Sample Prelim
CS 410, Summer 2000

Please note:

- The exam will be closed book, closed note.
- Partial credit will be available for all questions. However, be concise. Long answers will take your time and may hurt you.

Here are some theorems and definitions you may want:
(probably a longer list on the exam; otherwise, I'd just say "Here's the Master theorem")

- ***Theorem 4.1 (Master theorem)***
  Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence
  $$T(n) = a\,T(n/b) + f(n),$$
  where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$.
  Then $T(n)$ can be bounded asymptotically as follows.

  1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
  2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
  3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$,
     and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n,
     then $T(n) = \Theta(f(n))$.

Sample questions:

1. (20 points) Give asymptotic solutions for each of the following recurrences. Justify answers.

   (a) $T(n) = 2T(n/2) + n^{1/2}$

   Solution : a=2, b=2 and $\lg_b a = 1$. Case (1) of the master theorem applies . Hence T(n)=$\theta(n)$.

   (b) $T(n) = 5T(n-1)$

   Solution : T(n)=5T(n-1)=5^2T(n-2)=5^3T(n-3)= ... =5^{n-1}T(1). Hence T(n) = $\theta(5^n)$.

   (c) $T(n) = 4T(n/2) + \lg n$

   Solution : a=4, b=2 and $\lg_b a = 2$. Case (1) of the master theorem applies . Hence T(n)=$\theta(n^2)$.

   (d) $T(n) = 4T(n/2) + n \lg n$

   Solution : a=4, b=2 and $\lg_b a = 2$. Case (1) of the master theorem applies . Hence T(n)=$\theta(n^2)$.

(e) $T(n) = 4T(n/2) + n^2$

Solution : a=4, b=2 and $\lg_b a = 2$. Case (2) of the master theorem applies . Hence T(n)=$\theta(n^2 \lg n)$.

2. (10 points) Give a permutation of the set of numbers 1,2,3,4,5,6,7,8 which causes the fewest recursive calls by the quicksort algorithm discussed in class. Justify your answer.

   Solution We know that quicksort takes the least amount of time when the sub-halves to be sorted are of the same size. By a process of reverse engineering we can arrive at the following input for which each call to partition splits the given array into 2 equal halves.

   5 1 2 4 7 3 6 8

   Recall from our analysis of quicksort that if the pivot is the rank i element (i.e. the i^{th} smallest element) and $i > 1$, then the array partitions into subarrays of size $i-1$ and $n-i+1$, so for creating equal sized subarrays of an even-length array, we'd like the pivot element to be of rank $n/2+1$. So, we need the 5th smallest element of 8, and the third smallest element of a size 4 array. Once the array has been partitioned into arrays of size 2, any pivot will divide the arrays into equal sized subarrays of size 1, so we only need to carefully place the original pivot element and the pivot elements for the two subarrays of size 4.

   In the configuration (permutation) shown, in the first iteration of *Partition*, 5 will be swapped with 3, positioning 3 into the pivot position for the next *Partition* on the size 4 array. The 7 will remain where it is, already positioned to be the pivot element for the other size 4 array.

3. (35 points, 5 points each) Short answer questions

   (a) Is the following statement true or false?
   The number of times the procedure quicksort() is recursively called on an array of some fixed size n is independent of the order of the input.
   What about merge sort() ?
   Justify your answers.

   Solution : quicksort() : NO. Recall that for most inputs quicksort is $O(n \lg n)$, but for worst case it is different, $\Theta(n^2)$
   The function partition() depends upon the input data and hence the sizes of the resulting 2 sub-problems depends on the input. Hence the number of times the resulting 2 problems call quicksort() depends on their respective sizes and again on the contents and this recurses. Hence clearly the number of times quicksort is called is heavily dependent on the input data.
   mergesort() : YES . Clearly from the merge-sort algorithm, we see that the 2 smaller problems called are roughly half the size and is independent of the input. This is true for all levels of recursion.

   (b) Which of the following sorting algorithms use a constant amount of space other than the input array?
   insertion sort, heapsort, merge sort, quicksort, radix sort.

   Solution : Only insertion and heap sort use constant space.

   (c) Consider $S = 1 + 1/2 + 1/3 + 1/4 + \ldots$. Which of the following is true?

      i. S=O(1)

ii. S=theta(1)
iii. S=Omega(1)
iv. S=o(1)

Note: Here we refer to the size of S, not the time needed to compute S.

Solution : $S = \Omega(1)$

(d) Rank the following functions by order of growth. That is, find an arrangement $f_1, f_2, \ldots, f_n$ of the functions satisfying $f_1 = O(f_2), f_2 = O(f_3)$, etc. In each case indicate whether $f_i = \Theta(f_{i+1})$.
$n,\ 1,\ n^2,\ \lg n,\ 17,\ n \lg n,\ n/\lg n,\ \lg(n^2)$.

Solution : $1, 17, \lg n, \lg n^2, \frac{n}{\lg n}, n, n \lg n, n^2$
$1 = \theta(17)$
$\lg n = \theta(\lg n^2)$

(e) Explain why double hashing is better than hashing with linear probing.

Solution : The permutations produced by double hashing have many of the characteristics of randomly chosen permutations.Hence there is no primary or secondary clustering.

(f) Illustrate the operation of Bucket Sort on the array
$A = \langle\ .28,\ .73,\ .51,\ .17,\ .04,\ .58,\ .26,\ .43,\ .99,\ .24\ \rangle$.
Show each step/change as you did in your homework assignment.

Solution : There are 10 buckets : [0,.1), [.1,.2),... , [.9,1) .Let the 10 buckets be $b_0, b_1, \ldots, b_9$.
b_0 : .04
b_1 : .17
b_2 : .28, .26, .24
b_3 :
b_4 : .43
b_5 : .51, .58
b_6 :
b_7 : .73
b_8 :
b_9 : .99

(g) What is an abstract data structure (sometimes also called an abstract data type)? Give an example.

Solution : A collection of data elements together with a set of operations performed on the data. Examples are a priority queue, dictionary.

4. (15 points) Consider a heap of $n = 2^k - 1$ distinct numeric keys stored in an array A which is indexed 1 through n with the *largest* key (max value) at the root.

(a) At what positions in the array might the *second* largest key be found?
(b) At what positions in the array might the *smallest* key be found?
(c) At what positions in the array might the i^{th} smallest key be found?

Solution The second largest element occurs in positions 2 or 3 , the smallest in the last 2^{k-1} positions and the i^{th} smallest in the last $(2^{k-1} + 2^{k-2} + \ldots + 2^{k-\lfloor \lg i \rfloor -1})$ positions. (the second largest element is one of the elemets in **level 1**, the minimum is in the **last level** and the i^{th} smallest is in the last $\lfloor \lg i \rfloor + 1$ **levels** .)

5. (20 points) Describe an algorithm that, given n integers in the range 1 to k, preprocesses its input and then answers any query about how many of those integers fall *outside* a range $[a..b]$ in $O(1)$ time. Your algorithm may use $O(n + k)$ preprocessing time, but must then answer any number of range queries in time $O(1)$ for each query.

   For example, given input $1, 6, 4, 3, 8, 7, 9$ and query $[2..7]$ your algorithm should return the answer 3, since the three elements 1, 8, and 9 are outside the specified range.

   Hint: Your answer must run in $O(1)$ time, not $O(b - a)$. If you have an algorithm that determines how many of the integers are *equal* to x, you cannot simply enumerate all the keys between a and b and call your algorithm on each of them, because that takes too long – $\Omega(b - a)$ steps. Suppose you had an algorithm that determines how many of the integers are *less than* x in $O(1)$ time. You can extend such an algorithm to a solution of this problem.

   Solution *Outline of the algorithm :* We can determine the number of intergers less than j, where j is an integer and $1 \leq j \leq k$. (*see counting sort*). Let $C[j]$ denote the number of integers lej. This takes time O(n+k). Given a and b , if $a < 1$ then $a = 1$ and similarly if $b > k$ then $b = k$. The number of integers which fall in the range [a..b] is $x = C[\lfloor b \rfloor] - C[\lceil a \rceil]$. Hence the required answer is $n - x$.