COMP S265F Design and Analysis of Algorithms
Lab 6: Linear Time Selection

In this lab, we implement the linear time selection algorithm, in which you will learn how to sort 5 numbers using only 7 comparisons, and how to handle the case with duplicates (i.e., the given numbers are not all distinct).

1. Linear time selection algorithm

Given a list `N` of n *distinct* numbers and an integer k (where $1 \leq k \leq n$), the function `largest(N, k)` returns the k-th largest number in $O(n^2)$ time.

```
1  def largest(N, k):
2      a = N[0]
3
4      L = [x for x in N if x > a]
5      S = [x for x in N if x < a]
6
7      len_L = len(L) + 1
8      if len_L == k:
9          return a
10     elif len_L > k:
11         return largest(L, k)
12     else:
13         return largest(S, k-len_L)
```

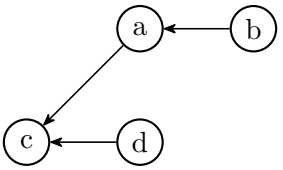
In the above function, we use a number a to divide list N into two lists L and S, where L contains all the numbers larger than a, and S contains all the numbers small than a (recall that all numbers in N are distinct). We always set a as the first number in list N.

As shown in Unit 3, we can improve the time complexity to $O(n)$ by picking a, as follows:

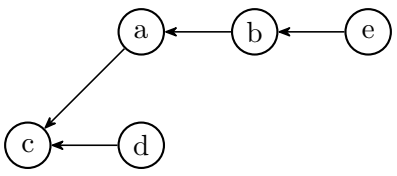
1. Divide the n numbers in N arbitrarily into $\left\lfloor \frac{n}{5} \right\rfloor$ groups, each with 5 numbers.
2. For each group, sort the 5 numbers in descending order and determine its median.
3. The number a is the median of all these $\left\lfloor \frac{n}{5} \right\rfloor$ medians.

Sorting five numbers in 7 comparisons. Given any five numbers, we can represent each number as a node, and use a *directed edge* (x, y) (i.e., $x \to y$) to indicate the relation $x \geq y$.

Then, we can obtain the following relations on four numbers using 3 comparisons:



For the remaining number x, we can determine whether $x \leq c$ or $c \leq x \leq a$ or $a \leq x \leq b$ or $b \leq x$ using 2 more comparisons (compare with a, and then with c or b). We can obtain the following relations (by relabeling the numbers):



Finally, for number d, we can determine whether $c \le d \le a$ or $a \le d \le b$ or $b \le d \le e$ or $e \le d$ using 2 more comparisons (compare with b, and then with a or e).

The following function `sort5(L)` implements the above idea on a list `L` of five numbers:

sort5.py

```
def sort5(L):
    "Sort a list L of 5 numbers in 7 comparisons"
    a, b, c, d, e = L
    if b < a:  a, b = b, a
    if d < c:  c, d = d, c
    if a < c:  a, b, c, d = c, d, a, b

    if e > a:
        if e > b:  pass
        else:      b, e = e, b
    else:
        if e < c:  c, a, b, e = e, c, a, b
        else:      a, b, e = e, a, b

    if d < b:
        if d < a:  return [c, d, a, b, e]
        else:      return [c, a, d, b, e]
    else:
        if d > e:  return [c, a, b, e, d]
        else:      return [c, a, b, d, e]

if __name__ == "__main__":
    from itertools import permutations
    assert all(sort5(p) == sorted(p) for p in permutations(range(5)))
```

Your task. Revise `00largest.py` to improve the time complexity of `largest(N, k)` to $O(n)$.

Hints:

- Below shows the index m of the lower median of n numbers in ascending order, and the value k such that it is the k-th largest number.

n	Indexes	m	k
1	$[0]$	0	1
2	$[0, 1]$	0	2
3	$[0, 1, 2]$	1	2
4	$[0, 1, 2, 3]$	1	3
5	$[0, 1, 2, 3, 4]$	2	3
6	$[0, 1, 2, 3, 4, 5]$	2	4

  For a positive integer n, what are m and k?

- If `N` has less than 5 numbers, `sorted(N)` returns `N` sorted in ascending order but does not modify `N` (while `N.sort()` does not return any value but modify `N` in ascending order.) You can simply select the median as number a. What is the median's index in `sorted(N)`?

- If `N` has more than 5 numbers, then you can form one or more groups of 5 numbers. For each group `N[i:i+5]` where $i = 0, 5, 10, \ldots$, you can use `sort5(N[i:i+5])[2]` to sort it and obtain its median.

- You can use a recursive call on `largest(medians, k1)` to find the median of the medians. But, what is the value of `k1`?

2. Linear time selection algorithm with duplicates

If the number list `N` contains duplicates, you can assign a unique ID to each number such that we can compare the duplicates using the IDs. One easy way to assign a unique ID is to use the index of the numbers in `N`.

The number of comparisons will be doubled as we need to compare both the numbers and IDs, but the time complexity of the algorithm is still $O(n)$.

Your task. Revise your solution to allow duplicates in `N`. You may use the following class, where `__str__(self)` returns a string representation of the object such that we can print the object directly:

```
from functools import total_ordering

@total_ordering
class Number:
    def __init__(self, number, id):
        self.number = number
        self.id = id

    def __eq__(self, other):
        return self.number == other.number and self.id == other.id

    def __lt__(self, other):
        if self.number != other.number:
            return self.number < other.number
        else:
            return self.id < other.id

    def __str__(self):
        return str(self.number)
```