quickselect

Release 0.2.0

Azat Ibrakov

Apr 23, 2020

CONTENTS

Note: If object is not listed in documentation it should be considered as implementation detail that can change and should not be relied upon.

CHAPTER

ONE

FLOYD_RIVEST MODULE

Based on selection algorithm by Robert W. Floyd and Ronald L. Rivest.

Reference: https://en.wikipedia.org/wiki/Floyd%E2%80%93Rivest_algorithm

quickselect.floyd_rivest.**nth_largest**(*sequence: MutableSequence[Domain]*, *n: int*, ***, *key: Optional[Callable[[Domain], Any]] = None*) → Domain

Returns n-th largest element and partially sorts given sequence while searching.

complexity	best	average	worst
time	`O(size)`	`O(size)`	`O(size ** 2)`
memory	`O(1)`	`O(log size)`	`O(size)`

where `size = len(sequence)`.

Parameters

- **sequence** – sequence to search in
- **n** – index of the element to search for in the sequence sorted by key in descending order (e.g. `n = 0` corresponds to the maximum element)
- **key** – single argument ordering function, if none is specified compares elements themselves

Returns n-th largest element of the sequence

```
>>> sequence = list(range(-10, 11))
>>> nth_largest(sequence, 0)
10
>>> nth_largest(sequence, 1)
9
>>> nth_largest(sequence, 19)
-9
>>> nth_largest(sequence, 20)
-10
>>> nth_largest(sequence, 0, key=abs)
-10
>>> nth_largest(sequence, 1, key=abs)
-10
>>> nth_largest(sequence, 19, key=abs)
-1
>>> nth_largest(sequence, 20, key=abs)
0
```

quickselect.floyd_rivest.**nth_smallest**(*sequence: MutableSequence[Domain]*, *n: int*, ***, *key: Optional[Callable[[Domain], Any]] = None*) → Domain

Returns n-th smallest element and partially sorts given sequence while searching.

complexity	best	average	worst
time	`O(size)`	`O(size)`	`O(size ** 2)`
memory	`O(1)`	`O(log size)`	`O(size)`

where `size = len(sequence)`.

Parameters

- **sequence** – sequence to search in
- **n** – index of the element to search for in the sequence sorted by key in ascending order (e.g. `n = 0` corresponds to the minimum element)
- **key** – single argument ordering function, if none is specified compares elements themselves

Returns n-th smallest element of the sequence

```
>>> sequence = list(range(-10, 11))
>>> nth_smallest(sequence, 0)
-10
>>> nth_smallest(sequence, 1)
-9
>>> nth_smallest(sequence, 19)
9
>>> nth_smallest(sequence, 20)
10
>>> nth_smallest(sequence, 0, key=abs)
0
>>> nth_smallest(sequence, 1, key=abs)
-1
>>> nth_smallest(sequence, 19, key=abs)
-10
>>> nth_smallest(sequence, 20, key=abs)
10
```

quickselect.floyd_rivest.**select**(*sequence: MutableSequence[Domain]*, *n: int*, ***, *start: int = 0*, *stop: Optional[int] = None*, *key: Optional[Callable[[Domain], Any]] = None*, *comparator: Callable[[Domain, Domain], bool]*) → Domain

Partially sorts given sequence and returns n-th element.

complexity	best	average	worst
time	`O(size)`	`O(size)`	`O(size ** 2)`
memory	`O(1)`	`O(log size)`	`O(size)`

where `size = len(sequence)`.

Parameters

- **sequence** – sequence to select from
- **n** – index of the element to select
- **start** – index to start selection from

- **stop** – index to stop selection at
- **key** – single argument ordering function, if none is specified compares elements themselves
- **comparator** – binary predicate that defines the sorting order

Returns n-th element of the sequence with slice partially sorted by key in given order

```
>>> from operator import gt, lt
>>> sequence = list(range(-10, 11))
>>> select(sequence, 0, stop=5, comparator=gt)
-5
>>> select(sequence, 0, stop=5, comparator=lt)
-10
>>> select(sequence, 20, start=15, comparator=lt)
10
>>> select(sequence, 20, start=15, comparator=gt)
5
>>> select(sequence, 5, start=5, stop=15, key=abs, comparator=lt)
0
>>> select(sequence, 5, start=5, stop=15, key=abs, comparator=gt)
10
```

CHAPTER

TWO

HOARE MODULE

Based on "quickselect" selection algorithm by Tony Hoare.

Reference: https://en.wikipedia.org/wiki/Quickselect

quickselect.hoare.**nth_largest**(*sequence: MutableSequence[Domain]*, *n: int*, ***, *key: Optional[Callable[[Domain], Any]] = None*) → Domain

Returns n-th largest element and partially sorts given sequence while searching.

complexity	best	average	worst
time	`O(size)`	`O(size)`	`O(size ** 2)`
memory	`O(1)`	`O(log size)`	`O(size)`

where `size = len(sequence)`.

Parameters

- **sequence** – sequence to search in
- **n** – index of the element to search for in the sequence sorted by key in descending order (e.g. `n = 0` corresponds to the maximum element)
- **key** – single argument ordering function, if none is specified compares elements themselves

Returns n-th largest element of the sequence

```
>>> sequence = list(range(-10, 11))
>>> nth_largest(sequence, 0)
10
>>> nth_largest(sequence, 1)
9
>>> nth_largest(sequence, 19)
-9
>>> nth_largest(sequence, 20)
-10
>>> nth_largest(sequence, 0, key=abs)
10
>>> nth_largest(sequence, 1, key=abs)
-10
>>> nth_largest(sequence, 19, key=abs)
1
>>> nth_largest(sequence, 20, key=abs)
0
```

quickselect.hoare.**nth_smallest**(*sequence: MutableSequence[Domain]*, *n: int*, ***, *key: Optional[Callable[[Domain], Any]] = None*) → Domain

Returns n-th smallest element and partially sorts given sequence while searching.

complexity	best	average	worst
time	`O(size)`	`O(size)`	`O(size ** 2)`
memory	`O(1)`	`O(log size)`	`O(size)`

where `size = len(sequence)`.

Parameters

- **sequence** – sequence to search in
- **n** – index of the element to search for in the sequence sorted by key in ascending order (e.g. `n = 0` corresponds to the minimum element)
- **key** – single argument ordering function, if none is specified compares elements themselves

Returns n-th smallest element of the sequence

```
>>> sequence = list(range(-10, 11))
>>> nth_smallest(sequence, 0)
-10
>>> nth_smallest(sequence, 1)
-9
>>> nth_smallest(sequence, 19)
9
>>> nth_smallest(sequence, 20)
10
>>> nth_smallest(sequence, 0, key=abs)
0
>>> nth_smallest(sequence, 1, key=abs)
1
>>> nth_smallest(sequence, 19, key=abs)
-10
>>> nth_smallest(sequence, 20, key=abs)
10
```

quickselect.hoare.**select**(*sequence: MutableSequence[Domain]*, *n: int*, ***, *start: int = 0*, *stop: Optional[int] = None*, *key: Optional[Callable[[Domain], Any]] = None*, *comparator: Callable[[Domain, Domain], bool]*) → Domain

Partially sorts given sequence and returns n-th element.

complexity	best	average	worst
time	`O(size)`	`O(size)`	`O(size ** 2)`
memory	`O(1)`	`O(log size)`	`O(size)`

where `size = len(sequence)`.

Parameters

- **sequence** – sequence to select from
- **n** – index of the element to select
- **start** – index to start selection from
- **stop** – index to stop selection at
- **key** – single argument ordering function, if none is specified compares elements themselves
- **comparator** – binary predicate that defines the sorting order

Returns n-th element of the sequence with slice partially sorted by key in given order

```
>>> from operator import gt, lt
>>> sequence = list(range(-10, 11))
>>> select(sequence, 0, stop=5, comparator=gt)
-5
>>> select(sequence, 0, stop=5, comparator=lt)
-10
>>> select(sequence, 20, start=15, comparator=lt)
10
>>> select(sequence, 20, start=15, comparator=gt)
5
>>> select(sequence, 5, start=5, stop=15, key=abs, comparator=lt)
0
>>> select(sequence, 5, start=5, stop=15, key=abs, comparator=gt)
10
```

PYTHON MODULE INDEX

q

INDEX

M

N

Q

S