# 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 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 | 2 | 20 | | |
| 5 | 2 | 20 | | |
| 6 | 3 | 20 | | |
| 7 | 2 | 20 | | |
| Total | | 120 | | |

Name: _____

| Wed/Fri Recitation: | Ying | Kevin | Sarah | Yafim | Victor |
|---|---|---|---|---|---|
| | **10, 11 AM** | **11 AM** | **12, 1 PM** | **12 PM** | **2, 3 PM** |

**Problem 1.** [2 points] Write your name on top of each page.

**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 $g_1 = O(g_2)$, $g_2 = O(g_3)$, $g_3 = O(g_4)$, $g_4 = O(g_5)$, $g_5 = O(g_6)$, $g_6 = O(g_7)$, $g_7 = O(g_8)$.

$$f_1(n) = n^{\pi} \qquad f_2(n) = \pi^n \qquad f_3(n) = \binom{n}{5} \qquad f_4(n) = \sqrt{2^{\sqrt{n}}}$$

$$f_5(n) = \binom{n}{n-4} \qquad f_6(n) = 2^{\log^4 n} \qquad f_7(n) = n^{5(\log n)^2} \qquad f_8(n) = n^4\binom{n}{4}$$

**(b)** [5 points] Find a solution to the recurrence $T(n) = T(\frac{n}{3}) + T(\frac{2n}{3}) + \Theta(n)$.

**(c)** [5 points] Find an asymptotic solution of the following recurrence. Express your answer using $\Theta$-notation, and give a brief justification.

$$T(n) = \log n + T\left(\sqrt{n}\right)$$

**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] Binary insertion sorting (insertion sort that uses binary search to find each insertion point) requires $O(n \log n)$ total operations.

**(b)** **T** F [2 points] In the merge-sort execution tree, roughly the same amount of work is done at each level of the tree.

**(c)** T **F** [2 points] In a BST, we can find the next smallest element to a given element in $O(1)$ time.

**(d)** T **F** [2 points] In an AVL tree, during the insert operation there are at most two rotations needed.

**(e)** **T** F [2 points] Counting sort is a stable, in-place sorting algorithm.

**(f)** T **F** [2 points] In a min-heap, the next largest element of any element can be found in $O(\log n)$ time.

**(g)** T **F** [2 points] The multiplication method satisfies the simple uniform hashing assumption.

**(h)** T **F** [2 points] Double hashing satisfies the uniform hashing assumption.

**(i)** **T** F [2 points] Python generators can be used to iterate over potentially infinite countable sets with $O(1)$ memory.

**Problem 4. Peak Finding (again!)** [20 points] (2 parts)

When Alyssa P. Hacker did the first 6.006 problem set this semester, she didn't particularly like any of the 2-D peak-finding algorithms. A peak is defined as any location that has a value at least as large as all four of its neighbors.

Alyssa is excited about the following algorithm:

1. Examine all of the values in the first, middle, and last columns of the matrix to find the maximum location $\ell$.
2. If $\ell$ is a peak within the current subproblem, return it. Otherwise, it must have a neighbor $p$ that is strictly greater.
3. If $p$ lies to the left of the central column, restrict the problem matrix to the left half of the matrix, including the first and middle columns. If $p$ lies to the right of the central column, restrict the problem matrix to the right half of the matrix, including the middle and last columns.
4. Repeat steps 1 through 3 looking at the first, middle, and last **rows**.
5. Repeat steps 1 through 4 until a peak is found.

Consider the $5 \times 5$ example depicted below. On this example, the algorithm initially examines the first, third, and fifth columns, and finds the maximum in all three. In this case, the maximum is the number $4$. The number $4$ is not a peak, due to its neighbor $5$.

| | | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 4 | 5 | 0 | 0 | 0 |
| 0 | 0 | 1 | 2 | 0 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 6 | 0 | 7 | 0 |

| | | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 4 | 5 | 0 | 0 | 0 |
| 0 | 0 | 1 | 2 | 0 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 6 | 0 | 7 | 0 |

| | | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| (4) | 5 | 0 | 0 | 0 |
| 0 | 0 | 1 | 2 | 0 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 6 | 0 | 7 | 0 |

The number $5$ is to the left of the middle column, so we restrict our view to just the left half of the matrix. (Note that we include both the first and middle columns.) Because we examined columns in the previous step, we now examine the first, middle, and last rows of the submatrix. The largest value still visible in those rows is $6$, which is a peak within the subproblem. Hence, the algorithm will find the peak $6$.

| | | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 4 | 5 | 0 | 0 | 0 |
| 0 | 0 | 1 | 2 | 0 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 6 | 0 | 7 | 0 |

| | | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 4 | 5 | 0 | 0 | 0 |
| 0 | 0 | 1 | 2 | 0 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 6 | 0 | 7 | 0 |

| | | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 4 | 5 | 0 | 0 | 0 |
| 0 | 0 | 1 | 2 | 0 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | (6) | 0 | 7 | 0 |

**(a)** [5 points] What is the worst-case runtime of Alyssa's algorithm on an $m \times n$ matrix ($m$ rows, $n$ columns), in big-$\Theta$ notation? Give a brief justification for your answer.

**(b)** [15 points] Does Alyssa's algorithm return a peak in all cases? If so, give a short proof of correctness. Otherwise, provide a counterexample for the algorithm.

**Problem 5. Who Let The Zombies Out?** [20 points] (2 parts)

In an attempt to take over Earth, evil aliens have contaminated certain water supplies with a virus that transforms humans into flesh-craving zombies. To track down the aliens, the Center for Disease Control needs to determine the epicenters of the outbreak—which water supplies have been contaminated. There are $n$ potentially infected cities $C = \{c_1, c_2, \ldots, c_n\}$, but the FBI is certain that only $k$ cities have contaminated water supplies.

Unfortunately, the only known test to determine the contamination of a city's water supply is to serve some of that water to a human and see whether they turn ravenous. Several brave volunteers have offered to undergo such an experiment, but they are only willing to try their luck once. Each volunteer is willing to drink a single glass of water that mixes together samples of water from any subset $C' \subseteq C$ of the $n$ cities, which reveals whether at least one city in $C'$ had contaminated water.

Your goal is to use the fewest possible experiments (volunteers) in order to determine, for each city $c_i$, whether its water was contaminated, under the assumption that exactly $k$ cities have contaminated water. You can design each experiment based on the results of all preceding experiments.

**(a)** [10 points] You observe that, as in the comparison model, any algorithm can be viewed as a decision tree where a node corresponds to an experiment with two outcomes (contaminated or not) and thus two children. Prove a lower bound of $\Omega(k \lg \frac{n}{k})$ on the number of experiments that must be done to save the world. Assume that $\lg x! \sim x \lg x$ and that $\lg(n-k) \sim \lg n$ (which is reasonable when $k < 0.99n$).


**(b)** [10 points] Save the world by designing an algorithm to determine which $k$ of the $n$ cities have contaminated water supplies using $O(k \lg n)$ experiments. Describe and analyze your algorithm.


**Problem 6. Shopping Madness** [20 points] (3 parts)

Ben Bitdiddle was peer-pressured into signing up for the tryouts in a shopping reality TV show, and he needs your help to make it past the first round. In order to qualify, Ben must browse a store's inventory, which has $N$ items with different positive prices $P[1], P[2], \ldots, P[N]$, and the challenge is to spend exactly $S$ dollars on exactly $K$ items, where $K$ is a small even integer. Ben can buy the same item multiple times. For example, 3 brooms and 2 wizard hats add up to 5 items.

In your solutions below, you may use a subroutine MULTISETS$(k, \mathbb{T})$ which iterates over all the $k$-element multisets (like subsets, except the same elements can show up multiple times) of a set $\mathbb{T}$, in time $O(k \cdot |\mathbb{T}|^k)$, using $O(k)$ total space. Note that if your code holds onto the results of MULTISETS, it may end up using more than $O(k)$ space.

**(a)** [5 points] Write pseudo-code for a data structure that supports the following two operations.

INIT$(N, K, P)$ — preprocesses the $P[1 \ldots N]$ array of prices, in $O(K \cdot N^K)$ expected time, using $O(K \cdot N^K)$ space, to be able to answer the query below.

BAG$(S)$ — in $O(1)$ expected time, determines whether $K$ of the items have prices summing to $S$, and if so, returns $K$ indices $b_1, b_2, \ldots, b_K$ such that $S = \sum_{i=1}^{K} P[b_i]$.

**(b)** [10 points] Write pseudo-code for a function PWN-CONTEST$(N, S, K, P)$ that determines whether $K$ of the items have prices summing to $S$, and if so, returns $K$ indices $b_1, b_2, \ldots, b_K$ such that $S = \sum_{i=1}^{K} P[b_i]$. Unlike part (a), PWN-CONTEST should run in $O(K \cdot N^{K/2})$ and use $O(K \cdot N^{K/2})$ space.

**(c)** [5 points] Analyze the running time of your pseudo-code for the previous part.

**Problem 7. When I Was Your Age...** [20 points] (2 parts)

In order to design a new joke for your standup comedy routine, you've collected $n$ distinct measurements into an array $A[1 \ldots n]$, where $A[i]$ represents a measurement at time $i$. Your goal is to find the longest timespan $i \ldots j$, i.e., maximize $j - i$, such that $A[i] < A[j]$. [1] Note that the values in between $A[i]$ and $A[j]$ do not matter. As an example, consider the following array $A[1 \ldots 7]$:

$$A[1] = 14 \quad A[2] = 6 \quad A[3] = 8 \quad A[4] = 1 \quad A[5] = 12 \quad A[6] = 7 \quad A[7] = 5$$

Your algorithm should return a span of 4 since $A[2] = 6$ and $A[6] = 7$. The next biggest span is $A[4] = 1$ to $A[7] = 5$.

**(a)** [5 points] Give an $O(n)$-time algorithm to compute the minimums of the prefix $A[1 \ldots k]$ for each $k$, and store in $MA[k]$: $MA[k] = \min_{i=1}^{k} A[i]$.

**(b)** [15 points] Using the $MA[i]$ computed above, give an $O(n \log n)$-time algorithm to maximize $j - i$ subject to $A[i] < A[j]$.
*Hint*: The $MA$ is a sorted array.

[1] The joke could be along these lines: "You thought time $j$ was bad with $A[j]$? Back in time $i$, we only had $A[i]$!"

6.006 Introduction to Algorithms
Fall 2011