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 writing an algorithm, a **clear** description in English will suffice. Pseudo-code is not required.
- When asked for an algorithm, your algorithm should have the time complexity specified in the problem with a correct analysis. 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	3	20		
3	9	18		
4	1	20		
5	3	20		
6	5	20		
7	1	20		
Total		120		

Name: ____________________

Friday Recitation:	Aleksander **11 AM**	Arnab **12 PM**	Alina **3 PM**	Matthew **4 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. Asymptotics & Recurrences [20 points] (3 parts)

(a) [10 points] Rank the following functions by *increasing* order of growth. That is, find any arrangement $g_1, g_2, g_3, g_4, g_5, g_6, g_7, g_8$ of the functions satisfying $g_1 = O(g_2)$, $g_2 = O(g_3)$, $g_3 = O(g_4)$, $g_4 = O(g_5)$, $g_5 = O(g_6)$, $g_6 = O(g_7)$, $g_7 = O(g_8)$.

$$f_1(n) = n\log^2 n \qquad f_2(n) = n + \sqrt{n}\log^4 n \qquad f_3(n) = \binom{n}{4} \qquad f_4(n) = \binom{n}{n/3}$$

$$f_5(n) = \binom{n}{n-2} \qquad f_6(n) = 2^{\log^2 n} \qquad f_7(n) = n^{\sqrt{\log n}} \qquad f_8(n) = n^3\binom{n}{2}$$

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

$$T(n) = 16 \cdot T(n/4) + n^2 \log^3 n$$

(c) [5 points] Find an asymptotic solution of the following recurrence. Express your answer using Θ-notation, and give a brief justification. (Note that $n^{\frac{1}{\log n}} = 1$.)

$$T(n) = T(\sqrt{n}) + 1$$

Problem 3. True/False [18 points] (9 parts)

Circle (T)rue or (F)alse. You don't need to justify your choice.

(a) T F [2 points] Inserting into an AVL tree can take $o(\log n)$ time.

(b) T F [2 points] If you know the numbers stored in a BST and you know the structure of the tree, you can determine the value stored in each node.

(c) T F [2 points] In max-heaps, the operations insert, max-heapify, find-max, and find-min all take $O(\log n)$ time.

(d) T F [2 points] When you double the size of a hash table, you can keep using the same hash function.

(e) T F [2 points] We can sort 7 numbers with 10 comparisons.

(f) T F [2 points] Merge sort can be implemented to be stable.

(g) T F [2 points] If we were to extend our $O(n)$ 2D peak finding algorithm to four dimensions, it would take $O(n^3)$ time.

(h) T F [2 points] A $\Theta(n^2)$ algorithm always takes longer to run than a $\Theta(\log n)$ algorithm.

(i) T F [2 points] Assume it takes $\Theta(k)$ time to hash a string of length k. We have a string of length n, and we want to find the hash values of all its substrings that have length k using a division hash. We can do this in $O(n)$ time.

Problem 4. Runway Reservation Modifications [20 points] (1 part)

Recall the Runway Reservation system used in Problem Set 2, where we wanted to schedule flight landing times so that each scheduled landing was at least 3 minutes away from the others. We would like to expand the functionality of this system to deal with more types of requests.

Suppose that we have a valid schedule setup already, but as the flights are landing according to this schedule, at some time t_1 there is a plane that requests an immediate *emergency* landing. Of course, we want to accommodate such a request immediately, thus we schedule this plane to land at time t_1. However, this landing might cause collisions with the flights that are scheduled to land after t_1. (It might also cause collision with the flight scheduled to land just before t_1, but since this flight already landed, there is nothing we can do about this and we ignore this fact.) To adjust the schedule to this disruption, we want to shift (if necessary) the landing times of the flights scheduled to land after t_1 to make sure that each subsequent landing happens at least 3 minutes after the previous one. (Note that we want to preserve the order of landing times from the original schedule.)

For example, assuming a window of 3 minutes with flights at times 28, 31, 34, 37, 40, 46, 49, 53, 57, 60, attempting to insert an emergency flight at time 32 would cause a new set of flights at times 28, 31, 32, 35, 38, 41, 46, 49, 53, 57, 60. In this case we would say three flights needed adjusting - 34, 37, and 40 - to accommodate the emergency landing.

Give an algorithm to find the minimum number of adjustments needed. This algorithm should find only the number of flight times requiring adjusting; it does not need to perform the flight updates. Full points will be given for an algorithm that runs in time polynomial in $\log n$ i.e. in time being $O(\log^c n)$ for some constant c. More credit will be given to more efficient solutions, as long as their runtime is correctly analyzed. You may use data structure augmentations provided that you explain the augmentation. Its maintenance may not increase the asymptotic running time of other operations but you are not required to prove this.

Problem 5. Computing Fibonacci numbers [20 points] (3 parts)

The Fibonacci numbers are defined by the following recurrence: $F_0 = 0, F_1 = 1, F_i = F_{i-1} + F_{i-2}$ for $i \geq 2$, yielding the sequence $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$ There is a closed-form formula for F_n given by $F_n = \frac{\phi^n - (1-\phi)^n}{\sqrt{5}}$, where $\phi = (1 + \sqrt{5})/2 \approx 1.618$ is the golden ratio. However, this formula isn't practical for computing the exact value of F_n as it would require increasing precision on $\sqrt{5}$ and ϕ as n increases. In this problem we are interested in obtaining practical algorithms for computing the n^{th} Fibonacci number F_n for any given n. Assume that the cost of adding, subtracting, or multiplying two integers is $O(1)$, independent of the size of the integers we are dealing with.

(a) [5 points] From the recurrence definition of the Fibonacci sequence, one can use the following simple recursive algorithm:

```
$FIB1(n)$:
  if $n \leq 1$ then
    return $n$
  else
    $x \leftarrow FIB1(n-1)$
    $y \leftarrow FIB1(n-2)$
    return $x + y$
  end if
```

Give the running time of this algorithm. Express your answer using Θ-notation.

(b) [5 points] Give an algorithm that computes F_n in $\Theta(n)$ and justify its running time.

(c) [10 points] Consider the matrix $A = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$. Show that for $i \geq 1$, $A^i = \begin{pmatrix} F_{i-1} & F_i \\ F_i & F_{i+1} \end{pmatrix}$. From the fact that $A^{2i} = (A^i)^2$, and $A^{2i+1} = A*(A^i)^2$, use divide and conquer to show that A^n can be calculated in time $\Theta(\log n)$. Conclude by giving a $\Theta(\log n)$ algorithm for computing F_n.

Problem 6. One Hash, Two Hash [20 points] (4 parts)

We talked in class about two methods for dealing with collisions: chaining and linear probing. Cornelius Beefe decided to come up with his own method. He creates two hash tables T_1 and T_2, each of size m, and two different hash functions h_1 and h_2 and decides to use T_2 to resolve collisions in T_1. Specifically, given an element x, he first calculates $h_1(x)$ and tries to insert x into T_1. If there is a collision, he calculates $h_2(x)$ and inserts x into T_2.

(a) [4 points] Assume T_2 uses chaining to deal with collisions. Given an element y, give an algorithm for deciding whether or not y is in the hash table.

(b) [2 points] Cornelius tries using the following hash functions for each table:

1) $h_1(x) = (a_1)^x \bmod m$ for table T_1
2) $h_2(x) = (a_2)^x \bmod m$ for table T_2

He first tries $a_1 = a_2$ and uses chaining in table T_2. Is this likely to result in fewer collisions than if he had just used one hash table of size m with chaining?

(c) [4 points] Assume $m = 21$. Circle the set of values that will result in the fewest collisions in T_2 and explain why you chose it.

(1) $a_1 = 9, a_2 = 5$

(2) $a_1 = 5, a_2 = 7$

(3) $a_1 = 4, a_2 = 16$

(4) $a_1 = 13, a_2 = 11$

(d) [10 points] Consider the case in which T_2 uses linear probing to deal with collisions. Also, assume that both h_1 and h_2 satisfy the simple uniform hashing assumption. We insert 4 elements. What is the probability that while inserting the fourth element there are at least two collisions? (In this count we include the collisions that occur in table T_1.)

Problem 7. Extreme Temperatures [20 points] (1 part) Professor Daskalakis is interested in studying extreme temperatures on the Arctic Cap. He placed temperature-measuring devices at m locations, and programmed each of these devices to record the temperature of the corresponding location at noon of every day, for a period of n days. Moreover, using techniques that he learned while preparing the Heapsort lecture, he decided to program each device to store the recorded temperatures in a max-heap. To cut a long story short, Prof. Daskalakis now has m devices that he collected from the Arctic Cap, each of which contains in its hard-drive a max-heap of n elements. He now wants to compute the ℓ largest temperatures that were recorded by any device, e.g., if $m = 2$, $n = 5$, $\ell = 5$, and the two devices recorded temperatures $(-10, -20, -5, -34, -7)$ and $(-13, -19, -2, -3, -4)$ respectively, the desired output would be $(-2, -3, -4, -5, -7)$. Can you help your professor find the ℓ largest elements in $O(m + \ell \log \ell)$ time? Partial credit will be given for less efficient algorithms, as long as the run-time analysis is correct.

SCRATCH PAPER

SCRATCH PAPER