { "offset": 4 }
]
```

Delete all points

`DELETE /points`

Specifications This endpoint deletes all points added to the line.

Retrieve the neighbours of a given point

`GET /neighbours/{point}?k=<NUMBER>`

Specifications This endpoint returns the *k* points which are the closest to *point*.

The path parameter *point* is a **real number** representing the offset of a point (with respect to the *origin*) which must not necessarily be on the line. The query parameter *k* is a **non-negative integer** which, if missing, defaults to 1.

If *k* is zero, this endpoint returns the empty list. If it is larger than the number of points currently on the line, this endpoint returns all of them. Finally, if two points on the line are equidistant from the given point, this endpoint always returns the leftmost point first (w.r.t. the *origin*).

Examples Given the following points:

```
[
    { "offset": 1 },
    { "offset": 3 },
    { "offset": 4 }
]
```

The following requests would produce the following responses:

Request	Response
`GET /neighbours/3?k=0`	`[]`
`GET /neighbours/3`	`[{ "offset": 3 }]`
`GET /neighbours/3?k=1`	`[{ "offset": 3 }]`
`GET /neighbours/3?k=2`	`[{ "offset": 3 }, { "offset": 4 }]`
`GET /neighbours/4?k=2`	`[{ "offset": 4 }, { "offset": 3 }]`
`GET /neighbours/3.5`	`[{ "offset": 3 }]`
`GET /neighbours/3.5?k=2`	`[{ "offset": 3 }, { "offset": 4 }]`
`GET /neighbours/0?k=100`	`[{ "offset": 1 }, { "offset": 3 }, { "offset": 4 }]`

Additional rules

- All code should be under version control, on a publicly accessible git repository (e.g., a GitHub repository);
- Unless specified in the instructions above, the API should consume and produce JSON;
- The solution must be in one of the following programming languages: Java, Scala, Kotlin and JavaScript.

Suggestions

- Properly naming variables and documenting the code can help us understand your solution;
- Validating all inputs to your program will help your solution pass our test cases;
- There is no bound on the computational complexity of the solution, but solutions with good computational complexity will earn you bonus points.