Algorithm Lab

Week 2: Non-Comparison Sort

Description

Classical sorting algorithms are algorithms rearrange the input list to a certain order defined by the comparison function. For some special domain (in most case, subdomain of natural numbers), we can design special sorting algorithms that working without generalized comparison functions, respectively, non-comparison sorting algorithms.

Instance: a list of non-negative integers $A = (a_1, a_2, \dots, a_n)$ *that for all* $1 \leq i \leq n, 0 \leq i < r^d$ *where* r *is radix and* d *is the position of most significant digit.*

Result: a list $B = (b_1, b_2, \dots, b_n)$ *that* $\{a_1, a_2, \dots, a_n\} = \{b_1, b_2, \dots, b_n\}$ *and for all* $1 < i \leq n, b_{i-1} < b_i$.

Algorithm Design

1. Set processing position to least significant order.
2. Rearrange the list by digit at processing position.
3. Set processing position to next digit.
4. Repeat step 2, 3 until finished the most significant digit rearrangement.

Implementation

Language: C

```
void sort_by_digit(int *A, int n, int r, int p)
{
        int base = 1;
        while (p > 0)
        {
                --p;
                base *= r;
        }
        int count[r], B[n], m = 0;
        for (int j = 0; j < r; ++j)
                count[j] = 0;
        for (int i = 0; i < n; ++i)
                count[(A[i] / base) % r]++;
        for (int j = 0, psum = 0, sum = 0; j < r; ++j)
        {
                psum = sum;
                sum += count[j];
                count[j] = psum;
        }
        for (int i = 0; i < n; ++i)
                B[count[(A[i] / base) % r]++] = A[i];
        for (int i = 0; i < n; ++i)
                A[i] = B[i];
}

void radix_sort(int *A, int n, int r, int d)
{
        for (int i = 0; i < d; ++i)
                sort_by_digit(A, n, r, i);
}
```

Questions

- Function `sort_by_digit` is a stable counting sort. If we rewrite the function to an unstable version, can radix sort still work correctly?
- This radix implement is processing through least significant digit to most significant digit. Can't you design radix algorithm which start at most significant digit?
- Please design an algorithm to measure minimum usable d of list A and radix r.
- Please analysis the space complexity and the time complexity. (should be functions with arguments n, r, d)