Prelim Solutions CS 410, Summer 2000

1. (30 points: 5 points each) Short answer questions.

   (a) *Which is correct:*

   i. $2^{n!} = o((n!)^2)$
   ii. $2^{n!} = \Theta((n!)^2)$
   iii. $2^{n!} = \omega((n!)^2)$
   iv. None of the above, $2^{n!}$ *and* $(n!)^2$ *are incomparable.*

   Substantiate your answer.
   Answer iii is correct. If we let $x = n!$, we are trying to compare 2^x with x^2, and clearly 2^x is asymptotically larger.

   (b) *Consider the following program.*

```
DivideAndConquer (A)
 1 Divide (A,B,C)        // divide A into two equal sized pieces called B and C
 2 DivideAndConquer (B)
 3 DivideAndConquer (C)
 4 Combine (A,B,C)       // combine B and C into the processed A
```

   Let n be the number of elements in the input array A.
   Step 1 (Divide) takes time $\Theta(n)$, and step 4 (Combine) takes time $\Theta(n \lg n)$.
   What is the recurrence that describes the time used by this program?
   $T(n) = 2T(n/2) + \Theta(n \lg n)$

   (c) *Recall the modified bubble sort presented in homework 1 (using an* any_swaps *flag). Although bubble sort generally takes time $\Theta(n^2)$, with this modification it takes linear time for some inputs. For example, any length input which is already in sorted order will be processed in linear time by bubble sort.*

   Which of the following sorting algorithms similarly take linear time for some inputs? Circle your answer(s).

   Note: *Do not consider "inputs which are small enough"; given any size n, the algorithm should take linear time for some inputs.*

   insertion sort, selection sort, merge sort, quicksort, bucket sort.

   insertion sort and bucket sort are linear on some or all inputs. The others are not.

   (d) *Illustrate the operation of Radix Sort on the array* $A = \langle\ 300,\ 205,\ 115,\ 100,\ 310\ \rangle$.
   Show each step/change as you did in your homework assignment.

300	300	300	100
205	100	100	115
115	310	205	205
100	205	310	300
310	115	115	310

   (e) *The following statement is found in a stable sort:*

```
if data[i].key < data[j].key
   swap data[i] and data[j]
```

Tell whether each of the following modifications will also result in a stable sort:

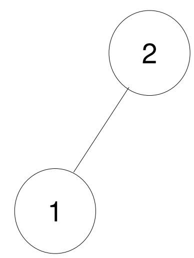
i.
```
if data[i].key < data[j].key
   swap data[j] and data[i]
```
ii.
```
if data[i].key <= data[j].key
   swap data[i] and data[j]
```
iii.
```
if data[i].key < data[j].key
   swap data[i].key and data[j].key
```

i. is stable, ii. is not stable, and iii. doesn't even sort! So the answer is i.

(f) *Define the load factor α which is used to analyze hashing methods.*
Be sure to define your notation.
$\alpha = n/m$, where n = number of items in the hash table, m = number of slots in the table.

2. (a) *Give an example of a heap which is not a complete binary tree.*

Consider a leaf node with key 2, and a left child with key 1. This is a heap, but not complete.



(b) *What is the precise maximum number of elements which may be contained in a heap of height h? Do not use asymptotic notation.*
$2^h - 1$

(c) *You are given a heap of n elements.*
True or False: To determine if an element x is present in the heap takes $O(\lg n)$ time.
Give a 1 or 2 line justification of your answer.

False. In a heap we do not have any ordering information other than the parents are larger then either child. This does not enable us to eliminate one child's subtree as containing the key x.

3. (10 points: 5,3,2) *Consider the array*

```
[ 8 7 6 5 4 3 2 1 ]
```

Recall the quicksort *algorithm discussed in class and in CLR.*

(a) *Show where the partition will be placed at the end of the (first) call to* Partition. *You should also indicate where each element is located at the end of* Partition *(i.e., indicate what the array looks like after* Partition *is run.*

1 7 6 5 4 3 2 | 8

(b) *Show where the partition will be placed when* Partition *is run recursively on the subarrays formed in part (a). Again, show the contents of the subarrays formed as well.*

1 | 7 6 5 4 3 2

(c) *One last time, show where the partition will be placed when* Partition *is run recursively on the subarrays formed in part (b). Again, show the contents of the subarrays formed as well.*

2 6 5 4 3 | 7

4. (20 points) *Recall Stooge-Sort from homework assignment 3:*

```
Stooge-Sort (A,i,j)
if A[i] > A[j]
   swap A[i] and A[j]
if i+1 >= j
   return
k = floor((j-i+1)/3)       // round down
Stooge-Sort (A,i,j-k)      // First two-thirds
Stooge-Sort (A,i+k,j)      // Last two-thirds
Stooge-Sort (A,i,j-k)      // First two-thirds again
```

Consider the Curly-Larry-Mo (CLM) version of Stooge sort:

```
CLM-Sort (A,i,j)
if i == j
   return
if i+1 == j
   if A[i] > A[j]
      swap A[i] and A[j]
   return
k = floor((j-i+1)/3)       // round down
CLM-Sort (A,i,j-k)         // First two-thirds
CLM-Sort (A,i+k,j)         // Last two-thirds
CLM-Sort (A,i,j-k)         // First two-thirds again
```

Does CLM-Sort(A,1,n) also correctly sort input array A of length n?
Prove your answer. Do not rely on the correctness of Stooge-Sort in your proof.

Your answer should essentially be the same as for homework 3.

(base case) It is easy to see that it sorts correctly when the length of input is 1 and 2.

Induction Hypothesis (IH): CLM-sort sorts correctly for input of any size between 1 and $n-1$.

(induction step) Consider input of size $n : A = (a_1, a_2, ...a_n)$.

Consider line 8 (CLM-sort(A,1,2n/3)). Since $2n/3 < n$, by induction hypothesis the $2n/3$ numbers $B = (a_1, a_2, ..., a_{2n/3})$ are sorted after the call to CLM-sort. Now the first one-half elements of B (the first one-third elements of A) can never be a part of the last one-third elements in the sorted order of A, since we know that the last one-half elements of B are all larger than these. Therefore the second call to CLM-sort on line 9 involving the last one-half elements of sorted array B and the last one-third of A fixes the last one-third part of A correctly. Finally, the third call to CLM-sort on line 10 fixes the first two-thirds.
Hence CLM-sort correctly sorts the input data.

5. (25 points: 5,15,5) Hash functions.

   (a) *What formula is used in the* division method *for creating hash functions?*

   $h(k) = k \mod m$

   (b) *Recall that the formula used in the* multiplication method *for creating hash functions is*

   $$h(k) = \lfloor m\ (kA \mod 1) \rfloor$$

   The division method and the multiplication method were described as distinct methods, but in fact the multiplication method is more general. Specifically, the division method can be implemented with the multiplication method. Explain carefully how this can be done.

   Suppose we let $A = 1/m$. The division algorithm (from discrete math) tells us $\exists$ integers q, r such that $k = mq + r$, where $0 \le q$ and $0 \le r < m$ (that is, r is the remainder when k is divided by m). So we have

   $$\begin{aligned} \lfloor m(kA \mod 1) \rfloor &= \lfloor m(kA - \lfloor kA \rfloor) \rfloor \\ &= \lfloor m(k/m - \lfloor k/m \rfloor) \rfloor \\ &= \left\lfloor m \left(\frac{mq+r}{m} - \left\lfloor \frac{mq+r}{m} \right\rfloor \right) \right\rfloor \\ &= \lfloor m(q + r/m - \lfloor q + r/m \rfloor) \rfloor \\ &= \lfloor m(q + r/m - q) \rfloor && \text{since } r < m \\ &= \lfloor m(r/m) \rfloor \\ &= r \\ &= k \mod m && \text{by definition of mod} \end{aligned}$$

   (c) *Use the division method to insert the following list of keys into a hash table of size 11. Resolve collisions by chaining.*

12	5	57	89	20	49	9	33	71

0	1	2	3	4	5	6	7	8	9	10
33	12	57			5				20	
	↓				↓				↓	
	89				49				9	
					↓					
					71					