MIDTERM (with solutions)

Please answer all questions. Read carefully. This is a closed book exam, but you may refer to a "cheat sheet" of prepared notes (8" by 11", 2-sided). Remember that we give partial credits. To get full credit, working MUST be shown. Reserve a separate part of the exam book for your scratch work. Good Luck!

Problem 1 TRUE OR FALSE QUESTIONS [10 points each]

Brief justification is required. Negative 3 points for wrong answers (0 points for no answers).

(a) It is possible to sort 5 elements using at most 6 comparisons.

ANSWER: FALSE. This is a question about the worst case number of comparisons to sort 5 elements. By information theoretic bound, we need at least $\lg(5!) = \lg(120) > 6$.

PITFALLS: some people say that it is false because sorting of n elements need $n \log n$ comparisons and $5 \log 5 > 6$. The problem is that "$n \log n$" is too imprecise. This is because this has no base. If you say "base 2" this is also wrong. This is because if you use "base e", we get $\log_e(n!) = n \ln n - n + \ln n + O(1)$ This comes from equation (1.19) on page 27 of the text.

(b) In the best case, we can determine the median of n elements in $n-1$ comparisons among keys.

ANSWER: TRUE. For instance, if we happen to choose the median and compare it to all the other elements, we would have confirmed the median in $n-1$ comparisons. This might be the partition step in a recursive algorithm.

(c) Suppose $T(n)$ and $T'(n)$ satisfy the recurrences

$$T(n) = 30T(n/3) + n, \qquad T'(n) = 40T'(n/4) + n,$$

respectively. Then $T(n) = O(T'(n))$.

ANSWER: FALSE. The critical exponents for $T(n)$ and $T'(n)$ are $\alpha = \log_3(30) > \log_3(27) = 3$ and $\alpha' = \log_4(40) < \log_4(64) = 3$. By the Master Theorem, both $T(n) = \Theta(n^\alpha)$ and $T'(n) = \Theta(n^{\alpha'})$. But n^α is not $O(n^{\alpha'})$.

(d) Consider the following two functions.

```
int strange (int n) {
    if (n == 0) return 1;
    return strange(n − 1) + strange(n − 1);
}
```

```
int stranger (int n) {
    if (n <= 0) return 1;
    return stranger(n − 2) + stranger(n − 2);
}
```

Let their running times be $S(n)$ and $T(n)$, respectively. Then $S(n) = \Theta(T(n))$.
ANSWER: FALSE: $S(n) = 2S(n-1)+1$ and $T(n) = 2T(n-2)+1$. The solution is $S(n) = \Theta(2^n)$ and $T(n) = \Theta(2^{n/2})$. These two functions are not the same Θ-order.

PITFALLS: It is very easy to get fooled and say that they are the same Θ-order. If you get to this point, and say they are the same Θ-order, you get partial credit.
(e) This is a game of chance where you either win or lose. If you win, you are ahead by \$10. If you lose, you are behind by \$1. The probability of winning is 0.1. You should stay home: it is a waste of time playing a game that is apparently neutral.
ANSWER: FALSE: Let X be the random variable where $X = 10$ if you win, and $X = -1$ if you lose. Then $E[X] = 0.1 \times 10 + 0.9 \times (-1) = 1 - 0.9 = 0.1$. Since you are expected to win \$ 0.1 per game, you should play and make some money (better than staying home).

PITFALL: it is the expectation, not the probability that matters.

Problem 2 ASYMPTOTICS [10+10+15 points]

(a) Let $f(n) = n^2 - 100n$ and $g(n) = 100\sqrt{n}$. Prove that $g(n) = O(f(n))$. You must use the definition of the big-Oh notation and produce the requisite constants. *BE CAREFUL to prove that your inequality holds for ALL sufficiently large values of n, and NOT just at a single value of n.*
ANSWER: We must show that $100\sqrt{n} \le C(n^2 - 100n)$ when $n \ge n_0$ for some $C > 0$ and some n_0. We can choose $C = 1$ and $n_0 = 200$. Then we have

$$g(n) = 100\sqrt{n} \le 100n \le n(n-100) = f(n)$$

(each of the inequalities is seen to be true for $n \ge 200$). You can try other values for C and n_0, of course.

REMARK: if you use limits (as in Calculus) you lose some points because that is NOT going back to the definition of big-Oh. E.g., using limits, you will not be able to produce the constants we need.
(b) Let $S(n) = \sum_{i=1}^{n} i^2 \log i$. Prove that $S(n) = \Theta(n^3 \log n)$. Be sure to prove an upper AND a lower bound.
ANSWER: Clearly, $S(n) = O(n^2 \log n)$ (replace every term by the maximum term of $n \log n$). Conversely, $S(n) \ge (n/2)^2 \log(n/2)$ (discard smaller half of the terms, and replace the remaining terms by the smallest which is $(n/2)^2 \log(n/2)$).
(c) Order the following functions in increasing Θ-order:

(i) $4^{\lg n} + \lg^4 n$	(ii) $n^{1/\lg n}$
(iii) $2^{n \lg n}$	(iv) $\lg(H_n)$
(v) $n^{\lg n}$	(vi) $n!$
(vii) $2^{n/2}$	(viii) $1/\lg n$
(ix) $\sqrt{n} \lg n$	(x) $n^2 + 200n \lg^2 n$.

When two or more functions have the same Θ-order, you must group them together. Recall that $\lg n = \log_2 n$, $\lg^k n$ means $(\lg n)^k$, and H_n is the nth Harmonic number.
ANSWER: Here is the ordering:
(1) (viii) $1/\lg n$,

(2) (ii) $n^{1/\lg n} = \Theta = O(1)$,
(3) (iv) $\lg(H_n) = \Theta(\lg\lg(n))$,
(4) (ix) $\sqrt{n}\lg n$
(5,6) (i)=(x): $4^{\lg n} + \lg^4 n = \Theta(n^2)$ and $n^2 + 200n\lg^2 n = \Theta(n^2)$,
(7) (v): $n^{\lg n}$,
(8) (vii) $2^{n/2}$,
(9) (vi) $n! = (n/e)^n\sqrt{2\pi n}(1+\epsilon)$ (where $0 < \epsilon < 1/(11n)$),
(10) (iii): $2^{n\lg n}$.
Note that the last 2 is slightly tricky to order, but you should convert to a common base of 2. But look at the answer to question 1(a) for note about $n!$.

Problem 3 RECURRENCES [20 points]
Solve the recurrence using the EGVS method

$$T(n) = 9T(n/3) + n^3.$$

ANSWER:

$$\begin{aligned}
T(n) &= 9T(n/3) + n^3 \\
&= 9^2T(n/3^2) + 9(n/3)^3 + n^3 \quad (EXPAND) \\
&= 9^3T(n/3^3) + 9^2(n/3^2)^3 + 9(n/3)^3 + n^3 \quad (EXPANDAGAIN) \\
&= 9^3T(n/3^3) + n^3[1/3^2 + 1/3 + 1] \\
&= \ldots \\
&= 9^iT(n/3^i) + n^3[\sum_{j=0}^{i-1} 3^{-j}] \quad (GUESS!) \\
&= 9^{i+1}T(n/3^{i+1}) + 9^i(n/3^i)^3 + n^3[\sum_{j=0}^{i-1} 3^{-j}] \\
&= 9^{i+1}T(n/3^{i+1}) + n^3[\sum_{j=0}^{i} 3^{-j}] \quad (VERIFIED!)
\end{aligned}$$

Now to STOP. Choose $i = \lceil \log_3 n \rceil$ and assume $T(n/3^i) = 0$ for this choice. Then $T(n) = n^3[\sum_{j=0}^{i-1} 3^{-j}] = O(n^3)$.

PITFALL: Many students fail to do the first step (expansion) correctly!

Problem 4 SORTING [15+10 points]
(a) Suppose that in the base case of QuickSort, we do nothing whenever the size of the subarray to be sorted has 10 or less keys. Call this "`QuirkSort`". (i) Describe the nature of the output from `QuirkSort`. (ii) Describe a linear time method to take the output of `QuirkSort` and make it into a sorted array. HINT: one of the sorting methods we know can be used. (iii) Explain why your method takes linear time.
ANSWER: (i) `QuirkSort` return an array in which the keys are separated into groups by pivot elements. Each group has up to 10 keys, The relative order of the groups are correct (i.e., all the elements of one group are less than all the elements in another group that follows it in the array). Thus every key is at most 9 places away

from its final position. So the result is ALMOST SORTED.
(ii) To fix the output of `QuirkSort`, you can simply call insertion sort on the array. The point of this part is that, by looking at the array from QuirkSort, you do not know where the groups are (there are no markers to tell you which are pivot elements). But insertion sort will still do the right thing!
(iii) Recall that in insertion sort, the distance moved by the key that you are trying to insert is equal to the number of comparisons it make. This means that each key makes at most 9 comparisons, and time to insert one key is $O(1)$. So the overall time is $O(n)$.
(b) Suppose the following array (indexed from 1 to 9) of numbers is given:

Position:	1	2	3	4	5	6	7	8	9
Key:	8	3	4	5	2	9	7	6	1

We want to convert this into a heap by repeated application of the `fixHeap` operation. You can use the `constructHeap` algorithm in the book, or any method we may have discussed in class. Show the transformations of the array after each `fixHeap` operation. Please do not show any intermediate states. HINT: it is easier to draw the array in the shape of a binary tree.
ANSWER: One way is to do `fixHeap` from index $i = \lfloor n/2 \rfloor$ down to $i = 1$ (here $n = 9$). Assume that we are constructing a MaxHeap (as opposed to a MinHeap). In the following, we underline the key (and their two children) that will be the examined in the next round of `fixHeap`.

Position	1	2	3	4	5	6	7	8	9
Input	8	3	4	$\underline{5}$	2	9	7	$\underline{6}$	$\underline{1}$
Fix(4)	8	3	$\underline{4}$	6	2	$\underline{9}$	$\underline{7}$	5	1
Fix(3)	8	$\underline{3}$	9	$\underline{6}$	$\underline{2}$	4	7	5	1
Fix(2)	$\underline{8}$	6	$\underline{9}$	$\underline{5}$	2	4	7	3	1
Fix(1)	9	6	8	5	2	4	7	3	1

Drawing these arrays as heap structures will may the transformations easier to see! You should do this in your answers.

PITFALL: many students do not have a systematic method to do the fixHeap. This is not acceptable.

Problem 5 SELECTION [20 points]
Devise an algorithm that uses as few key comparisons as possible to compute the second smallest and the second largest keys in an input set. Determine the worst-case number of comparisons used by your algorithm. HINT: you may combine the ideas in two optimal selection-type algorithms in the book.
ANSWER: We combine the ideas of two optimal algorithms: the $1.5n$ comparison algorithm for the min and max, and the $\lceil \lg n \rceil - 1$ extra comparison method for finding the second largest. Thus, we need only $1.5n + 2\lceil \lg n \rceil - 2$ comparisons.

Briefly, the algorithm proceeds as follows: first pair up all the keys, and compare them. Then construct a balanced binary "tournament" tree for the winners of the first round to find the best player. Similarly, construct a tournament for the losers of the first round, but to find the worst player. This takes $1.5n$ comparisons as in the optimal min-max algorithm. Now, we "remove" the best player from the winners tournament, and recompute the next best player. This will be the second best player. This takes $\lceil \lg n \rceil - 1$ comparisons. Similarly, compute the second worst player.