CSE 2320 Lab Assignment 4

Due November 8, 2018

Goals:

1. Understanding of radix and counting sorts.

2. Understanding of selection.

Requirements:

1. Write a C program to compute the *k*th smallest number in a sequence of *n* integers in the range 0 . . . 999,999,999. The input will consist of:

   a. *n* and *k*. $1 \le k \le n \le 10{,}000{,}000$
   b. *n* integers in the indicated range, one per line. Duplicate inputs are possible.

   The input will be read from standard input as either keyboard typing or as a shell redirect (<) from a file. Do NOT prompt for a file name!

   Your program will apply counting sort up to 9 times, once for each of the digit positions. The first sort will operate on the "hundred millions" digits. The last sort will operate on the "ones" digits. (This is similar to an MSD radix sort.) Each of the sorts may eliminate a significant fraction of the remaining values. This may also decrease *n* and *k* for the next counting sort.

   Your program should indicate the number of remaining values after each counting sort.

2. Submit your C program on Blackboard by 10:45 am (section 004) or 1:45 pm (section 003) on November 8. One of the comment lines should indicate the compilation command used on OMEGA.

Getting Started:

1. Before writing your counting sort, write a function to reliably isolate a specific digit position from an `int`.

2. Suppose *n* is 4,000,001 and *k* is 2,000,001 (i.e. the median is needed). After running the second phase (see Notes 8.E) of the counting sort for the "hundred millions" digits, suppose the count table contains:

```
0  300,000
1  400,000
2  400,000
3  400,000
4  400,000
5  400,000
6  400,000
7  400,000
8  400,000
9  500,001
```

The third phase of this counting sort then gives the following starting positions:

```
0           0
1     300,000
2     700,000
3   1,100,000
4   1,500,000
5   1,900,000
6   2,300,000
7   2,700,000
8   3,100,000
9   3,500,000
```

Since the *k*th smallest number would be stored at array position *k* - 1 (2,000,000) after all digit positions have been processed, only the 400,000 inputs in the range 500,000,000 . . . 599,999,999 are kept in the fourth phase for the next counting sort. In addition, *k* is reduced to 100,001.

After running the second phase of the counting sort for the "ten millions" digits, suppose the count table contains:

```
0   30,000
1   40,000
2   40,000
3   40,000
4   40,000
5   40,000
6   40,000
7   40,000
8   40,000
9   50,000
```

The third phase of this counting sort gives the following starting positions:

```
0         0
1    30,000
2    70,000
3   110,000
4   150,000
5   190,000
6   230,000
7   270,000
8   310,000
9   350,000
```

Since the *k*th smallest number would be stored at array position *k* - 1 (100,000) after all digit positions have been processed, only the 40,000 inputs in the range 520,000,000 . . . 529,999,999 are kept in the fourth phase for the next counting sort. In addition, *k* is reduced to 30,001.

3. If *n* is reduced to 1, no further phases are needed. Duplicate input values may prevent early termination.

4. Errors are most likely to occur in the last few sorts when some of the counts are zero after the second phase.

5. It is easy to adapt the program `http://reptar.uta.edu/NOTES2320/selection.c` to compare your results against. This program demonstrates selection by partitioning.