Quiz 1

- Do not open this quiz booklet until directed to do so. Read all the instructions on this page.
- When the quiz begins, write your name on every page of this quiz booklet.
- You have 120 minutes to earn 120 points. 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.
- This quiz is closed book. You may use **one** $8\frac{1}{2}'' \times 11''$ or A4 crib sheet (both sides). No calculators or programmable devices are permitted. No cell phones or other communications devices are permitted.
- Write your solutions in the space provided. If you need more space, write on the back of the sheet containing the problem. Pages may be separated for grading.
- Do not waste time and paper rederiving facts that we have studied. It is sufficient to cite known results.
- When asked for an algorithm, your algorithm should have the time complexity specified in the problem. If you cannot find such an algorithm, you will generally receive partial credit for a slower algorithm **if you analyze your algorithm correctly**.
- 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	Parts	Points	Grade	Grader
1	2	2		
2	9	18		
3	3	20		
4	1	20		
5	2	25		
6	3	20		
7	2	15		
Total		120		

Name: ______________________________

Friday	Zuzana	Debmalya	Ning	Matthew	Alina	Alex
Recitation:	**10 AM**	**11 AM**	**12 PM**	**1 PM**	**2 PM**	**3 PM**

Problem 1. What is Your Name? [2 points] (2 parts)

(a) [1 point] Flip back to the cover page. Write your name there.

(b) [1 point] Flip back to the cover page. Circle your recitation section.

Problem 2. True or False [18 points] (9 parts)

For each of the following questions, circle either T (True) or F (False). There is no penalty for incorrect answers.

(a) T F [2 points] Given two heaps with n elements each, it is possible to construct a single heap comprising all $2n$ elements in $O(n)$ time.

(b) T F [2 points] Building a heap with n elements can always be done in $O(n \log n)$ time.

(c) T F [2 points] Given a hash table of size n with n elements, using chaining, the minimum element can always be found in $O(1)$ time.

(d) T F [2 points] Running merge sort on an array of size n which is already correctly sorted takes $O(n)$ time.

(e) T F [2 points] We can always find the maximum in a min-heap in $O(\log n)$ time.

(f) T F [2 points] In a heap of depth d, there must be at least 2^d elements. (Assume the depth of the first element (or root) is zero).

(g) T F [2 points] Consider a family of hash functions $\{h_a\}$ hashing an r-bit number to an s-bit number ($r > s$). This family is defined by $h_a(x) = (x \operatorname{xor} a) \bmod 2^s$, where "xor" represents binary exclusive or and where a is a randomly chosen r-bit number.

This hash function family is 2-universal.

(h) T F [2 points] Inserting an element into a binary search tree of size n always takes $O(\log n)$ time.

(i) T F [2 points] Any two (possibly unbalanced) BSTs containing n elements each can be merged into a single balanced BST in $O(n)$ time.

Problem 3. Asymptotics & Recurrences [20 points] (3 parts)

(a) [10 points] Rank the following functions by *increasing* order of growth. That is, find any arrangement $g_1, g_2, g_3, g_4, g_5, g_6, g_7$ of the functions satisfying $g_1 = O(g_2)$, $g_2 = O(g_3)$, $g_3 = O(g_4)$, $g_4 = O(g_5)$, $g_5 = O(g_6)$, $g_6 = O(g_7)$.

$$f_1(n) = n^4 + \log n \qquad f_2(n) = n + \log^4 n \qquad f_3(n) = n \log n \qquad f_4(n) = \binom{n}{3}$$

$$f_5(n) = \binom{n}{n/2} \qquad f_6(n) = 2^n \qquad f_7(n) = n^{\log n}$$

(b) [5 points] Find an asymptotic solution of the following functional recurrence. Express your answer using Θ-notation.

$$T(n) = 9 \cdot T(n/3) + n^3$$

(c) [5 points] Find an asymptotic solution of the following functional recurrence. Express your answer using Θ-notation.

$$T(n) = 2 \cdot T(n/4) + \sqrt{n}$$

Problem 4. Median of Lists [20 points]

You are given two lists of integers, each of which is sorted in ascending order and each of which has length n. All integers in the two lists are different. You wish to find the n-th smallest element of the union of the two lists. (That is, if you concatenated the lists and sorted the resulting list in ascending order, the element which would be at the n-th position.) Present an algorithm to find this element in $O(\log n)$ time. You will receive half credit for an $O\left((\log n)^2\right)$-time algorithm.

Problem 5. Party Conversation [25 points] (2 parts)

You are attending a party with n other people. Each other person i arrives at the party at some time s_i and leaves the party at some time t_i (where $s_i < t_i$). Once a person leaves the party, they do not return.

Additionally, each person i has some *coolness* c_i. At all times during the party, you choose to talk to the coolest person currently at the party. (All coolness values are distinct.) If you are talking to someone, and someone else cooler arrives at the party, you leave your current conversation partner and talk to the new person. If the person you are talking to leaves the party, you go talk to the coolest person remaining at the party. (This might or might not be a person with whom you have already talked.)

You are the first to arrive at the party and the last to leave. Additionally, you are the most popular person at the party, so everyone wants to talk with you.

(a) [15 points] Describe a data structure which allows you to decide in $O(1)$ time to whom you should talk at any moment. You should be able to update this data structure in $O(\log n)$ time when someone arrives or leaves.

(b) [10 points] If you know in advance when each person will arrive at and depart from the party, describe an $O(n \log n)$-time algorithm to compute the total amount of time that you will spend talking with each person.

Problem 6. Space Efficient Sets [20 points] (3 parts)

In this problem, we will use hashing to test membership of an element in a set. Suppose we have a set S of n elements and we need to build a data structure that efficiently answers queries of the form: *is x in S?* Ben Bitdiddle comes up with the following idea: construct a hash table T of size $m > n$, where each entry of the hash table is a single bit. Now, use a hash function h to map each element in S to an index in the hash table, and set the corresponding bit of T to 1. Thus, $T[i] = 1$ iff $i = h(y)$ for some $y \in S$. His search strategy is simply to output *yes* iff $T[h(x)] = 1$.

(a) [7 points] Assume simple uniform hashing. If $x \notin S$, what is the probability that the algorithm answers *yes*? (This is called the probability of a *false positive*.)

(b) [3 points] If $x \in S$, what is the probability that the algorithm answers *no*? (This is called the probability of a *false negative*.)

(c) [10 points] Ben Bitdiddle changes the algorithm to use k hash functions $h_1, h_2, \ldots, h_k$ as follows: we now set $T[i] = 1$ iff $i = h_j(y)$ for some $y \in S, j \in \{1, 2, \ldots, k\}$ and answer *yes* iff $T[h_j(x)] = 1$ for all $j \in \{1, 2, \ldots, k\}$. Assuming that hash functions are independent and each hash function satisfies the assumption of part (a), what is the probability of a false positive now?

Problem 7. Modal Weights [15 points] (2 parts)

You are given a collection of n iron weights of varying masses. Some weights have the same mass and some weights have different masses. You would like to find the size of the largest possible sub-collection of weights that all have the same mass.

(a) [8 points] Assume that the weights are unlabeled and that your only tool for comparing them is a balance scale which will compare two weights. The balance will tell you which weight is more massive, or tell you that they have the same mass. If there are k distinct mass values represented by the n weights, give an $O(n \log k)$-time algorithm for finding the size (the number of weights) of the largest collection of equal weights.

(b) [7 points] Now assume that you find a digital scale, which can tell you the exact mass of a weight. Now give an $O(n)$-time algorithm to accomplish the same task.

SCRATCH PAPER

SCRATCH PAPER