CS1020E: Data Structures and Algorithms I

Tutorial 9 – Complexity and Sorting

(Week starting 30 March 2015)

Q1. QuickSort with tri-partitioning

In standard Quicksort, we partition an array A[1..n] about a pivot element x. Prof Triple suggests that we might be able to do better if we use two pivot elements, instead of just one.

a) Tri-partition
Suppose that you choose two first elements `x` and `y` (such that `x <=y`) to partition the array `A[1..n]` into 3 partitions: the *left* partition contains all element <= `x`, the *middle* partition contains all elements >`x` but <=*y*, and the *right* partition contains all elements that >`y`. Design and implement an algorithm to perform this tri-partitioning with two pivots.

```
pair<int, int> tripartition(int A[], int low, int high){
      int x = A[low];
      int y = A[low+1];
      //swap x,y if needed.

      // Do tri-partitioning, such that:
      // each element in A[low .. px]  <= x;
      // x< each element in A[px+1 .. py] <= y;
      // each element in A[py+1 .. high-1]  >y;

      return make_pair(px, py);
}
```

What is the running time and space complexity?

b) Implement quick sort algorithm with tri-partition procedure in part a.

c) What is worst case and best case running time of your algorithm in part b.
(This question was adapted from CS3230 midterm)

Q2. Radix Sort

Given a list of N people with their first name and age, sort the list in an increasing order of age. If two people have same age, sort their name in lexicographic order.

a) Write an algorithm using vector of pairs and standard sorting routine (that is, sort() function in STL <algorithm>).

b) Design and implement a radix sort algorithm.
For both questions (any algorithm design question), analyze the time and space complexities.

2. Algorithm Ordering

Rearrange the following algorithms in increasing order of their Big O complexity. If there are multiple algorithms with the same complexity, group them together. Assume `foo()` has $O(1)$ complexity, the Big

O is determined only by how many times `foo()` is called (i.e. assuming `foo()` is the most relevant part of the algorithm).

i) Algorithm 1:

```
for(int i=0; i<n; i++)
  for(int j=n; j>=i; j--)
    foo();
```

ii) Algorithm 2:

```
for(int i=0; i<n; i++)
  for(int j=i+1; j>i; j--)
    foo();
```

iii) Algorithm 3:

```
for(int i=1; i<n; i*=2)
  for(int j=0; j<i; j++)
    foo();
```

iv) Algorithm 4:

```
for(int i=0; i<n; i+=2)
  for(int j=i+1; j<n; j*=2)
    foo();
```

v) Algorithm 5:

```
for(int i=1; i<n; i++)
  for(int j=0; j<i; j++)
    if(j%i == 0) foo();
```

vi) Algorithm 6:

```
for(int i=0; i<n; i++)
  for(int j=0; j<n; j++)
    if(j%2 == 0) for(int k=i; k<n; k++) foo();
    else for(int l=0; l<i; l++) foo();
```

vii) Algorithm 7:

```
for(int i=0; i>>n==0; i++)
  foo();
```

4. Algorithm Design

Given an array of `integer A` with size `n` and an `integer` value `k`, we need to rotate the array `A` to the right `k` times using only a constant number of temporary `integer` variables not related to either the size of the array `A`, or the value `k`. There are several ways to solve this, and the naïve way is to store the front most value to a temporary `integer temp`, and shift the remaining values in the array.

Example: `k=6, arr=[1,2,3,4,5,6,7,8]`
`res=[3,4,5,6,7,8,1,2]`

a) Write the algorithm for the naïve rotation method. What is the time-complexity of this algorithm in terms of `n` and `k`?

b) Develop another algorithm that would perform better than the naïve algorithm in terms of Big O notation *while still maintaining the constraint of constant number of temporary variables*. What is the time-complexity of this algorithm in terms of `n` and `k`?

```
void rotate(int arr[], int n, int k) {
  // Temporary Variables: only int is allowed, no Array

  // Your algorithm here...

}
```

Extra questions:

Solve extra questions for fun and challenge!

E1. Preprocess

These two problems can be solved more efficiently once you do sorting on the input. For each of the problem, present two solutions (and the time-complexity) to it: one where you don't use sorting and another where you use sorting.

a) Recurrence Problem: Given an array A of random integers, print all the integers that appear at least k-times.

b) Set Data Structure: Given two input arrays A and B of random integers without duplicates (this is a Set, Set will have to be unique), find the unique union or intersection (choose one, the algorithms aren't exactly that different) of the two sets.

E2. Classic N-queen problem

Place N queens into a chess board of size N, such that no two queens threaten each other. Given N, count the number of ways of placing.

Example N = 12: There are 14200 ways.

E3. Classic Knight's tour problem

Find a sequence of moves of a knight on a chessboard (of size N*N) such that the knight visits every square only once.

Input: N
x,y - start position of the Knight.
Output:
A chess-board with sequence of knight's moves