Final Exam Solutions

- 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 the top of *every* page of this quiz booklet.
- You have 180 minutes to earn a maximum of 180 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.
- **You are allowed three double-sided letter-sized sheets with your own notes**. No calculators, cell phones, or other programmable or communication devices are permitted.
- Write your solutions in the space provided. If you need more space, write on the scratch pages at the end of the exam, and refer to the scratch pages in the solution space provided. Pages will be scanned and separated for grading.
- Do not waste time and paper rederiving facts that we have studied. Simply cite them.
- When writing an algorithm, a **clear** description in English will suffice. Pseudo-code is not required. But be sure to prove the required bound on **running time** and explain **correctness**. Even if your running time is slower than the requested bound, you will likely receive partial credit if your algorithm and analysis are correct.
- If you give a correct answer *and* an incorrect answer to the same problem (without labeling the latter as incorrect), you will receive partial but not full credit (approximately the average of the answers).
- **Pay close attention to the instructions for each problem**. Depending on the problem, partial credit may be awarded for incomplete answers.

Problem	Parts	Points
0 : What is Your Name?	2	2
1 : True or False, and Justify	10	50
2 : Fast & Furious Coding	4	32
3 : Shortcut through the Circle of Life	1	10
4 : I'm All About That Base	3	15
5 : Escape the Graph	1	10
6 : Groundhog Graphs	1	10
7 : Legal Need For Speed	2	18
8 : Squeezing into the Page Limit	3	18
9 : Love Triangles	3	15
Total		180

Name: _____

Problem 0. [2 points] **What is Your Name?** (2 parts)

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

Solution: Algorithmee McAlgorithmsMan

(b) [1 point] Write your name on top of each page, including the scratch paper!

Solution: Done!

Problem 1. [50 points] **True or False, and Justify** (10 parts)

Circle T or F to indicate whether the statement is true or false, and explain your answer: if true, sketch a proof; and if false, give a counterexample. Your justification is worth more than your true or false designation.

(a) T F [5 points] The recurrence $T(n) = 2T(n/2) + O(n \lg n)$ solves to $O(n \lg n)$ by the Master Theorem.

Solution: False. We can use the recursion tree method to verify that the solution to the recurrence is $O(n \lg^2 n)$.

(b) T F [5 points] As implemented in class, heapsort runs in $O(n)$ time on a sorted array of n elements.

Solution: False. Every call to EXTRACT-MAX takes $\Theta(\lg n)$, and since heapsort requires $\Theta(n)$ EXTRACT-MAX calls, its runtime complexity is $\Theta(n \lg n)$.

(c) T F [5 points] Every algorithm that finds a peak (local maximum) in a 1D array A of n elements, in the comparison model, uses $\Omega(\lg n)$ comparisons in the worst case.

Solution: True. This is identical to the problem from Quiz 1. For more information, take a look at Quiz 1 solutions.

(d) T F [5 points] While running depth-first search, if the discover time of a node u is before the discover time of node v, then u must be an ancestor of v.

Solution: False. A simple counterexample is to have a directed graph over three nodes: a root r with two children u and v. If we start depth-first search from the root r, then neither u nor v is an ancestor of the other even though the discover time for one is smaller than the other.

(e) T F [5 points] For all weighted directed graphs $G = (V, E, w)$, Dijkstra's algorithm is asymptotically faster than an $O(V^{1.3})$-time algorithm.

Solution: False. Consider a complete graph on V nodes. Then Dijkstra's algorithm takes $\Omega(V^2)$ time.

(f) T F [5 points] To compute $\sqrt[3]{6.006}$ using gradient descent, the method we learned in class would use the update equation $x_{i+1} := x_i^3 - 6.006$.

Solution: False.
We are interested in obtaining a zero of the function

$$g(x) = x^3 - 6.006,$$

and in order to use gradient descent for this problem, we need to compute

$$f(x) = \int g(x)\, dx = \frac{x^4}{4} - 6.006x + C.$$

Recall that the update equation for gradient descent is $x_{i+1} = x_i - \eta \nabla f(x_i)$. By substituting in the particular $f(x)$, the update equation becomes

$$x_{i+1} = x_i - \eta(x_i^3 + 6.006),$$

which does not match the suggested update equation for any constant learning rate η due to the absence of the x_i term in the proposed update equation.

(g) T F [5 points] The number of intersection points among n lines is $O(n^2)$.

Solution: True. Every pair of lines has at most one intersection point. There are $\binom{n}{2}$ pairs of lines, and hence are at most $O(n^2)$ intersection points.
Note: It is also possible to claim that the answer is false by arguing that the number of intersection points of two overlapping lines is not finite. Those answers were also accepted.

(h) T F [5 points] Given n line segments in the plane, we can determine whether any of them cross each other in $O(n \log n)$ time.

Solution: True. Use the BENTLEY-OTTMANN algorithm presented in class that finds all k intersection points in $O((n + k) \lg(n + k))$ time, but stop immediately after finding a single intersection point, which leads to an $O(n \lg n)$-time algorithm.

(i) T F [5 points] The following problem is in P: given a weighted directed graph $G = (V, E, w)$, vertices $s, t \in G$, and a number ℓ, decide whether the shortest-path weight in G from s to t is $\leq \ell$.

Solution: True. We can find the shortest path between s and t by running BELLMAN-FORD, which is a polynomial-time algorithm.

(j) T F [5 points] The following problem is in NP: given a weighted directed graph $G = (V, E, w)$, vertices $s, t \in G$, and a number ℓ, decide whether there is a **simple** path in G from s to t of weight $\geq \ell$.

Solution: True. Given a path between s and t it is easy to verify (i.e. takes polynomial time) that it is both simple, and has weight $\geq \ell$.

Problem 2. [32 points] **Fast & Furious Coding** (4 parts)

Prof. Toretto is new to Python and its cost model, but needs to solve some important algorithmic problems for his upcoming fateful car heist. He has written **correct** code, but it's running too slow. For each of the following implementations, answer the following questions:

1. What is the asymptotic running time of the function right now? (Briefly justify your answer. If it's exponential, you don't need to give a precise bound—"exponential" will suffice. If it's polynomial, give a precise Θ bound.)
2. Find a small fix to make the function more efficient but still correct. (Your fix should give the optimal possible running time, and should involve changing only a few lines of code. **Just describe your changes to the algorithm in English; don't waste time by copying/writing code**. This part is worth the most points.)
3. What is the new asymptotic running time of the function, after your fix? (Briefly justify your answer. Give a precise Θ bound. It should be polynomial.)

Assume that input A is an array of n integers, and input k is a nonnegative integer.

(a) [8 points]

```
1 # Checks whether two (not necessarily distinct)
2 # elements in A sum to k.
3 def check_sum_to_k(A, k):
4     for x in A:
5         if k - x in A:
6             return True
7     return False
```

Solution:

1. $\Theta(n^2)$

2. Before line 4, add `A = set(A)`. This makes the `in` lookups cost constant expected time instead of linear time as in an array.

3. $O(n)$

(b) [8 points]

```
1  # Extracts the k smallest elements of A
2  # (possibly destroying A).
3  import heapq
4  def get_min_k(A, k):
5      ans = []
6      for i in range(k):
7          heapq.heapify(A)
8          # A.pop(0) removes the element at index 0,
9          # and returns that element. Because A is a heap,
10         # the first element must be the minimum.
11         x = A.pop(0)
12         ans.append(x)
13     return ans
```

Solution:

1. $O(n^2)$

2. Move `heapq.heapify(A)` up one line (outside the `for` loop). Replace line 11 (`x = A.pop(0)`) by `x = heapq.heappop(A)`.

3. $O(n + k \lg n)$

(c) [8 points]

```
1  # Counts the number of size-k subarrays A[i:i+k]
2  # of A that have a sum of zero.  Assume k > 0.
3  def get_num_zero_subarrays(A, k):
4      ans = 0
5      for i in range(len(A) - k + 1):
6          # sum(X) iterates through the elements of X and
7          # calculates their sum.
8          s = sum(A[i:i+k])
9          if s == 0:
10             ans += 1
11     return ans
```

Solution:

1. $O(nk)$

2. Maintain a rolling sum in `s`. Add before line 5: `s = sum(A[:k-1])` Replace line 8 with `s += A[i+k-1]`. Add after line 10 (inside `for` loop): `s -= A[i]`.

3. $O(n)$

(d) [8 points]

```
1  # Computes the length of the longest increasing subsequence
2  # in the suffix A[k:], assuming the sequence starts with A[k].
3  def longest_increasing_subsequence(A, k):
4      best = 1  # Sequence could be A[k] by itself.
5      # Guess that the sequence goes from A[k] to A[m].
6      for m in range(k+1, len(A)):
7          if A[m] > A[k]:  # Sequence must be increasing.
8              solution = 1 + longest_increasing_subsequence(A, m)
9              if solution > best: best = solution
10     return best
11 # You can assume that this function gets called only once
12 # by another function (i.e., ignoring recursions).
```

Solution:

1. Exponential. $T(n) = \sum_{i=1}^{n-1} T(i) \geq T(n-1) + T(n-2) \geq 2\,T(n-2) = 2^{n/2}$.

2. Memoize. Add before line 3: `memo = {}`. Add before line 4: `if k in memo: return memo[k]`. Add before line 10: `memo[k] = best`.

3. $O(n^2)$.

Problem 3. [10 points] **Shortcut through the Circle of Life** (1 part)

Prof. Timon has a weighted undirected graph $G = (V, E, w)$ that is just a cycle (i.e., G is connected and every vertex has degree exactly 2). It has two distinguished vertices $s, t \in V$.

Design an $O(V + E)$-time algorithm to find the shortest path in G from s to t.

Solution: One can check in $O(E)$ time whether any edge is negative-weight; if so, return $-\infty$, because one can use it arbitrarily many times. Otherwise, there are exactly two simple paths from s to t, one going in each direction from S. The weight of each of those paths can be calculated in time proportional to the number of edges in it, and between them those two paths use all E edges, taking a total of $O(E)$ time. Finally, compare the two of them in $O(1)$ time and return the smaller.

Problem 4. [15 points] **I'm All About That Base** (3 parts)

Tired of regular old binary heaps, your colleagues at 6006LE decide to explore ***k-ary min-heaps*** which have the following properties:

1. Every node has exactly k children, except for the leaves which have zero children and are all at the same level, and except for the rightmost node in the level above the leaf level which has $\leq k$ children.
2. The key of every node is less than or equal to the keys of its children,

(a) [2 points] What is the height of an n-node k-ary min-heap? You can use Θ notation, but analyze in terms of both n and k.

Solution: Consider a k-ary heap of height h (i.e. contains $h+1$ full-node levels), where level 0 contains the root. The ith level of the heap contains k^i nodes. The total number of nodes in this heap is

$$\sum_{i=0}^{h} k^i = k^h \left(\sum_{i=0}^{h} \frac{1}{k^i} \right) = \Theta(k^h).$$

We can see that solving $n = ck^h$ for h where $c > 0$ is a constant yields

$$h = \Theta\left(\frac{\lg n}{\lg k}\right) = \Theta(\log_k n).$$

(b) [5 points] Design an array representation $A[0\,..\,n-1]$ of an n-node k-ary min-heap that lets you find the parent and cth child ($1 \leq c \leq k$) of node $A[i]$ in constant time (without pointers). You do not need to prove your answer. *Hint:* Make sure your solution works for $k = 2$ (regular binary heaps).

Solution: There are two approaches for this problem. A simple, intuitive approach is to adapt the array-index formulas for the binary heap covered in lecture for a k-child heap, and replace the 2 with a k and $0/1$ with c. This results in:

$$child(i, c) = ki + c.$$

$$parent(i) = \lfloor (i-1)/k \rfloor.$$

Another approach, and perhaps more rigorous, is to derive the formulas above in a similar fashion to how we derived the child/parent formulas for binary heaps. This can be viewed here: `http://imgur.com/a/PoKf9`.

(c) [8 points] Describe how to implement EXTRACT-MIN in an n-node k-ary min-heap. Analyze the asymptotic running time of your algorithm in terms of n and k. For full credit, your algorithm should run in $O(\log n)$ time whenever $k = O(1)$.

Solution: We adapt the corresponding algorithm for EXTRACT-MIN in a binary heap. Using the array representation from part (b):

1. Pop the minimum from the heap: remove the index zero element from the array and replace it with the last element in the array
2. Run an adapted verion of MIN-HEAPIFY, swapping using the formulas from part (b): starting with the root, trickle down the newly placed root element, swapping when the root element is greater than any of its children. If a swap is necessary, swap it with the minimum of the k children. Repeat until the heap restored the min-heap invariant.

The first step takes constant time. The second step takes $O(k)$ per level, and there are $O(\log_k n)$ levels. The final complexity of the whole procedure is $O(k \log_k n)$.

Problem 5. [10 points] **Escape the Graph** (1 part)

Prof. Houdini needs to find the quickest way out of any node in a graph. Design a data structure for representing a weighted directed graph $G = (V, E, w)$ supporting the following operations and time bounds:

FASTEST-ESCAPE(u): Find and delete the **minimum-weight** outgoing edge $(u, v) \in E$ from a given vertex $u \in V$, in $O(\log \textit{out-degree}(u))$ time.

INSERT(u, v): Insert an edge (u, v) into E in $O(\log \textit{out-degree}(u))$ time.

ADJ(u): List all outgoing edges $(u, v) \in E$ from a given vertex $u \in V$, in $O(\textit{out-degree}(u))$ time.

Solution: For each vertex u, we store a binary heap $u.\textit{outgoing}$ containing every outgoing edge (u, v) from u, keyed by its weight $w(u, v)$. Then we can implement the required functions as follows:

FASTEST-ESCAPE(u): HEAP-DELETE-MIN($u.\textit{outgoing}$)

INSERT(u, v): HEAP-INSERT($u.\textit{outgoing}$, (u, v), $\textit{key} = w(u, v)$)

ADJ(u): Return $u.\textit{outgoing}$ viewed as an array. (Equivalently, traverse all nodes in the heap and return all traversed edges.)

Alternatively, we could use an AVL tree instead of a binary heap, again keyed by edge weight, with the following implementations:

FASTEST-ESCAPE(u): AVL-DELETE(BST-FIND-MIN($u.\textit{outgoing}$))
(i.e., descend left until finding a leaf, and AVL-DELETE it)

INSERT(u, v): AVL-INSERT($u.\textit{outgoing}$, (u, v), $\textit{key} = w(u, v)$)

ADJ(u): BST-IN-ORDER-TRAVERSAL($u.\textit{outgoing}$) and output all traversed edges.

The first two operations run in time $O(\log |u.\textit{outgoing}|) = O(\log \textit{out-degree}(u))$, while the third operation runs in time $O(|u.\textit{outgoing}|) = O(\textit{out-degree}(u))$.

Instead of storing $u.\textit{outgoing}$ as an attribute of u, we could use a dictionary (hash table) to map vertices to heaps/AVL trees, without any loss of performance (in expectation). But using an AVL tree to map vertices to heaps/AVL trees would incur an additive $\Theta(\log n)$ time cost, which can be larger than the required time bound.

Problem 6. [10 points] **Groundhog Graphs** (1 part)

For every positive integer n, show how to construct

1. an unweighted undirected graph $G = (V, E)$ with $|V| = n$ vertices,
2. a source vertex $s \in V$, and
3. an ordering of the vertices in V

such that running DFS and BFS on G from start node s discovers the nodes in the same order. (Recall that the ***discovery time*** of a node v is the first time v is encountered during one of the search algorithms.)

Solution: Any n-vertex path with the source vertex at one end suffices: both DFS and BFS visit in path order.

An $(n-1)$-leaf star with the source vertex the center of the star suffices: both DFS and BFS visit the other vertices in the order of them determined by the order of all the vertices.

(There are more solutions.)

Problem 7. [18 points] **Legal Need For Speed** (2 parts)

You're racing through Cambridge, which is described by a weighted directed graph $G = (V, E, w)$ where vertices represent intersections and edges represent roads. The weight $w(u, v)$ of road (u, v) is the positive integer of time it takes to traverse the road from u to v.

Unfortunately, the city has **traffic lights**: to leave an intersection $v \in V$, you must wait until the current time is an integer multiple of $f(v)$. Your goal is to find the shortest-time path from $s \in V$ to $t \in V$ in $O(E + V \lg V)$ time, assuming you start at s at time 0 (and thus can leave it immediately).

(a) [8 points] Assume that $f(v)$ is the same value F for all vertices $v \in V$. Show how to modify the weights w into w' such that running Dijkstra on $G' = (V, E, w')$ from s correctly solves the problem. You do not need to prove correctness.

Solution: The new weight function $w' : V \to \mathbb{N}$ is given by

$$w'(u,v) = \begin{cases} w(u,v) \text{ rounded up to the next integer multiple of } F\,, & \text{if } v \neq t; \\ w(u,v), & \text{otherwise.} \end{cases}$$

Note that traffic lights are used only when leaving intersections, and thus we do not need to wait for traffic lights on the roads that lead to t.

Common Mistake: The most common mistake was to set $w'(u, v) = w(u, v) + F$, which does not obey the constraint that we can leave intersections only at times multiples of F.

(b) [10 points] Now consider the general case where $f(v)$ can be different for different vertices $v \in V$. Show how to change the RELAX(u, v) subroutine so that running Dijkstra on G from s correctly solves the problem. You do not need to prove correctness.

Solution: We can modify the RELAX function as follows:

RELAX(u, v)
  if $d[v] >$ [$d[u]$ rounded up to the next integer multiple of $f(u)$] $+ w(u, v)$
    $d[v] =$ [$d[u]$ rounded up to the next integer multiple of $f(u)$] $+ w(u, v)$

Under the above modification, $d[u]$ is an estimate for the least time it takes to reach intersection u without waiting on the traffic light at u.

Common Mistake: The most common mistake was to modify only the weights of the edges, which is not sufficient, because we need to take into account the time it takes to reach a given intersection before knowing exactly how much we need to wait on the particular traffic light.

Problem 8. [18 points] **Squeezing into the Page Limit** (3 parts)

You've already written your solutions to Problem 8 of this final, but there is a requirement that you get it to fit on exactly one page, which is exactly k lines long. Your solutions consist of n words given by array $words[0\,..\,n-1]$. You've defined a function $badness(i, j)$ that measures how ugly it would be to squeeze words $words[i : j]$ onto one line.

You've defined the following **subproblems**: $DP[i, k']$ is the optimal justification (of minimum total badness) of suffix $words[i :]$ into exactly k' lines.

(a) [6 points] How many subproblems are there? Use Θ notation. You do not need to prove your answer.

Solution: There are $\Theta(nk)$ subproblems because i varies between 0 and $n-1$, and k' varies between 0 and k.

(b) [6 points] Give a recurrence relating subproblem solutions. You do not need to prove correctness.

Hint: Recall from class the dynamic programming recurrence for justifying text into lines so as to minimize total badness of the lines:

$$DP[i] = \min\Big(badness(i, j) + DP[j] \text{ for } j \text{ in range}(i+1, n+1)\Big).$$

Solution: The recurrence (ignoring basecases) is given by

$$DP[i, k'] = \min\Big(badness(i, j) + DP[j, k'-1] \text{ for } j \text{ in range}(i+1, n+1)\Big).$$

(c) [6 points] What is the running time of the resulting dynamic program? You do not need to prove your answer.

Solution: There are $\Theta(nk)$ subproblems as shown in part (a). Since each subproblem takes $\Theta(n)$ to compute, the dynamic program takes $\Theta(n^2k)$ time.

Problem 9. [15 points] **Love Triangles** (3 parts)

Suppose you are given a list of n distinct points $p_1, p_2, \ldots, p_n$ with coordinates (x_1, y_1), (x_2, y_2), $\ldots$, (x_n, y_n). Assume that all coordinates are $\Theta(\log n)$-bit integers, and operations on these integers take constant time.

(a) [5 points] Design an expected $O(n)$-time algorithm to list all distinct x coordinates (in no particular order) and, for each such x coordinate, the number of points with that x coordinate. (Even if you don't solve this part, assume you have a solution for future parts.)

Solution: We start by creating an empty hashing-backed-dictionary H, also known as *hashtable*, that stores key-value pairs of two integers: x coordinates as keys, and integer counts as values. We iterate over all n points, and for each coordinate x_i check whether x_i is already present in the dictionary by using SEARCH(H, x_i):

- if x_i is not present in H, then we apply INSERT$(H, (x_i, 1))$, which adds a single occurrence of the coordinate x_i to the dictionary;
- if x_i is present in H, then we modify the value for the key-value pair (x_i, c) from c to $c+1$ to account for the additional point with x-coordinate x_i; we can accomplish this with a single DELETE followed by INSERT.

Finally, we iterate over H to list all distinct x coordinates, and the number of points with that x coordinate. Note that since all hashing operations takes $O(1)$ expected time, the running time of this algorithm is $O(n)$.

(b) [5 points] Design an expected $O(n)$-time algorithm to count the number of unordered pairs of distinct points $\{p_i, p_j\}$ having the same x coordinate ($x_i = x_j$) or same y coordinate ($y_i = y_j$), i.e., the number of unordered pairs of distinct points $\{p_i, p_j\}$ that define a line parallel to the x or y axis. (*Hint:* Use the solution to part (a) on both x and y.)

Solution: Obtain a list of all distinct x coordinates and number of points with those coordinates in $O(n)$ with the algorithm for part (a). Suppose that this list is given in terms of pairs $(x_1, k_1), (x_2, k_2), \ldots, (x_m, k_m)$, where $m \leq n$. The number of unordered pairs of points with the same x coordinate is given by

$$\sum_{i=1}^{m} \binom{k_i}{2} = \sum_{i=1}^{m} \frac{k_i(k_i - 1)}{2},$$

which takes $O(n)$ time to compute.

Analogously, we can obtain the number of unordered pairs of points that have the same y coordinates in $O(n)$. The total number of unorder pairs that have either have the same x coordinate or the same y coordinte is given by the sum of the two quantities we have computed.

(c) [5 points] Design an expected $O(n)$-time algorithm to count the number of (right) triangles $\triangle p_i p_j p_k$ having one side parallel to the x axis and another side parallel to the y axis. (*Hint:* Use the solution to part (a) on both x and y.) Partial credit will be awarded for slower but correct algorithms.

Solution: Let $P = \{(x_i, y_i) : 1 \leq i \leq n\}$ be the set of all points.

Let us count the number of right triangles that have a particular vertex (x, y) that is opposite of the hypotenuse. Consider the set

$$\mathcal{X} = \{(x', y') \in P : (x', y') \neq (x, y) \text{ and } x' = x\}$$

of other points with the same x coordinate. Define $\mathcal{Y}$ analogously. Since each point

$$(x', y') \in \{(x'', y'') : (x'', \cdot) \in \mathcal{X}(x), (\cdot, y'') \in \mathcal{Y}(y)\}$$

yields a distinct right triangle, the number of right triangles with fixed vertex (x, y) opposite of the hypotenuse is just $|\mathcal{X}(x)| \cdot |\mathcal{Y}(y)|$.

Use the solution to part (a) on both x and y. Let us denote the number of points with x coordinate of x' by $H_x[x']$, and similarly for $H_y[y']$. Note that $H_x[x']$ and $H_y[y']$ are almost the same as $|\mathcal{X}(x')|$ and $|\mathcal{Y}(y')|$. In fact, we only need to discount the point (x, y) to get the right counts. Therefore, the number of right triangles with a fixed point (x, y) opposite of the hypotenuse is given by $(H_x[x] - 1)(H_y[y] - 1)$. Summing this expression over all n points (x, y) gives the number of right triangles. The expected running time of this algorithm is $O(n)$.

JOKE QUESTIONS. DO NOT ANSWER THESE.

BUT ALSO DO NOT REMOVE FROM THE EXAM.

- [0 points] What are the names of the professors? (Do not look back at page 1.)

  Solution: Erik Demaine, Debayan Gupta, and Ronitt Rubinfeld

- [0 points] What is the name of your recitation instructor?

  Solution: Who knows !?

- [0 points] Write every misspelling of "Dÿkstra" that at least one student wrote on this exam.

  Solution:

  - Dikestra
  - Dikstra
  - Dikjstra
  - D$\epsilon\iota$kstra

- [0 points] For each problem on this exam, guess who wrote it.

  Solution: it's a grab bag, because most of the staff had things to say about almost every problem, and in some way, shape, or form helped mold the test as it is, but the original problem-contributor for the open-ended questions were:

  - 2: Akshay
  - 3: Ray Hua
  - 4: Skanda
  - 5: Adam
  - 6: Anonymous
  - 7: Anton
  - 8: Anton
  - 9: Anton

- [0 points] What percentage of students will not get full credit on Problem 0 on this exam?

  Solution: According to Gradescope, 10%.

- [0 points] How many emails were sent to 6.006-staff@mit.edu and/or free-fruit@mit.edu during this semester?

  Solution: Since Auguest 22, 87 emails were sent to `free-fruit@mit.edu` and 728 emails were sent to `6.006-staff@mit.edu`.

- [0 points] How many questions/posts were asked on Piazza?

  Solution: 1317 posts (questions and notes) were posted on the Fall 2016 6.006 Piazza.

- [0 points] If you were an algorithm, which algorithm would you be?

  Solution: Subjective, any reasonable answer accepted (e.g. Bellman-Ford).

- [0 points] Which algorithm is the best algorithm?

  Solution: Subjective, any reasonable answer accepted (e.g. BFS).

- [0 points] Write the asymptotically slowest sorting algorithm, among all students' answers to this question on this final.

  Solution: I haven't examined all student solutions, but of the ones I've seen, no-one listed anything slower than $O(n^3)$. Hmf – low bar to beat!