# 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 your answers.
   (a) $T(n) = 2T(n/2) + n^{1/2}$
   (b) $T(n) = 5T(n-1)$
   (c) $T(n) = 4T(n/2) + \lg n$
   (d) $T(n) = 4T(n/2) + n \lg n$
   (e) $T(n) = 4T(n/2) + n^2$
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.

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.

   (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.

   (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 $S$.

   (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 $f_i = \Theta(f_{i+1})$.
   $n,\ 1,\ n^2,\ \lg n,\ 17,\ n \lg n,\ n/\lg n,\ \lg(n^2)$.

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

   (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.

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

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?

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.