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 = (𝑎1, 𝑎2, … , 𝑎𝑛) *that*

𝑓𝑜𝑟 𝑎𝑙𝑙 1 ≤ 𝑖 ≤ 𝑛, 0 ≤ 𝑖 < 𝑟𝑑 *where* 𝑟 *is radix and* 𝑑 *is the position of most significant digit.*

*Result: a list* 𝐵 = (𝑏1, 𝑏2, … , 𝑏𝑛) *that* {𝑎1, 𝑎2, … , 𝑎𝑛} = {𝑏1, 𝑏2, … , 𝑏𝑛} *and*

𝑓𝑜𝑟 𝑎𝑙𝑙 1 < 𝑖 ≤ 𝑛, 𝑏𝑖−1 < 𝑏𝑖*.*

# 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 is a stable counting sort. If we rewrite the function to an unstable

sort\_by\_digit

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)

# Answer

1. It is very important that radix sort use a stable sort for sorting on the digit values in each position. This is because once an element has been assigned a place according to the digit value in a less significant position, its place must not change unless sorting on one of the more significant digits requires it

2. MSD Radix Sort

-Partition array into R pieces according to first character (use key-indexed counting)

-Recursively sort all strings that start with each character (key-indexed counts delineate/separates subarrays to sort)

NB: When a bucket contains only 1 element, it is already sorted and thus does not need to recurse on that bucket.

3.

* find the maximum a of A
* d<-0
* d<-d+1
* repeat step 3 until r^d > max(A)

r^d` = max(A)

d` = log(max(A))/log(r)

d > d`

d = floor(d`) + 1

4. Best case time complexity: O(d\*(n+r))

# Worst case time complexity: O(d\*(n+r))