empirical observations (based on COMPARE calls):

n= 1000, k=10: maximum= 1121, avg= 1109.42

n= 1000, k=20: maximum= 1237, avg= 1219.51

n= 1000, k=30: maximum= 1349, avg= 1329.09

n= 1000, k=40: maximum= 1459, avg= 1435.88

n= 1000, k=50: maximum= 1569, avg= 1541.05

n= 1000, k=60: maximum= 1680, avg= 1650.13

n= 1000, k=70: maximum= 1787, avg= 1759.81

n= 1000, k=80: maximum= 1896, avg= 1866.32

n= 1000, k=90: maximum= 2005, avg= 1973.10

n= 1000, k=100: maximum= 2113, avg= 2078.80

n=10000, k=40: maximum= 10624, avg=10588.34

n=10000, k=50: maximum= 10773, avg=10737.50

n=10000, k=60: maximum= 10926, avg=10889.59

When n=1000, $\frac{\Delta \text{avg}}{\Delta k} \approx 13$. When n=10000, $\frac{\Delta \text{avg}}{\Delta k} \approx 18$.

So, the upper bound of my algorithm cannot be O(n+klogk)

Based on the empirical observations, my algorithm has an upper bound of O(n+klogn)

theoretical predictions:

The data structure I used in my algorithm:

a modified version of Fibonacci Heap.

The steps of my algorithm are:

1. Insert every index [1,n] to the Fibonacci Heap but do not merge → 0 comparison
2. Merge the nodes in the Fibonacci Heap → n-1 comparison
3. Get the max node and report its value → 0 comparison
4. Pop the max node from the heap, then rebalance the heap → analyze later
5. repeat step 3 for k times and step 4 for k-1 times because we don't need to rebalance the heap after we get k largest values. →analyze later

In step 2, my algorithm call COMPARE exactly n-1 times because "lose" node won't be compared twice in merge phase of the Fibonacci Heap. We can get a recurrence relation from this:

$\begin{cases} C(n) = C(n-1) + 1 & (C > 2) \\ C(2) = 1 & (C = 2) \end{cases}$, which C stands for the number of comparison. It's easy to see that C(n)=n-1.

In step 4, the number of trees in my Fibonacci Heap is exactly the Hamming weight of n in binary representation. Let CountSetBits(n) denote the number and IfSet(n,i) denote if the ith bit is set in binary representation of n. (True=1, False=0)

We can get the average number of comparisons = $\frac{\sum_{i=0}^{\log n-1} i*IfSet(n,i)}{CountSetBits(n)} + CountSetBits(n) - 2$

Because after removing the max node, we need to link all its children to the heap, which gives us extra $\frac{\sum_{i=0}^{\log n-1} i*IfSet(n,i)}{CountSetBits(n)}$ trees in average, since we already have $CountSetBits(n) - 1$ trees in the heap.

There are $\frac{\sum_{i=0}^{\log n-1} i*IfSet(n,i)}{CountSetBits(n)} + CountSetBits(n) - 1$ trees in the heap and we need $\frac{\sum_{i=0}^{\log n-1} i*IfSet(n,i)}{CountSetBits(n)} + CountSetBits(n) - 2$ comparisons to merge them and find the max node.

As for the worst case, let LeftmostSetBit(n) denote the index of the last set bit in binary representation of n. And The number of comparisons in the worst case = $LeftmostSetBit(n) + CountSetBits(n) - 2$

in step 5, we repeat step 3 for k times and step 4 for k-1 times.

So, the theoretical AVG (n=10000, k=40)

$$= n - 1 + \sum_{i=0}^{k-1} \left(\frac{\sum_{j=0}^{\log(n-i)} j * IfSet(n-i,j)}{CountSetBits(n-i)} + CountSetBits(n-i) - 2 \right) = 10425.463$$

theoretical WC = theoretical expected WC (n=10000, k=40)

$$= n - 1 + \sum_{i=0}^{k-1} (LeftmostSetBit(n) + CountSetBits(n) - 2)$$

$$= 10745$$

In addition, after the first merge, we use a cache to store the result of COMPARE calls to prevent comparing the same pair of elements again in the future. This can decrease the number of COMPARE calls, but it doesn't decrease the number of element comparisons. Thus, AVG and WC number of element comparisons remain the same.

Observed WC and AVG (based on COMPARE calls):

n=10000, k=40: maximum= 10624, avg=10588.34

Since the samples are randomly generated, the worst sample may not be the real worst case. Thus, the observed WC is smaller than theoretical WC. the observed AVG is larger than theoretical AVG, this is because the observed AVG is a biased estimate of theoretical AVG. The seed of the random number generator determines the sample and bias the result. The sample size is another cause of this phenomenon, 1000 is relatively small. If we increase the size, the observed AVG will be more precise.