Oscar Bautista

CMSC351-0301

Smallest number of iterations – 1

Largest number of iterations – 8

Most popular repeat – 6174

My program could not be used to determine the answer if the input were five or six digits long because four digits are hard coded into it. However, a lot of my work is based on the number of digits for sorting and conversions. Because the input size grows much faster than the number of digits, this algorithm runs in n (n = input size) time, which makes it reasonable for larger input sizes were it built for it.

```
#include <stdio.h>

static int array_to_int(int input[]);
static void int_to_array(int result[], int input);
static void ascending_sort(int result[], int input[]);
static void descending_sort(int result[], int input[]);
static int get_smallest(int list[]);
static int get_largest(int list[]);
static int get_largest_index(int list[]);



int main () {
  //Integer arrays for sorting the digits
  int n1[4] = {0};
  int n2[4] = {0};
  int new_value[4] = {0};

  //The integers themselves for math operations and array indexing
  int n1_int = 0;
  int n2_int = 0;
  int value_int = 0;
  int new_value_int = 0;
  int i = 0;

  //The current run's already obtained newValues
  int current[10000] = {0};
  //The number of iterations in the current run
  int current_iter = 0;
  //The number of times a number was the final duplicate
  //i.t final_duplicate[1000] is the number of times 1000 was the final dup
  int final_duplicate[10000] = {0};
  //The number of iterations for each starting value
  //i.e final_iter[5] is the number of iterations it took for the value 5
```

```
  int final_iter[10000] = {0};

  for (value_int = 0; value_int < 10000; value_int++) {
    current_iter = 0;
    for (i = 0; i < 10000; i++) {
      current[i] = 0;
    }
    new_value_int = value_int;
    while (current[new_value_int] != 1) {
      current_iter++;
      current[new_value_int] = 1;
      int_to_array(new_value, new_value_int);
      descending_sort(n1, new_value);
      n1_int = array_to_int(n1);
      ascending_sort(n2, new_value);
      n2_int = array_to_int(n2);
      new_value_int = n1_int-n2_int;
    }
    final_duplicate[new_value_int]++;
    final_iter[value_int] = current_iter;
  }

  printf("The smallest number of iterations was %d.\nThe largest number of "
        "iterations was %d.\nThe most popular repeat was %04d.\n",
        get_smallest(final_iter), get_largest(final_iter),
        get_largest_index(final_duplicate));


}

int array_to_int(int* arr) {
  int return_value = arr[0] * 1000 + arr[1] * 100 + arr[2] * 10 + arr[3];
  return return_value;
```

```
}

void int_to_array(int* return_value, int n) {
  return_value[0] = n / 1000;
  return_value[1] = (n / 100) % 10;
  return_value[2] = (n / 10) % 10;
  return_value[3] = n % 10;
}

void ascending_sort (int* return_value, int* n) {
  int i = 0;
  int j = 0;
  int hold;
  int sorted = 0;

  for (i = 0; i < 4; i++)
    return_value[i] = n[i];

  while (!sorted) {
    sorted = 1;
    for (i = 1; i < 4; i++) {
      if (return_value[i-1] > return_value[i]) {
        hold = return_value[i-1];
        return_value[i-1] = return_value[i];
        return_value[i] = hold;
        sorted = 0;
      }
    }
  }
}

void descending_sort (int* return_value, int* n) {
  int i = 0;
```

```
  int j = 0;
  int hold;
  int sorted = 0;

  for (i = 0; i < 4; i++)
    return_value[i] = n[i];

  while (!sorted) {
    sorted = 1;
    for (i = 1; i < 4; i++) {
      if (return_value[i-1] < return_value[i]) {
        hold = return_value[i-1];
        return_value[i-1] = return_value[i];
        return_value[i] = hold;
        sorted = 0;
      }
    }
  }
}

int get_smallest (int* list) {
  int i;
  int min = 10000;

  for (i = 0; i < 10000; i++) {
    if (list[i] < min)
      min = list[i];
  }

  return min;
}

int get_largest (int* list) {
```

```
  int i;
  int max = 0;

  for (i = 0; i < 10000; i++) {
    if (list[i] > max)
      max = list[i];
  }

  return max;
}

int get_largest_index (int* list) {
  int i;
  int max = 0;
  int return_value = 0;

  for (i = 0; i < 10000; i++) {
    if (list[i] > max) {
      max = list[i];
      return_value = i;
    }
  }

  return return_value;
}
```