CSCI 3333 Homework: Bounded-Universe Sorting (with Solutions)

1 Algorithm Analysis

A sorting algorithm is stable provided that for any pair of elements $A[i]$, $A[j]$ in the input array $A$ with $A[i] == A[j]$ and $i < j$, these elements are in the sorted array at indices $i'$, $j'$ with $i' < j'$.

Problem 1. Is the following (insertion-sort-based) sorting algorithm stable?

```c
void sort(int* A, int n)
{
    for (int i = 0; i < n; ++i)
        for (int j = i-1; j >= 0; --j)
            if (A[j] >= A[j+1])
                {
                    int tmp = A[j];
                    A[j] = A[j+1];
                    A[j+1] = tmp;
                }
}
```

If it is not stable, how could it be modified to be stable?


Problem 2. Analyze the running time (on inputs of length $n$) of a radix sorting algorithm where the input is given in binary (i.e., each “digit” is a a bit with value 0 or 1) and comes from the universe $\{0,1,\ldots,2^p-1\}$.

Solution 2. Each counting sort invocation takes $O(n)$ time. Since input from the universe is represented using $p$ bits, counting sort is run $p$ times. So the total time spent is $O(n) \cdot p = O(np)$ time.

2 Algorithm Implementation

Problem 3. Implement a C++ function that uses counting sort to sort an array $A$ (of length $n$) consisting of Entry objects according to their rating values (guaranteed to be between 0 and 9, inclusive):
class Entry
{
    public:
        string hashtag;
        int rating;
};

Solution 3.

void sort(Entry* A, int n)
{
    int H[10];
    for (int i = 0; i < 10; ++i)
        H[i] = 0;

    for (int i = 0; i < n; ++i)
        ++H[A[i].rating];

    int offsets[10];
    offsets[0] = 0;
    for (int i = 1; i < 10; ++i)
        offsets[i] = offsets[i-1] + H[i-1];

    Entry* B = new Entry[n];
    for (int i = 0; i < n; ++i)
    {
        B[offsets[A[i].rating]] = A[i];
        ++offsets[A[i].rating];
    }

    for (int i = 0; i < n; ++i)
        A[i] = B[i];

    delete[] B;
}

3 Algorithm Design

A sequence is binary if it consists of only 0s and 1s.

Problem 4. Give an $O(n)$-time algorithm that sorts a binary array $A$ (of length $n$) and only modifies elements via the swap function.

Solution 4. Use two variables to store variables $l$ and $r$, initialized to indices 0 and $A$.size()-1. Increment $l$ until a 1-valued element is found. Similarly, decrement $r$ until
a 0-valued element is found. Swap the values at these two locations. Repeat these three steps (increment l, decrement r, swap) until l == r.

Running time: The algorithm visits each element once. Each visit takes O(1) time: incrementing/decrementing l or r, comparing l to r, comparing A[l] to A[r], and possibly swapping A[l] and A[r]. So O(1) · n = O(n) total time is used.

Correctness: The algorithm repeatedly swaps the most “extreme” (l farthest left, r farthest right) pair of elements in the wrong locations (a 1-valued element to the left of a 0-valued element). It repeats this until no such pair exists, i.e. every 0-valued element is to the left of every 1-valued element. Any binary array with every 0-valued element is to the left of every 1-valued element is sorted.