ECE608 Midterm 2 Spring 2007

Please fill in the following information.

THEN PUT YOUR EMAIL ADDRESS ON EVERY PAGE OF THE EXAM !

NAME: **SOULTION**

Email (please put complete address):

Neatness counts. We will not grade what we cannot read.

Exam is worth 100 points. Please phrase your answer succinctly. Write your answer on the same page, on which the question is given. There are five questions and a total of nine pages. Make sure you turn in all these pages.

Do not attempt to look at other students' work. Keep your answers to yourself.
Any sort of cheating will result in a zero grade.

Read and sign the statement below. Wait for instructions to start the examination before continuing to the next page.

"I signify that the work shown in this examination booklet is my own and that I have not received any assistance from other students nor given any assistance to other students."

_____(Signature)

Q1. Binary Search Tree (16 points total)

We can sort a given set of *n* numbers by constructing a binary search tree containing those numbers by using TREE-INSERT on the list of numbers and then performing INORDER-TREE-WALK on the resulting tree. Answer the following questions concerning the running time of this sorting algorithm.

(a) What is the worst-case running time for performing a sort in this manner? Identify at least one situation that would produce this result. Analyze the worst-case running time precisely using either a recurrence or summation, solving it in closed-form using Θ-notation. **(8 points)**

(b) What is the best-case running time for performing a sort in this manner? Identify at least one situation that would produce this result. Analyze the time precisely using either a recurrence or summation, solving it in closed-form using Θ-notation. **(8 points)**

Solution:

(a) The worst-case running time occurs when the list of elements to be inserted in the tree are already sorted in increasing (or decreasing) order. In this case, each element will be inserted into a linear list after being compared to all of the elements that have already been inserted. The first element can be inserted with no comparisons, the second with 1, the third with 2,…, the *i*th with *i* – 1,…, and the *n*th with *n*-1 comparisons; hence, the time to insert the *i*th elements is *i* – 1 + 1 = *I*, giving $\sum_{i=1}^{n} i = \frac{(n+1)n}{2} = \Theta(n^2)$ worst-case time. Note that because the time to perform the inorder tree walk is $\Theta(n)$, the overall worst-case time is $\Theta(n^2)$.

(b) The best case occurs when the order of the numbers in the set results in a balanced binary search tree. In this case, the first element will be compared to no others, the left and right child will have one comparison to the root, their children will have two comparisons, etc. For $i \geq 1$, *i*th element inserted will be compared to $\lfloor \lg i \rfloor$ elements for its insertion, giving a time to insert of $\sum_{i=1}^{n} (\lfloor \lg i \rfloor + 1) \leq \sum_{i=1}^{n} (\lg i + 1) \leq 2\sum_{i=1}^{n} \lg i = 2\lg(\prod_{i=1}^{n} i) = 2\lg(n!) = O(n \lg n)$.

Also, $\sum_{i=1}^{n} (\lfloor \lg i \rfloor + 1) \geq \sum_{i=1}^{n} ((\lg i - 1) + 1) = \sum_{i=1}^{n} \lg i = \lg(n!) = \Omega(n \lg n)$. Note that because of this and the time to perform the inorder tree walk is $\Theta(n)$, the overall best-case time is $\Theta(n \lg n)$,

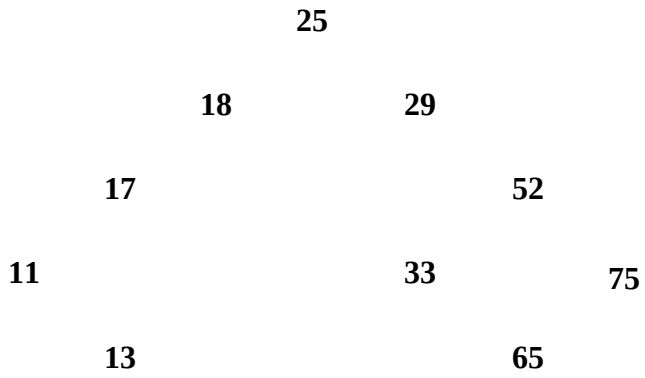
Q2. Binary Search Tree (19 points total)

(a) Draw the binary search tree that results from inserting the following keys into an initially empty tree, in the order given: 25, 18, 17, 29, 52, 75, 65, 33, 11, 13. **(8 points)**

(b) Give two permutations of the above keys (in Part c) that will result in a maximal height binary search tree. Is there a third permutation? If so, give it. If not, explain why not. **(11 points)**

Solution:

(a):



(b):

11 13, 17, 18, 25, 29, 33, 52, 65, 75
75, 65, 52, 33, 29, 25, 18, 17, 13, 11

There are others; simply delete the parent of an only child which is a leaf and then reinsert the deleted node.

11, 13, 17, 18, 25, 29, 33, 52, 65, 75
75, 65, 52, 33, 29, 25, 18, 17, 11, 13

Q3. Hashing (total points 15)
Suppose *n* records are stored in a hash table of size m with *m* > *n* using chaining, and suppose that a good hash function is used so that the probability that a key is hashed into any of the *m* slots is *1/m*

i. For a particular slot in the chained hash table, what is probability that the slot is empty? **(7 points)**

$$(1-\frac{1}{m})^n$$

ii. If open addressing were used instead of chaining, what would the probability be that a certain slot is empty? **(8 points)**

$$\prod_{i=0}^{n-1}(1-\frac{1}{m-i})=\prod_{i=0}^{n-1}(\frac{m-i-1)}{m-i})=\frac{m-n}{m}=1-\frac{n}{m}$$

Q 4. Hashing (20 points total)

Consider a hash table of size 7. For this table, assume that the quadratic probing based on the following rehash function

$$(k+j^2) \bmod 7$$

is used where k is the original hash value of an input key and j = 0, 1,2,3,4,….

(a) (2 points). Write down a four-element input sequence x_1, x_2, x_3, x_4 (where $0 <= x_i <= 6$) such that **no collision** occurs and the hash values map into the pattern of occupied (shaded) cells shown below (indices 1, 2, 4, and 6).

Elements							
Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**

1, 2, 4, 6

(b) (6 points) List one more sequence of four elements x_1, x_2, x_3, x_4 (where $0 <= x_i <= 6$) that maps into the shaded indices of part (a) and is not a permutation of the sequence that you have found in part (a). The total number of collisions in this new sequence should as minimum as possible. Provide the total number of collisions for this sequence.

#collisions = 1

1 4 6 1

(c) (12 points). Consider a table of size 13 where insertion of a key (k) is carried out using a double hashing, $h(k,i) = (h_1(k) + i\ h_2(k))\ mod\ 13$, where $i = 0, 1, 2, \ldots, 12$, with $h_1(k) = k\ mod\ 13$ *and* $h_2(k) = 1 + (k\ mod\ 11)$.

Write an ordered sequence of input keys stored in this table that results in the mapping as shown below. **Give your reasoning for full credit.**

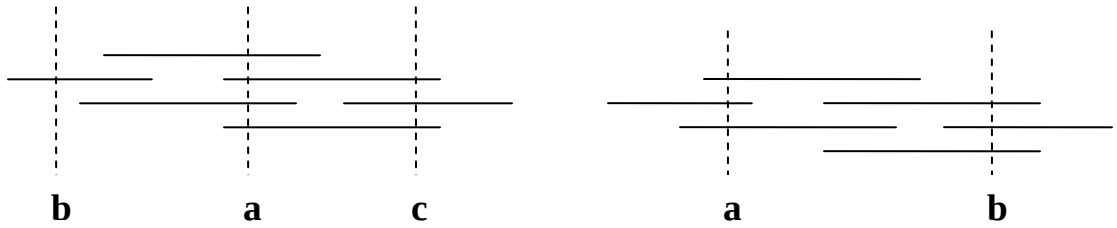
Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Elements		92			82	111		85		27		63	

No sequence exists.
92, 82, 85, and 63 can come in any order. 111 and 27 cannot be mapped at the locations shown in figure. 111 can be re-mapped to location 9 after colliding with 85. Similarly, 27 can be re-mapped to location 7, after colliding with 92.

Q5. Dynamic Programming and Greedy Algorithms (30 points total)
Given n intervals $I_i = [s_i, f_i]$, $i = 1, \ldots, n$, on the real line, we want to determine points p_1, $p_2, \ldots p_k$ such that for each i there is some j with $p_j \in I_i$, and such that k is as small as possible.

(a) Consider the following greedy strategy: *Choose a point that covers the largest number of intervals, remove those intervals from further consideration and iterate.* Show that this strategy does not result in an optimal solution. **(8 points)**

Solution (a): In the example in the figure (left) below, this greedy strategy would choose three points: first *a* that covers four intervals, and then *b* and *c*, each covering a remaining uncovered interval. The optimal solution (right), in contrast, needs only two points to cover all intervals.



(b) Propose another greedy strategy that produces an optimal solution. Prove that your greedy algorithm it is correct **(14 points)**. The resulting algorithm should be as efficient as possible. Give the computational complexity of your algorithm **(8 points)**

Solution: Order the intervals according to their right endpoint. When the next interval $[s_i, f_i]$ is considered, add its right endpoint f_i to the set of covering points and remove from the set of intervals those that are covered by f_i. To be more precise, we offer an efficient implementation. For simplicity, we assume that the interval endpoints are all different (we'll note how to take care of the case when some are equal).

```
MININTCOVER((s_i , f_i) : i = 1,…,n
   sort the s_i and f_i in increasing order into a single list
         of points x_1, x_2, ..., x_{2n-1}, x_{2n}
I ← {1, 2, 3, ..., n − 1, n}; P ← ∅; L ← ∅
for j = 1 to 2n do
      if x_j is s_k for some k then
            L ← L ∪ {k}
      if x_j is f_k for some k and k ∈ I then
            P ← P ∪ {f_k}
            I ← I – L
            L ← ∅
   return P
```

The algorithm considers the interval endpoints from left to right, keeping a list *L* of *active* intervals at each moment, namely, intervals that span $[x_{j-1}, x_j]$ and are not yet covered by a point in *P*, which maintains the set of covering points. *I* keeps the set of intervals not yet covered.

Correctness. Let P_j denote the set P at the end of the *j*-th iteration. We verify by induction that the set P_j can be extended into an optimal covering set of points. The basis of the induction is trivial. If *P* is not updated in an iteration, there is nothing to prove. So, consider the moment in which the line $P \leftarrow P \cup \{f_k\}$ is executed (a point is added to P_{j-1}). By induction hypothesis, P_{j-1} can be extended into an optimal set P'_{j-1}. But P_{j-1} did not cover I_k, so P'_{j-1} must include a point $p \le f_k$. If $p \le f_k$ then $P'_j = P'_{j-1}$ and $p < f_k$ we can substitute *p* with f_k in P'_{j-1} to obtain a solution P'_j that is an extension of P_j and is optimal: it is a solution because it still covers all the intervals, and it is optimal because it has the same size.

Running Time. The time for sorting is $O(n \log n)$. All the other work performed takes time $O(n)$ (with proper implementation; for example *I* should be implemented as an array). So the total time is $O(n \log n)$.