ESO207: Data Structures and Algorithms	*Time: 120min.*
Mid Semester Exam 1	*Max. Marks: 120*

Problem 1. For each of the problems listed below, (i) give a recurrence that describes its worst-case running time, and, (ii) give its worst-case running time using Θ notation. You need not derive your answers. $4 \times 3 = 12$

a. Randomized Quicksort. $T(n) = T(n-1) + \Theta(n)$, $T(1) = \Theta(1)$. Solution: $T(n) = \Theta(n^2)$. Also $T(n) = T(k) + T(n-k-1) + \Theta(n)$, $T(j) = \Theta(1)$, for $1 \leq j \leq k$, where, k is some constant is also correct and has solution $T(n) = \Theta(n^2)$.

b. Strassen's Algorithm. $T(n) = 7T(n/2) + \Theta(n^2)$. Here $n/2$ may be interpreted as $\lceil n/2 \rceil$ or $\lfloor n/2 \rfloor$. Solution: $T(n) = \Theta(n^{\log_2 7})$.

c. Merge Sort. $T(n) = 2T(n/2) + \Theta(n)$. Solution: $T(n) = \Theta(n \log n)$.

Problem 2. Short Answer Questions. Answer, with clear justifications, the following questions. $(3 \times 6 = 18)$

1. Let f and g be asymptotically non-negative and monotonically increasing functions. Then is it true that least one of the relationships $f(n) = O(g(n))$ or $g(n) = O(f(n))$ holds.

This question is now dropped from the exam. The answer is FALSE. Please see the end of this write-up for the solution.

2. Derive the solution to the recurrence relation $T(n) = T(n/4) + T(n/2) + \Theta(n)$, $T(1) = \Theta(1)$. (You may ignore the issue of floor and ceil, i.e., assume n is a power of 4.)

Guess the solution as $T(n) \leq cn$. By substitution,

$$T(n/4) + T(n/2) + dn \leq cn/4 + cn/2 + dn = 3cn/4 + dn \leq cn$$

provided, $c \geq 4d$.

3. Suppose that an array contains n numbers, each of which is $-1, 0$ or 1. Can this array be sorted in time $O(n)$ in the worst case?

Yes, using COUNTING-SORT, it can be sorted in time $O(n+3) = O(n)$.

4. Can there exist a comparison sort of 6 numbers that uses at most 9 comparisons in the worst case?

Any comparison sort uses $\log_2(n!)$ comparisons in the worst case for sorting n numbers. Since, $6! = 720$, a comparison sort must use $\log_2 720 > 9$ comparisons on some input.

5. Explain the main problem of open address hash tables that use linear probing.

The problem is of primary clustering, namely, there are long runs of contiguous occupied slots. The chance that a run of length i gets longer (assuming that keys map to the slots uniformly and independently) is $(i+1)/m$, that is, long runs have a better chance of becoming longer.

6. List two operations for which you would prefer Min priority queues over hash tables and vice-versa. Give reasons.

Min priority queues are better than hash tables for FIND-MINIMUM and EXTRACT-MINIMUM. Hash table are better for SEARCH and INSERT (expected time $O(1)$ versus $O(\log n)$).

Problem 3. You are given a sequence of n nuts and n bolts such that each nut has a corresponding bolt that it fits. Each nut matches exactly one bolt and vice-versal. Design an efficient algorithm that finds all corresponding nut and bolt matching pairs. It is not possible to compare a pair of nuts or a pair of bolts to determine which is larger or smaller. However, it is possible to try a nut to a bolt and determine whether the nut is of larger size than the bolt, or fits the bolt, or is of size smaller than the bolt. (30 points)
Note: For full credit, your algorithm must have either a worst-case or expected case time complexity of $O(n \log n)$. Give arguments to justify your time complexity.

Soln. Make a pass over the sequence of nuts and bolts and separate them into two arrays, one for the nuts and another for the bolts. Let the nuts array be denoted as N and the bolts array as B. We will design a recursive method to match nuts and bolts pairs by rearranging the N and the B arrays such that at the end of the procedure, $N[i]$ matches with $B[i]$, for $i = 1, 2, \ldots, n$. Call this procedure $Match(N, B, l, r)$ where, the call to *Match* assumes that the nuts in the array segment $N[l \ldots r]$ matches with the bolts in the array segment $B[l \ldots r]$ and after the call $N[i]$ matches $B[i]$ for each $i = l, \ldots, r$.

The *Match* procedure uses *Dual-Partition* which is a variant of Quicksort's Partition procedure. The *Dual-Partition*(N, B, l, r) assumes that the nuts in the array segment $N[l \ldots r]$ matches with the bolts in the array segment $B[l \ldots r]$. It rearranges $N[l \ldots r]$ and $B[l \ldots r]$ and returns an index j such that $N[l \ldots j-1]$ matches with $B[l \ldots j-1]$ after some rearrangement, $N[j]$ matches with $B[j]$ and $N[j+1 \ldots r]$ matches with $B[j+1 \ldots r]$ after some rearrangement.

We first describe *Dual-Partition*(N, B, l, r).

1. Let $N[l]$ be the pivot. This will be chosen randomly from $N[l \ldots r]$ in the outer recursion.
2. Use Quicksort's *Partition* on $B[l \ldots r]$ using $N[l]$ as pivot. This can be done since the Partition algorithm compares each element in the array with the pivot element. Since we can compare each bolt to the nut $N[l]$, the *Partition* procedure works correctly.
3. This returns an index j such that $B[l \ldots j-1]$ have smaller size than $N[l]$, and, $B[j]$ matches $N[l]$ and $B[j+1 \ldots r]$ have larger size than $N[l]$.
4. Now use $B[j]$ as the pivot and use Quicksort's *Partition* on $N[l \ldots r]$ to rearrange $N[l \ldots r]$.
5. This returns the same index $j' = j$ such that $N[l \ldots j-1]$ have smaller size than $B[j]$ and $N[j]$ matches $B[j]$ and $N[j+1 \ldots r]$ have larger size than $B[j]$. The index j is the same since each nut matches exactly one bolt and vice-versa.

The Matching procedure can now be described as follows: $Match(N, B, l, r)$.

1. If $l = r$ return.

2. Choose a random index $i \in \{l, l+1, \ldots, r\}$. Exchange $N[l]$ with $N[i]$.
3. Call *Dual-Partition*(N, B, l, r). Suppose it returns j.
4. If $j > l$, call *Match*$(N, B, l, j-1)$.
5. If $j < r$, call *Match*$(N, B, j+1, r)$.

The time complexity is the same as that of *Randomized Quicksort*, since the time taken by the *Dual-Partition*(N, B, l, r) procedure is $O(l - r + 1)$. Hence, the expected time complexity has the same order as that of the expected time complexity of *Randomized Quicksort*, which is $O(n \log n)$.

Problem 4. Design a data structure for a dynamic set of numbers that supports *Insert* in $O(\log n)$ time, *find the median* in $O(1)$ time and *delete the median* in $O(\log n)$ time. (30 points)

Soln. We will follow the convention that the median in a ranked list of n elements is the $\lfloor (n+1)/2 \rfloor$ th ranked element. For n odd, this gives the unique median, and for n even, it gives the "lower" median.

The data structure is as follows. Keep a max-heap H for the elements whose rank is at most $\lfloor (n+1)/2 \rfloor$. And keep a min-heap K for elements whose rank is $> \lfloor (n+1)/2 \rfloor$. These invariants will be maintained by the *Insert* and *delete median* operations. Let n track the number of elements in the heap. Consider each of the operations.

1. Finding the median is the simplest, since this returns the maximum element in the max-heap H.
2. *Delete median*: If $n = 2k + 1$ is odd, then, we call *Extract-Max* in max-heap H and reduce n by 1. The number of elements in H is $\lfloor (n+1)/2 \rfloor - 1 = k = \lfloor n/2 \rfloor$, as desired. The min-heap K does not change. *Extract-Max* takes $O(\log(n+1)/2) = O(\log n)$ time.

   Now suppose that $n = 2k$ is even. Then, (a) we call *Extract-Max* in max-heap H. Next (b) we transfer the minimum element in K to H, that is, let $r =$*Extract-Min*(K) and insert r into the max-heap H. All the operations require time $O(\log n)$.
3. *Insert item with key k*. This is done in two steps: (a) insert into the correct heap, and then balance the sizes of the two heaps, if necessary, by transferring the min or max element from one heap to the other.

   Increment n by 1. If the key k is smaller than or equal to the maximum key of H, then insert k into max-heap H and set n_1, the size of the H as $\lfloor n/2 \rfloor + 1$ and $n_2 = n - \lfloor n/2 \rfloor$. Otherwise, insert into min-heap K, and set $n_1 = \lfloor n/2 \rfloor$ and $n_2 = n + 1 - \lfloor n/2 \rfloor$. Now balance the structure to maintain the rank property of the invariant.

   (a) If $n_1 > n_2 + 1$, then, transfer the maximum element of H to K.
   (b) If $n_2 \geq n_1 + 1$ then transfer the minimum element of K to H.

   Transferring an element means call Extract-Min (or Extract-Max) on one heap and insert this element into the other heap.

Problem 5. Let U be the universe of keys with $|U| \gg m$. We consider families of hash functions, where each hash function maps U to $\{0, 1, \ldots, m-1\}$.

a Let m be a prime number. Let $\mathbb{Z}_m = \{0, 1, \ldots, m-1\}$ and $\mathbb{Z}_m^* = \{1, 2, \ldots, m-1\}$. For $a \in \mathbb{Z}_m$ and $b \in \mathbb{Z}_m^*$ define $h_{a,b}(k)$ as $a + b \cdot k \mod m$. Consider the family $\mathcal{H} = \{h_{a,b} \mid a \in Z_m, b \in \mathbb{Z}_m^*\}$. Is this family universal? (10)

Soln. If $k = l \mod m$, then, $h_{a,b}(k) = h_{a,b}(l)$ for every value of a, b. Hence, the family is not universal.

b A family $\mathcal{H}$ of hash functions is said to be 2-*universal* if for any $k, l \in U$ and distinct, and for any $0 \leq i, j \leq m-1$,

$$\Pr_{h \in \mathcal{H}} \{h(k) = i \text{ and } h(l) = j\} = 1/m^2 \ .$$

Give an example of a hash family that is universal but not 2-universal. (10)

Soln. . Consider $U = \{a, b, c\}$, $\mathcal{H} = \{h_1, h_2, h_3, h_4\}$, where, each of the functions $h_1, \ldots, h_4$ map the domain $\{a, b, c\} \to \{0, 1\}$ and are defined as follows.

	a	b	c
h_1	0	0	0
h_2	0	0	1
h_3	1	0	0
h_4	1	0	1

Let us count the number of solutions for $h(k) = h(l)$, for distinct k and l.

1. Let $\{k, l\} = \{a, b\}$. Then, for $h \in \{h_1, h_2\}$, $h(a) = h(b)$ and for $h \in \{h_3, h_4\}$, $h(a) \neq h(b)$. Hence, there are 2 out of 4 hash functions satisfying $h(a) = h(b)$.
2. Let $\{k, l\} = \{b, c\}$. Then, for $h = h_1$ or h_3, $h(b) = h(c)$ and for $h = h_2$ or h_4, $h(a) \neq h(b)$. Hence there are 2 out of 4 hash functions satisfying $h(b) = h(c)$.
3. Lt $\{k, l\} = \{a, c\}$. Then, for $h = h_1$ or $h = h_4$, $h(a) = h(c)$ and for $h = h_2$ or h_3, $h(a) \neq h(c)$. Hence there are 2 out of 4 hash functions satisfying $h(a) = h(c)$.

The above calculations show that for any $k, l \in \{a, b, c\}$ and distinct, $\Pr_{h \in H} \{h(k) = h(l)\} = 2/4 = \frac{1}{2} = \frac{1}{m}$, since, $m = 2$. Thus, the family is universal.

However, the family is not 2-universal, since, there is no solution to $h(a) = 0$ and $h(b) = 1$. Hence, $\Pr_{h \in H} \{h(a) = 0 \text{ and } h(b) = 1\} = 0$, whereas for the family to be 2-universal, this probability should have been $1/m^2 = 1/4$.

c Under simple uniform hashing assumption and collision resolution using chaining, calculate the expected value of the square of the length of a chain at a given slot. (10)

Soln. Let s be a slot. For $k \in K$, define the indicator variable X_k as follows:

$$X_k = \begin{cases} 1 & \text{if } k \text{ hashes to slot } s \\ 0 & \text{otherwise.} \end{cases}$$

The chain length C_s at slot s is

$$C_s = \sum_{k \in K} X_k$$

Therefore,

$$C_s^2 = \left(\sum_{k \in K} X_k \right)^2 = \sum_{k \in K} X_k^2 + 2 \sum_{\substack{k,l \in K \\ k \neq l}} X_k X_l$$

First we note that $X_k^2 = X_k$ since X_k is 0/1. Now taking expectation, $\mathbb{E}[X_k] = 1/m$ by uniformity assumption, and $\mathbb{E}[X_k X_l] = \Pr[X_k = 1 \text{ and } X_l = 1] = 1/m^2$, by uniformity assumption (and independence implied in the uniformity assumption). Now using linearity of expectation,

$$\begin{aligned} \mathbb{E}\left[C_s^2\right] &= \sum_{k \in K} \mathbb{E}[X_k] + 2 \sum_{\substack{k,l \in K \\ k \neq l}} \mathbb{E}[X_k X_l] \\ &= \frac{n}{m} + 2\binom{n}{2}\frac{1}{m^2} = \frac{n}{m} + \frac{n(n-1)}{m^2} \end{aligned}$$

Q 2.1 Let f and g be asymptotically non-negative and monotonically increasing functions. Then is it true that least one of the relationships $f(n) = O(g(n))$ or $g(n) = O(f(n))$ holds.

Divide the positive real line into contiguous intervals $[x_1, x_2), [x_2, x_3), \ldots, [x_i, x_{i+1}), \ldots$ such that $x_1 = 1$ and $x_{i+1} = x_i + x_i^2$. Define two functions $f(\cdot)$ and $g(\cdot)$ on the intervals such that, $f(x)$ grows linearly on the odd numbered interval and stays constant on the even numbered intervals.

$$f(x) = \begin{cases} \text{is a straight line joining } (x_i, x_{i-1}) \text{ with } (x_{i+1}, x_{i+1}) \ . & \text{if } x \in [x_i, x_{i+1}) \ i \text{ odd} \\ x_i & \text{if } x \in [x_i, x_{i+1}),\ i \text{ even} \end{cases}$$

$g(\cdot)$ stays constant on the odd numbered intervals and grows linearly on the even numbered intervals.

$$g(x) = \begin{cases} \text{is a straight line joining } (x_i, x_{i-1}) \text{ to } (x_{i+1}, x_{i+1}) & \text{if } x \in [x_i, x_{i+1}),\ i \text{ even} \\ x_i & \text{if } x \in [x_i, x_{i+1}) \text{ and } i \text{ is odd} \end{cases}$$

Consider the ratio $f(x)/g(x)$ at interval boundaries. For i odd, $f(x_i) = x_{i-1}$ and $g(x_i) = x_i$. Thus,

$$\frac{f(x_i)}{g(x_i)} = \frac{x_{i-1}}{x_i} \leq \Theta(1/\sqrt{x_i})$$

For i even, $f(x_i) = x_i$ and $g(x_i) = x_{i-1}$ and so

$$\frac{f(x_i)}{g(x_i)} = \frac{x_i}{x_{i-1}} \geq \Theta(\sqrt{x_i})$$

Hence neither function is within $O(1)$ of the other function.