

Ph.D. Qualifying Exam (Spring 2008)
Algorithms

Question 1.

We want to show by induction that radix sort works.

Given that A is an array of d -digit numbers, where digit 1 is the least significant digit and digit d is the highest significant digit. Radix sort performs the following:

```
For i <- 1 to d
  do use a stable sort to sort array A on digit i
```

We are going to prove by induction on j that the program

```
For i <- 1 to j
  do use a stable sort to sort array A on digit i
```

returns an array that is the same as the one obtained when we use a stable sort to sort array A on the tuple [digit j , digit $j - 1 \dots$ digit 1].

The statement is trivially true for the base case when $j = 1$.

Assume that the statement is true for $j = k$ where $k \geq 1$. Consider the statement when $j = k + 1$.

The program

```
For i <- 1 to k+1
  do use a stable sort to sort array A on digit i
```

is equivalent to

```
For i <- 1 to k
  do use a stable sort to sort array A on digit i;
use a stable sort to sort array A on digit k+1
```

By the induction hypothesis,

```
For i <- 1 to k
  do use a stable sort to sort array A on digit i
```

is equivalent to sorting the array A using a stable sort on the tuple [digit k , digit $k - 1 \dots$ digit 1]. Thus,

```
For i <- 1 to k+1
  do use a stable sort to sort array A on digit i
```

is equivalent to

```
use a stable sort to sort array A
  on [ digit k , digit k-1 , ..., digit 1 ];
use a stable sort to sort array A on digit k+1
```

We want to argue that the following program (which we called program P)

```
use a stable sort to sort array A
  on [ digit k , digit k-1 , ..., digit 1 ];
use a stable sort to sort array A on digit k+1
```

is equivalent to use a stable sort to sort array A on [digit $k + 1$, digit $k \dots$ digit 1].

Given two array elements a and b of A such that after A is sorted using a stable sort on [digit $k + 1$, digit $k \dots$ digit 1], a is smaller (that is, appear earlier in the output) than b , there are three cases (see below) to consider. We want to show that program P will output a before b .

Notation: we write $a[p]$ to denote the p -digit of a and $a[p..q]$ to denote the tuple [digit p , digit $p - 1 \dots$ digit q] of a .

Case 1. $a[k + 1] < b[k + 1]$.

The second line of program P sorts the sorting output of the first line according to digit $k + 1$. By the assumption that $a[k + 1] < b[k + 1]$, a must appear before b after the execution of the second line of program P .

Case 2. $a[k+1] = b[k+1]$ and $a[k..1] < b[k..1]$.

Since $a[k..1] < b[k..1]$, a appears before b after the execution of the first line of program P . Next, applying the sorting in the second line of program P will not change the order of appearances of a and b because $a[k+1] = b[k+1]$ and the sorting is stable.

Case 3. $a[k+1] = b[k+1]$ and $a[k..1] = b[k..1]$.

Given that $a[k+1] = b[k+1]$ and $a[k..1] = b[k..1]$ and the fact that a appears before b after A is sorted using a stable sort on [digit $k+1$, digit $k \dots$ digit 1], we conclude that a appears before b in the original array A .

Since $a[k..1] = b[k..1]$ and a appears before b in the original array A , a would appear before b after the execution of the first line of program P because the sorting is stable. Finally, applying the sorting in the second line of program P will not change the order of appearances of a and b because $a[k+1] = b[k+1]$ and the sorting is stable.

Question 2.

We need to re-do the analysis given in page 175-176 of the textbook.

$$\begin{aligned}
 & E[n_i^2] \\
 &= \sum_{j=1}^n E[X_{ij}^2] + \sum_{1 \leq j \leq n} \sum_{1 \leq k \leq n \text{ and } k \neq j} E[X_{ij} X_{ik}] \\
 &= \sum_{j=1}^n \frac{2}{n} + \sum_{1 \leq j \leq n} \sum_{1 \leq k \leq n \text{ and } k \neq j} \left(\frac{2}{n}\right)^2 \\
 &= 2 + (n^2 - n) \left(\frac{2}{n}\right)^2 \\
 &= 2 + 4 - \frac{4}{n} \\
 &= 6 - \frac{4}{n}
 \end{aligned}$$

$$E(T(n)) = \Theta(n) + \sum_{i=0}^{n/2-1} O(E[n_i^2]) = \Theta(n) + O(3n - 2)$$

On the other hand, the traditional bucket sort with n buckets runs in expected time $E(T(n)) = \Theta(n) + O(2n - 1)$.

Therefore, the modified bucket sorts could take 50% more time than the traditional bucket sort assuming that the time to distribute the n data into the buckets are insignificant relative to the total time needed to perform the insertion sort.

Question 3.

Let $p(i, j)$ = length of the longest palindromic subsequence in $x[i, i+1, \dots, j]$, where $1 \leq i \leq j \leq n$.

Base case: $p(i, i) = 1$

In addition, we define $p(i, i - 1) = 0$.

Recursive formula: $p(i, j) = p(i - 1, j - 1) + 2$ if $x[i] = x[j]$ and $p(i, j) = \max\{p(i + 1, j), p(i, j - 1)\}$ otherwise.

We compute $p(i, j)$ in increasing order according to the value of $j - i$.

$p(1, n)$ gives the length of the longest palindromic subsequence.

There are $O(n^2)$ subproblem instances of $p(i, j)$ where $1 \leq i \leq j \leq n$. As each subproblem instance can be computed in $O(1)$ time, the total time is $O(n^2)$.

The above answer computes the length of the longest palindromic subsequence. In order to return the longest palindromic subsequence, we need to remember the decisions in computing $p(i, j)$. There are three possible decisions: decision (1) is when $x[i] = x[j]$, decision (2) is when $x[i] \neq x[j]$ and $p(i + 1, j) > p(i, j - 1)$, and decision (3) is when $x[i] \neq x[j]$ and $p(i + 1, j) \leq p(i, j - 1)$. Knowing the decisions, we can re-trace the longest palindromic subsequence easily in time proportional to the length of the subsequence.