Quiz 1 Solutions

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

$$f_1(n) = n^{\pi} \qquad f_2(n) = \pi^n \qquad f_3(n) = \binom{n}{5} \qquad f_4(n) = \sqrt{2^{\sqrt{n}}}$$
$$f_5(n) = \binom{n}{n-4} \qquad f_6(n) = 2^{\log^4 n} \qquad f_7(n) = n^{5(\log n)^2} \qquad f_8(n) = n^4\binom{n}{4}$$

Solution: $f_1(n), f_5(n), f_3(n), f_8(n), f_7(n), f_6(n), f_4(n), f_2(n)$

Scoring: We computed the score for this problem as $\text{ROUND}(10 \cdot \frac{L-1}{N-1})$, where N is the number of functions ($N = 8$ for this instance) and L is the length of the longest common subsequence between our solution and the student's answer.

The intuition behind the longest common subsequence is that we want to cross out as few functions as possible from a student's answer, such that the remaining functions will be correctly ordered. Who said the 6.006 staff isn't nice?

We used $\frac{L-1}{N-1}$ to normalize the scores, because a completely wrong answer will still share a common subsequence of length 1 with the correct answer.

The longest common subsequence can be computed using Dynamic Programming, which will be taught in 6.006 towards the end of the term.

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

Solution: Draw recursion tree. At each level, do $\Theta(n)$ work. Number of levels is $\log_{3/2} n = \Theta(\lg n)$, so guess $T(n) = \Theta(n \lg n)$ and use the substitution method to verify guess.

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

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

Solution: $T(n) = \Theta(\log n)$.

To see this, note that if we expand out $T(n)$ by continually replacing $T(n)$ with its formula, we get:

$$\begin{aligned} T(n) &= \log n + \log \sqrt{n} + \log \sqrt{\sqrt{n}} + \log \sqrt{\sqrt{\sqrt{n}}} + \ldots \\ &= \log n + \tfrac{1}{2} \log n + \tfrac{1}{2} \log \sqrt{n} + \tfrac{1}{2} \log \sqrt{\sqrt{n}} + \ldots \\ &= \log n + \tfrac{1}{2} \log n + \tfrac{1}{4} \log n + \tfrac{1}{8} \log n + \ldots \\ &= \Theta(\log n) \end{aligned}$$

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.

Solution: False. While binary insertion sorting improves the time it takes to find the right position for the next element being inserted, it may still take $O(n)$ time to perform the swaps necessary to shift it into place. This results in an $O(n^2)$ running time, the same as that of insertion sort.

(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.

Solution: True. At the top level, roughly n work is done to merge all n elements. At the next level, there are two branches, each doing roughly $n/2$ work to merge $n/2$ elements. In total, roughly n work is done on that level. This pattern continues on through to the leaves, where a constant amount of work is done on n leaves, resulting in roughly n work being done on the leaf level, as well.

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

Solution: False. Finding the next smallest element, the predecessor, may require traveling down the height of the tree, making the running time $O(h)$.

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

Solution: True. The AVL property is restored on every operation. Therefore, inserting another item will require at most two rotations to restore the balance.

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

Solution: False. Counting sort is stable. It is not in-place, however, since we must make additional space to store the counts of the various elements. This space requirement grows as the size of the input increases. Additionally, we have to make a separate output array to produce the answer using counting sort.

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

Solution: **False.** A min-heap cannot provide the next largest element in $O(\log n)$ time. To find the next largest element, we need to do a linear, $O(n)$, search through the heap's array.

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

Solution: **False.** We don't really know of hash functions that satisfy the simple uniform hashing assumption.

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

Solution: **False.** The notes state that double hashing 'comes close.' Double hashing only provides n^2 permutations, not $n!$.

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

Solution: **True.** Python generators do not require the whole set to reside in memory to iterate over it, making this assertion true.

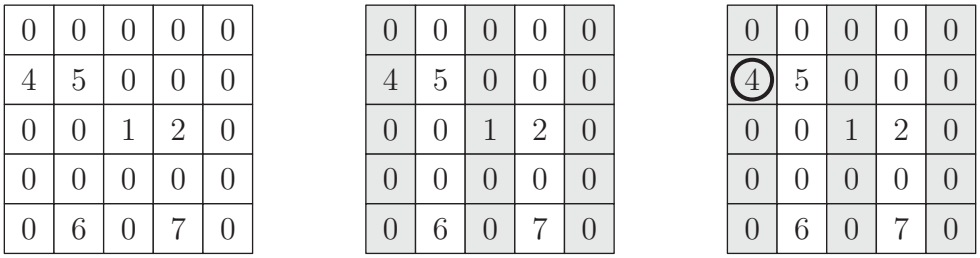
Problem 4. Peak Finding (again!) [20 points] (2 parts)

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.

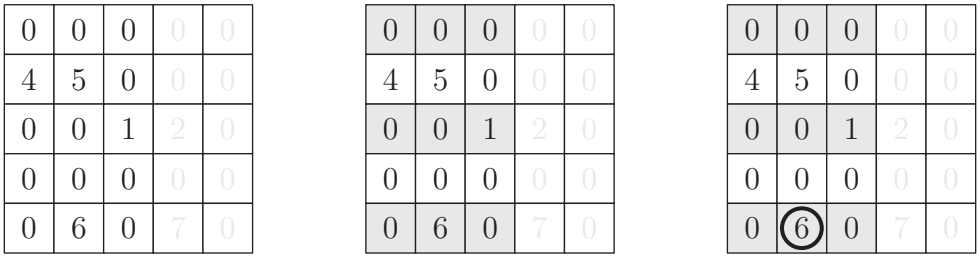
Alyssa is excited about the following algorithm:

1. Examine all of the values in the first, middle, and last columns of the matrix to find the maximum location ℓ.
2. If ℓ 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 5×5 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.



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.



(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-Θ notation? Give a brief justification for your answer.

Solution: Let $S(m, n)$ be the runtime of the algorithm when run on an $m \times n$ matrix starting with columns. Let $T(m, n)$ be the runtime of the algorithm when run on an $m \times n$ matrix starting with rows. Then $S(m, n) \leq T(m, n/2 + 1) + \Theta(m)$ and $T(m, n) \leq S(m/2+1, n)+\Theta(n)$. Hence, $S(m, n) \leq \Theta(m+n)+S(m/2+1, n/2+1)$. When we resolve this recurrence relation, we get $S(m, n) = O(m + n)$. In the case of a square $n \times n$ matrix, we get an asymptotic runtime of $\Theta(n)$.

(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.

Solution: The following is an example of a matrix where the algorithm will return the wrong value:

0	0	0	0	0
4	5	0	0	0
0	0	1	2	0
0	0	0	0	0
0	0	0	7	0

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 $k < 0.99n$).

Solution: The number of possible outcomes—which cities are contaminated—is $\binom{n}{k}$. Thus any decision tree must have at least $\binom{n}{k}$ leaves. Because a decision tree is binary, it must therefore have height at least

$$\lg \binom{n}{k} = \lg \frac{n!}{k!(n-k)!},$$

which by the first assumption is

$$\sim n \lg n - k \lg k - (n-k) \lg(n-k) = n[\lg n - \lg(n-k)] + k[\lg(n-k) - \lg k],$$

which by the second assumption is

$$\sim k[\lg n - \lg k] = k \lg \frac{n}{k},$$

which is our desired lower bound.

(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.

Solution: The algorithm is based on divide and conquer: divide the n cities into two groups of size $n/2$; test each group for contamination (using two experiments); and recurse into each contaminated group. The recursion tree has exactly k leaves, and the height of the tree is at most $\lg n$, so the number of internal nodes leading to the leaves is at most $k \lg n$. Each internal node costs 2, for a total cost of $O(k \lg n)$.

In fact, it is possible to prove an $O(k \lg \frac{n}{k})$ bound on the same algorithm. To minimize the number of shared nodes among the k paths from root to leaves, the worst case is when the recursion tree branches for the first $\lg k$ levels (to get enough leaves), and then has k straight paths for the number of levels: $\lg n - \lg k = \lg \frac{n}{k}$. There are $O(k)$ nodes in the top branching, and $O(k \lg \frac{n}{k})$ nodes in the bottom paths.

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.

In your solutions below, you may use a subroutine SUBSETS$(k, \mathbb{T})$ which iterates over all the k-element subsets 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 SUBSETS, 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 $S = \sum_{i=1}^{K} P[b_i]$.

Solution:

INIT(N, K, p)

```
1  h ← empty hash table
2  for c ← SUBSETS(K, {1 ... N})
3       do s ← Σ_{i=1}^{K} P_{c_i}
4          h[s] ← c
```

BAG(S)

```
1  if S ∈ h
2     then return h[S]
3     else  return NIL
```

(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.

Solution:

PWN-CONTEST(N, S, K, p)

```
1  h ← empty hash table
2  for c ← SUBSETS(K/2, {1...N})
3        do s ← Σ_{i=1}^{K/2} P_{c_i}
4           h[s] ← c
5  for c ← SUBSETS(K/2, {1...N})
6        do s ← S − Σ_{i=1}^{K/2} P_{c_i}
7           if s ∈ h
8             then return c + h[s]
9  return NIL
```

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

Solution: The following table shows a line-by-line analysis of our pseudo-code.

Line	Time	Number of iterations	Total time
1	$O(1)$	1	$O(1)$
2	$K \cdot N^{K/2}$	1	$K \cdot N^{K/2}$
3	$O(K)$	$N^{K/2}$	$K \cdot N^{K/2}$
4	$O(K)$	$N^{K/2}$	$K \cdot N^{K/2}$
5	$K \cdot N^{K/2}$	1	$K \cdot N^{K/2}$
6	$O(K)$	$N^{K/2}$	$K \cdot N^{K/2}$
7	$O(1)$	$N^{K/2}$	$N^{K/2}$
8	$O(K)$	1	$O(K)$
9	$O(1)$	1	$O(1)$

The total running time is the maximum in the "Total time" column, which is $K \cdot N^{K/2}$, as requested.

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 $MA[k]$: $MA[k] = \min_{i=1}^{k} A[i]$.

Solution: $MA[i]$ can be computed incrementally. Initially, $MA[1] = A[1]$. $MA[j] = \min(A[j], MA[j-1])$. This takes $O(n)$ time.

(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.

Solution: Consider a single element $A[j]$. If we have $MA[1 \ldots j-1]$ we want to find an index i such that $MA[i] < A[j]$ but $MA[i-1] \geq A[j]$. This implies that $MA[i] = A[i]$ is the unique minimum element of $A[1 \ldots i]$. This gives us an $A[i],\ A[j]$ pair and we can compute $j - i$ for this pair.

We do a binary search over the indices $[1, j-1]$. We start with $j/2$, and we test whether $MA[j/2] < A[j]$ or not. If $MA[j/2]$ is less than $A[j]$, we recurse on $MA[1 \ldots (j/2 - 1)]$. If $MA[j/2]$ is larger than $A[j]$, we recurse on $MA[j/2 + 1 \ldots j - 1]$. We halve the number of possible indices for i each time, until we find the right i for this j. This takes $O(\log n)$ time. We do this for each j – hence the $O(n \log n)$. After we have the right i, j pairs, we pick the one that maximizes $j - i$.

[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