Quiz 1

- Do not open this quiz booklet until you are directed to do so. Read all the instructions first.
- When the quiz begins, write your name on every page of this quiz booklet.
- The quiz contains five multi-part problems. You have 120 minutes to earn 120 points.
- This quiz booklet contains **14** pages, including this one. An extra sheet of scratch paper is attached. Please detach it before turning in your quiz.
- This quiz is closed book. You may use one handwritten A4 or $8\frac{1}{2}'' \times 11''$ crib sheet. No calculators or programmable devices are permitted.
- Write your solutions in the space provided. Extra scratch paper may be provided if you need more room, although your answer should fit in the given space.
- Do not waste time and paper re-deriving facts that we have studied. It is sufficient to cite known results.
- Do not spend too much time on any one problem. Read them all through first, and attack them in the order that allows you to make the most progress. Generally, a problem's point value is an indication of how much time to spend on it.
- Show your work, as partial credit will be given. You will be graded not only on the correctness of your answer, but also on the clarity with which you express it. Be neat.
- Good luck!

Problem	Points	Grade	Initials
1	20		
2	15		
3	30		
4	25		
5	30		
Total	120		

Name: Solutions

Circle your recitation:

Brian 11 Brian 12 Jen 12 Jen 1 Brian 2

Problem 1. Recurrences [20 points] (5 parts)

For each of the following algorithms in parts (a)-(c), provide a recurrence relation and give an asymptotically tight ($\Theta(\cdot)$) bound. For parts (d)-(e), just provide an asymptotically tight bound. Justify your answer by naming the particular case of the Master Method, by iterating the recurrences, using the substitution method, or using Akra-Bazzi.

Example: [0 points] BINARY SEARCH

Recurrence: $T(n) = T(n/2) + c$

Solution by iteration:

$$T(n) = T(n/4) + c + c = \sum_{i=0}^{\log n} c = c \log n = \Theta(\log n)$$

(a) [4 points] MERGE SORT

Solution: Recurrence: $T(n) = 2T(n/2) + \Theta(n)$.
$T(n) = \Theta(n \log n)$ by part 2 of the Master Method.

(b) [4 points] KARATSUBA INTEGER MULTIPLICATION
(The Divide and Conquer algorithm from Lecture 2 and Problem Set 1-4.)

Solution: Recurrence: $T(n) = 3T(n/2) + \Theta(n)$
$T(n) = \Theta(n^{\log_2 3})$ by part 1 of the Master Method.

(c) [4 points] DETERMINISTIC SELECT (Code provided on the last page of this exam.)

Solution: Recurrence: $T(n) \leq T(n/5) + T(7n/10) + \Theta(n)$
Using a recursion tree method, depicted in Figure 1, we deduce that:

$$T(n) = \left(\sum_{i=0}^{\log_5 n} (\frac{9}{10})^i n \right) = \Theta(n)$$

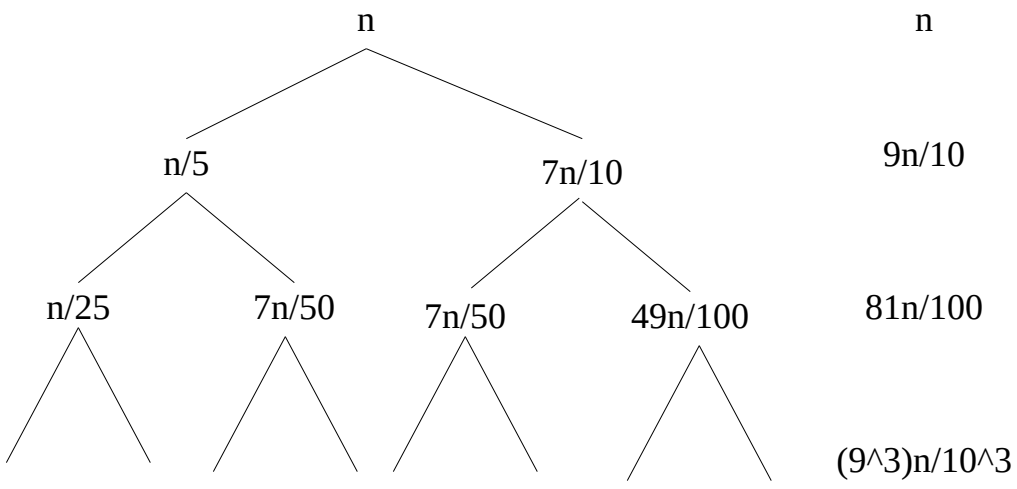

Figure 1: Recursion tree for Deterministic Select

(d) [4 points] $T(n) = 7T(n/2) + n^3$.

Solution:
$T(n) = \Theta(n^3)$ by part 3 of the Master Method.

(e) [4 points] $T(n) = 2T(n/4) + 5\sqrt{n}$.

Solution: $T(n) = \Theta(\sqrt{n}\log n)$ by part 2 of the Master Method.

Problem 2. True or False, and Justify [15 points] (5 parts)

Circle **T** or **F** for each of the following statements, and briefly explain why. The better your argument, the higher your grade, but be brief. Your justification is worth more points than your true-or-false designation.

(a) T F [3 points] RADIX SORT works correctly if we sort each individual digit with HEAPSORT instead of COUNTING SORT.

Solution: False. HEAPSORT is not stable. If the first $i-1$ digits are sorted in RADIX SORT, sorting the ith digit using stable COUNTING SORT won't change the order of two items wiht the same ith digit. This cannot be guaranteed with HEAPSORT

(b) T F [3 points] The following array A is a Min-Heap:

$$1 \quad 11 \quad 3 \quad 12 \quad 100 \quad 5 \quad 10 \quad 13 \quad 50$$

Solution: True. $\forall i(A[i] \leq A[2i]) \wedge (A[i] \leq A[2i+1])$

(c) T F [3 points] RADIX-SORT can be used to sort n elements with integer keys in the range from 1 to $n^{f(n)}$ in $O(nf(n))$ time.

Solution: True. By using base n representations, each COUNTING-SORT takes $O(n)$ time, and there are a total of $\log_n n^{f(n)} = f(n)$ digits, for a runtime of $O(nf(n))$ time.

(d) T F [3 points] There is an $\Omega(n \log n)$ lower bound to build a Min-Heap of size $\Theta(n)$ in the comparison model.

Solution: False. Using the BUILD HEAP procedure from CLRS, we can build a new heap in $\Theta(n)$ time.

(e) T F [3 points] If we only assume that all buckets have the same size, BUCKET SORT runs in $O(n)$-time on average independent of the input distribution.

Solution: False. By picking an input distribution that overloaded a single bucket, you could force BUCKET SORT to run in the worst-case time of it's subroutine sorting algorithm ($O(n^2)$ for INSERTION SORT).

Problem 3. Short Answer [30 points] (3 parts)

Give *brief*, but complete, answers to the following questions.

(a) [5 points] Describe the difference between ***average-case*** and ***worst-case*** analysis of algorithms, and give an example of an algorithm whose average-case running time is different from its worst-case running time.

Solution: An average-case analysis assumes some distribution over the inputs (e.g., uniform), and computes the expected (average) running time of an algorithm subject to that distribution. A worst-case analysis considers those inputs which force an algorithm to run for the longest amount of time, and computes the running time under those inputs. QUICKSORT has a worst-case running time of $\Theta(n^2)$ (on an already-sorted or reverse-sorted array), but has an average-case running time of $\Theta(n \log n)$ (assuming all input permutations are equally likely).

(b) [10 points] Suppose you are given an array A of size n that either contains all zeros or $2n/3$ zeros and $n/3$ ones in some arbitrary order. Your problem is to determine whether A contains any ones.

1. Give an exact lower bound in terms of n (not using asymptotic notation) on the worst-case running time of any deterministic algorithm that solves this problem.
2. Give a randomized algorithm that runs in $O(1)$-time and gives the right answer with probability at least $1/3$.
3. Give a randomized algorithm that runs in $O(1)$-time and gives the right answer with probability at least $5/9$.

Solution:

1. Any correct deterministic algorithm must look at $2n/3 + 1$ entries, because if it didn't, it could see all zeros even when there was a one somewhere.
2. Pick a random location. If it is a 1, output "has ones". Otherwise, output "doesn't have ones". Clearly the algorithm runs in $O(1)$ time. If the array is all zeros, it is always correct.
   However if the array contains $1/3$ ones, the algorithm may make a mistake. Since there are $n/3$ ones, you will select a 1 value at random and correctly output "has ones" with probability $1/3$.
3. Pick two random locations. The correctness and $O(1)$ runtime are identical as above. Again, if the array is all zeros, the algorithm is always correct.
   A mistake arises if the array contains ones, but the algorithm picks two zeros. Picking a single zero occurs with probability $2/3$, and picking two zeros independently occurs with probability $(2/3)^2 = 4/9$. Therefore, the probability that the algorithm is correct is $1 - (4/9) = 5/9$.

(c) [15 points] A *k-multiset* is a set of n elements in which k distinct elements each appear exactly n/k times. For example, $\{1, 1, 2, 2, 3, 3, 4, 4, 5, 5\}$ is a sorted 5-multiset of size $n = 10$. Give an $\Theta(n \log k)$-time algorithm to sort a k-multiset of size n. For convenience, you may assume that $n = 2^i$ and $k = 2^j$ for some $i \geq j$.

Solution: On the k-multiset S, do the following:

1. Use DETERMINISTIC SELECT to find the median s_m of S.
2. Partition S around s_m.
3. Since there are n/k values equal to s_m, the partition may not split the set exactly in half. Extract all values equal to s_m from both sides of the partition and add them to the "lighter" of the two sublists.
4. Recurse on each subproblem if it has size greater than n/k.

Each level of recursion will split the set S into two subproblems of size $S/2$ in $O(n)$ time. This algorithm will halt with subproblems of size n/k. Since $n/k = n/2^j$, this algorithm will recurse exactly $j = \log k$ levels before halting on sublist leaves of size exactly n/k.

Consider the "least" leaf produced by our partitions. It will contain the n/k least elements of the array, which all happen to be the same value. Similarly, the next "least" leaf will contain the next n/k least elements, which are also all the same value. Inductively, we can see that each n/k-sized leaf is homogeneous. After $j = \log k$ levels of recursion, the algorithm can just output its leafs in order.

$O(n)$ work is done on each level of recursion and there are $\log k$ levels of recursion, so this algorithm takes $O(n \log k)$ total time.

Problem 4. Gaagle's Cappuccino Machine [25 points] (1 parts)

You've been hired by the high-tech start-up Gaagle to find an ideal location for their new cappuccino machine. Gaagle's offices are arranged at various points along a single long corridor. Gaagle wishes to minimize the total distance walked by all their employees. You may assume the following:

- Each employee visits the cappuccino machine exactly the same number of times each day.
- Gaagle has an odd number of employees.

Describe where you should place the cappuccino machine and prove that it is optimal.

Solution:
Suppose Gaagle's employees occupy offices $(p_1, \ldots, p_n)$. Considering a particular placement x, let's consider the sets $L = \{p_i | p_i < x\}$ and $G = \{p_i | p_i > x\}$ would result in the employees having to walk:

$$\sum_{p_i \in L} (x - p_i) + \sum_{p_i \in G} (p_i - x)$$

Suppose x is the median office location p_m. For convenience let $d = p_{m+1} - p_m$. If we changed x to be p_{m+1}, it would result in each of the $(n-1)/2$ person in L and the person at p_m having to walk an additional d distance. Each of the $(n-1)/2$ people in G would have to walk d less distance. Thus, moving the machine from the median results in a net gain of d distance walked.
The same argument holds if we move x to p_{m-1}. Therefore, since a move in either direction necessarily increases the total distance walked, placing the cappuccino machine at the median is optimal.

Problem 5. Perfect Powers [30 points] (4 parts)

A perfect power is an integer X, for which there exist integers $B \geq 2$ and $e \geq 2$ such that $X = B^e$. In this problem, you will come up with a fast algorithm that on input X, outputs B and e such that $X = B^e$. If no such values exist, your algorithm will output **null**.

If you are unable to answer one part of the problem, you may assume its results for the other parts. Take care not to confuse the input *lengths* with the input *values*.

(a) [3 points] Let X be an n-bit integer. If $X = B^e$, prove that $(e \leq n)$.

Solution: First, since X is a n-bit integer $(n \geq \log X = e \log B)$. Since $(B \geq 2)$, then $(e \leq e \log B \leq n)$.

(b) [5 points] Give an $O(n^2 \log n)$-time algorithm that on inputs B and e, computes B^e. Assume that $B^e \leq 2^n$ and that multiplying an n-bit integer by an m-bit integer takes $O(nm)$ time.

Solution:

EXPONENT(B, e):
1 $k \leftarrow \log e$ $\triangleright$ e is a k-bit integer: $e = 1e_{k-1} \ldots e_0$
2 $X \leftarrow B$ $\triangleright$ $X = B^{e_k}$ and $e_k = 1$
3 **for** $i \leftarrow k - 1$ **downto** 1:
4 $X \leftarrow 2 * X$ $\triangleright$ $X = B^{b_k \ldots b_{i+1} 0}$.
5 **if** $(e_i = 1)$ $X \leftarrow X + B$ $\triangleright$ $X = B^{b_k \ldots b_i}$
6 **return** X

This code performs k multiplications, where $(k = \log e \leq \log n)$. Because $(B^e \leq 2^n)$, each multiplication is of two integers less than n bits long. Each multiplication thus takes $O(n^2)$ time, so the overall algorithm takes $O(n^2 \log n)$ time.

(c) [12 points] On inputs X and e, where X is an n-bit integer and $e \leq n$, give an $O(n^3 \log n)$-time algorithm to find B such that $X = B^e$. That is, find the e^{th} integer root of X, $B = \sqrt[e]{X}$. If no such B exists, return **null**.

Solution:

Initial Values for L and H:

$L \leftarrow 2$

$H \leftarrow \sqrt{X}$

ROOT(X, e, L, H):

$M \leftarrow (L + H)/2$

$Y \leftarrow$ EXPONENT(M, e)

if $(Y = X)$ **return** M

elseif $(L = H)$ **return null**

elseif $(Y > X)$ ROOT(X, e, L, M)

elseif $(Y < X)$ ROOT(X, e, M, H)

If B exists, it must be in the range $[2, \sqrt{X}]$. Suppose such a B exists. We can BINARY SEARCH for B by selecting a midpoint M of the range $[L, H]$ and computing M^e using our $O(n^2 \log n)$-time algorithm from part (b). If $(M^e < X)$, then clearly $B \in [M, H]$. Similarly, if $(M^e > X)$, then $B \in [L, M]$. If B does not exist, then the binary search will halt and return **null** after $(L = H)$.

This BINARY SEARCH is over a range with size at most $(\sqrt{X} \leq X)$. It will perform at most $(\log X = n)$ queries, which each take $O(n^2 \log n)$ time, for a total runtime of $O(n^3 \log n)$.

(d) [10 points] Given a n-bit integer X, give an $O(n^4 \log n)$-time algorithm to determine whether X is a perfect power. If X is not a perfect power, return **null**.

Solution:

PERFECT POWER SEARCH(X):

$n \leftarrow \log X$

for $e := 2$ **to** n:

BINARY-SEARCH for a $B \in [2, \sqrt{X}]$ such that $(X = B^e)$.

If successful, **return** (B, e)

return null.

We can exhaustively test every one of n possible e values using our method in part (c). Each call takes $O(n^3 \log n)$ time, so the entire search will take $O(n^4 \log n)$ total time.

SCRATCH PAPER

DETERMINISTICSELECT(A, n, i)

1 Divide the elements of the input array A into groups of 5 elements.
2 Find median of each group of 5 elements and put them in array B.
3 Call DETERMINISTICSELECT($B, n/5, n/10$) to find median of the medians, x.
4 Partition the input array around x into A_1 containing k elements $\leq x$
and A_2 containing $n - k - 1$ elements $\geq x$.
5 If $i = k + 1$, then return x.
6 Else if $i \leq k$, DETERMINISTICSELECT(A_1, k, i).
7 Else if $i > k$, DETERMINISTICSELECT($A_2, n - k - 1, i - (k + 1)$).