Quiz 2 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 every page of this quiz booklet.
- You have 120 minutes to earn 110 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 two 1-page cheat sheets.** 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. Simply cite them.
- When writing an algorithm, a **clear** description in English will suffice. Pseudo-code is not required unless asked for.
- **Pay close attention to the instructions for each problem.** Depending on the problem, partial credit may be awarded for incomplete answers.
- **Blank answers will receive approximately 25% of the credit for the problem.** If your answer shows no understanding of the problem, you will receive less than 25%. So, if you have no idea how to attack a problem, you should leave it blank.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Parts</th>
<th>Points</th>
<th>Grade</th>
<th>Grader</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td><strong>110</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Name: ____________________________

Circle your recitation:

- R01 Gurtej Ashwin 10AM
- R02 Gurtej Ashwin 11AM
- R03 Jennifer Quanquan 12PM
- R04 Jennifer Quanquan 1PM
- R05 Arvind Vlad 2PM
- R06 Arvind Vlad 3PM
- R07 Justin Sherwin 11AM
- R08 Justin Sherwin 12PM
Problem 1.  [20 points]  **True or False**

Circle True or False for each statement below. If you need to make any assumptions for each question, you may write a sentence of explanation next to your answer.

(a) T F  [2 points]  Dijkstra’s Algorithm can be implemented using an AVL tree instead of a heap for the Priority Queue and still run in $O((V + E) \log V)$ time.

True. Using AVL trees, both Decrease-Key and Extract-Min can be implemented in $O(\log V)$ time, so the overall running time in $O((V + E) \log V)$ time.

(b) T F  [2 points]  Dijkstra’s Algorithm can be implemented using a sorted array instead of a heap for the Priority Queue and still run in $O((V + E) \log V)$ time.

False. With a sorted array, Decrease-Key needs $\Theta(V)$ time in the worst case (to shift the array), so the overall running time would be $\Theta((V + E)V)$.

(c) T F  [2 points]  Using the fastest known algorithms, multiplying two $n$-bit numbers is asymptotically faster than dividing two $n$-bit numbers (up to $n$ bits of precision).  

(Asymptotically faster means “faster by more than a constant factor”.)

False. As we saw in lecture, division can be done via Newton’s Method using only a constant factor more time than any given method for multiplication.

(d) T F  [2 points]  Suppose the function $f$ (from reals to reals) has positive first and second derivatives everywhere.

Then Newton’s Method successfully finds a root of the equation $f(x) = 0$ no matter what starting point $x_0$ is used. (You may assume that the equation does have a solution.)

True, as argued in lecture via a graphical approach.

(e) T F  [2 points]  It is possible to choose the distance heuristic for $A^*$ so that $A^*$ becomes equivalent to Dijkstra’s algorithm.

True, by setting all heuristic distances to zero.
(f) T F [2 points] The asymptotically fastest way to find a negative-weight cycle in a graph is to discard all of the positive-weight edges and then use DFS to find a cycle in the remaining graph.

False: this algorithm is incorrect, as it may miss negative-weight cycles with both positive and negative weights. (In fact, the best known algorithm is to run Bellman-Ford.)

(g) T F [2 points] Because of the $\Omega(n \lg n)$ lower-bound on comparison-based sorting, it is not possible to have an algorithm for topologically sorting an $n$-vertex $n$-edge directed graph that runs in time $o(n \lg n)$.

False: in fact, topological sort on such a graph will run in $O(n)$ time.

(h) T F [2 points] In Bellman-Ford, suppose we have two consecutive edges on a shortest path: $(u, v)$ and $(v, w)$. Edge $(v, w)$ is guaranteed to be relaxed within $O(1)$ relaxations of $(u, v)$ being relaxed.

False: Bellman-Ford runs repeatedly through all edges in some arbitrary order, so two edges may be visited $\Theta(E)$ steps away from each other.

(i) T F [2 points] A graph $G$ has a cycle if and only if a DFS traversal produces a back edge.

True, as argued in lecture.

(j) T F [2 points] If a DFS on a connected undirected graph produces $n$ back edges, then a BFS on the same graph will produce $n$ cross-edges.

True: BFS on an undirected graph has no forward or back edges, while DFS on an undirected graph has no forward or cross edges.
Problem 2.  [10 points] Multi-Source Shortest Paths

How do you compute shortest paths when the source is not a single vertex \( s \), but instead a set \( S \) of vertices? More precisely, suppose you are given a weighted directed graph \( G = (V, E, w) \) with non-negative edge weights, you are given a source set \( S \subseteq V \), and you are given a single target node \( t \in V \). Your goal is to find the weight of the shortest path from some vertex \( s \in S \) to \( t \). (In other words, among all choices \( s \in S \), you want to find the choice that results in the shortest shortest path from \( s \) to \( t \).) Describe an efficient algorithm for computing this *multi-source shortest path distance*. For full credit, your algorithm should run in \( O(V \lg V + E) \).

| Solution 1. Add a dummy vertex \( s^* \) and add zero-weight edges from \( s^* \) to all \( s \in S \). Run Dijkstra from \( s^* \). Running Dijkstra on \( O(V) \) vertices and \( O(E) \) edges using a Fibonacci heap takes \( O(V \lg V + E) \) time. This is essentially the same as presetting the distances to 0 for all \( s \in S \). |
| Solution 2. Reverse all edges. Run Dijkstra from \( t \) until some vertex \( s \in S \) is extracted from the heap. Reversing the edges takes \( O(E) \) time and running Dijkstra on \( O(V) \) vertices and \( O(E) \) edges using a Fibonacci heap takes \( O(V \lg V + E) \) time. |
Problem 3. [25 points] Robot Treasure Hunt

Rossum the Robot is looking for a treasure. He is in a maze that can be modelled as an undirected graph $G = (V, E)$, where each vertex is a pair $(x, y)$ of non-negative integers with $0 \leq x + y < n$. Rossum starts at the origin $s = (0, 0)$. Each vertex $(x, y)$ (other than the origin) has an edge connecting it to either the vertex below it $(x, y - 1)$ or the vertex to its left $(x - 1, y)$, but not both. See figure above. Thus, there is a unique path from each vertex back to the origin, and the graph is a tree. Rossum can travel along any edge of the graph in unit time, but cannot teleport between two distant vertices (as a naive implementation of BFS would require).

Suppose that the treasure is located at an unknown vertex $t = (i, j)$ at distance $r = i + j$ from the origin. Rossum’s goal is to find the treasure by visiting vertex $t$ in $O(r^2)$ steps. When he starts the search, he does not know $t$, or any bound on $r$, or even $n$.

(a) [5 points] Explain why DFS will find $t$, but may fail to achieve time bound $O(r^2)$.

DFS will eventually visit all the vertices in the graph, but Rossum may wander off and end up visiting $O(n^2)$ vertices before finding $t$. Since $n >> r$, $n^2 \neq O(r^2)$. 


(b) [10 points] Rossum then thinks of “DFS with progressive deepening”, where he first searches (using DFS) the graph of vertices at distance at most distance 1 from s, then (if the treasure isn’t yet found) those at most distance 2 from s, then (if the treasure isn’t yet found) those at most distance 3 from s, and so on. (Here the DFS is modified to take a depth bound b as an additional argument, and DFS won’t visit any vertices at distance greater than the depth bound b from s.) For a given b, how many nodes are within the depth bound b (asymptotically)? Explain why this algorithm will fail to achieve the desired time bound of $O(r^2)$.

There are $\frac{(b + 1)(b + 2)}{2} = O(b^2)$ nodes reachable within depth bound b. Thus each iteration takes $O(b^2)$ time to complete, and so the total time to find $t$, will be $\sum_{b=0}^{r} \frac{(b + 1)(b + 2)}{2} = O(r^3)$.

(c) [10 points] Explain how to modify the choice of depths of the previous problem part in a way that ensures the desired time bound of $O(r^2)$.

We can modify our choice of depth bounds to double each iteration. In particular, we first try $b = 1$, then $b = 2$, $b = 4$, $b = 8$, etc. Using this strategy, to find $t$, the algorithm will take time $\sum_{i=0}^{\log r} (2^i)^2 \leq 2^{2 \log r + 1} = O(r^2)$. 
Problem 4. [20 points] Use your graph algorithms

(a) [4 points] Fill in the chart below with the worst-case runtimes of the single source shortest path (SSSP) algorithms you learned in class (listed below) on a graph with \( V \) vertices and \( E \) edges, when the algorithms successfully return a shortest path.

<table>
<thead>
<tr>
<th>Label</th>
<th>Algorithm</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>BFS (possibly including a linear-time graph transformation)</td>
<td>( O(V + E) )</td>
</tr>
<tr>
<td>b</td>
<td>Bellman-Ford</td>
<td>( O(VE) )</td>
</tr>
<tr>
<td>c</td>
<td>Dijkstra’s Algorithm (binary heap implementation)</td>
<td>( O((V + E) \log V) )</td>
</tr>
<tr>
<td>d</td>
<td>Topological sort and relaxation</td>
<td>( O(V + E) )</td>
</tr>
</tbody>
</table>

(b) [16 points] Fill in the chart below with the label (a,b,c,d) of the most efficient algorithm listed above to be used on the given class of graphs. If the graphs in the class could have no finite SSSPs, write down the algorithm which you can use to determine whether this is the case.

<table>
<thead>
<tr>
<th>Graph</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive weighted DAGs</td>
<td>d</td>
</tr>
<tr>
<td>Arbitrarily weighted DAGs</td>
<td>d</td>
</tr>
<tr>
<td>Weighted graph with no negative cycles</td>
<td>b</td>
</tr>
<tr>
<td>Weighted graph with positive integer weights no greater than 5</td>
<td>a</td>
</tr>
<tr>
<td>Weighted graph with negative cycles</td>
<td>b</td>
</tr>
<tr>
<td>Unweighted graph</td>
<td>a</td>
</tr>
<tr>
<td>Complete undirected graph (all nodes are connected) with positive weights</td>
<td>c</td>
</tr>
<tr>
<td>Weighted tree graph</td>
<td>a,d</td>
</tr>
</tbody>
</table>

NOTE: We accepted both BFS and Topological sort for the last cell. We intended the tree to be undirected, in which case the correct answer should be BFS (Topological sort cannot work on undirected graphs). However, we realize that it was perfectly reasonable to assume that the tree was directed, in which case either BFS or Topological sort would both work correctly.
Problem 5.  [20 points] Bellman-Ford Relaxation Shortcuts

Ben Bitdiddle is given a directed graph $G = (V, E)$. Each edge $e_{i,j}$ has an associated edge weight $w_{i,j}$, which can be any real number. Ben knows that $G$ does not contain any negative-weight cycles. Ben is interested in finding the minimum-weight path between vertex $X$ and vertex $Y$.

(a) [5 points] Ben uses BFS from $X$ and determines that $Y$ can be reached by traversing $t_1$ edges. Rather than performing $|V| - 1$ iterations of Bellman-Ford, Ben decides to perform $t_1$ iterations. Ben’s new algorithm is not guaranteed to return the minimum-weight path between $X$ and $Y$. Give an example of a graph in which Ben’s algorithm will not return the minimum-weight path.

(b) [5 points] Ben persists and finds the length $t_2$ of the longest acyclic path in $G$. For Ben’s graph, $t_2 < |V| - 1$. Ben then performs $t_2$ iterations of Bellman-Ford. Will Ben’s latest algorithm return the minimum-weight path from $X$ to $Y$? If so, provide justification. If not, provide a counterexample graph $G$.

Yes. Ben’s algorithm will return the minimum-weight path. The invariant of the Bellman-Ford algorithm states that after $k$ iterations, $d(v)$ will store the minimum-weight cost to reach $v$ in at most $k$ hops. Since we know $G$ does not contain any negative-weight cycles, all paths longer than $t_2$ must be cyclic and therefore not minimum weight paths. Therefore, $d$ must converge to $\delta$ in $t_2$ iterations.

(c) [10 points] Is there a relaxation ordering of edges in $G$ such that Bellman-Ford will find the minimum-weight path from $X$ to every other vertex after only one iteration of Bellman-Ford? If so, provide justification. If not, provide a counterexample graph $G$ and show why no such relaxation ordering can exist.

Yes. First, we run Bellman-Ford with $|V| - 1$ iterations. The “parent points” $\pi$ will induce a subgraph $G'$ of $G$ in which the only edges that appear are edges of shortest paths. Graph $G'$ cannot contain any cycles because $G$ does not have any negative-weight cycles. Furthermore, $G'$ must be a tree, because every reachable vertex has exactly one parent. In addition, the minimum cost to reach any vertex in $G'$ is exactly the minimum cost to reach that vertex in $G$. (In fact, $G'$ is called the shortest path tree.) We can use any pre-order traversal of the tree $G'$ (e.g., via DFS, BFS, or a topological sort) as the relaxation ordering of edges to force Bellman-Ford to converge to the correct solution in only one iteration.
Problem 6. [15 points] **Low-Precision Newton’s Method**

Ben Bitdiddle has recently learned about Newton’s Method and wants to apply it to find the zero of a function $f$:

$$f(x) = \sqrt{1 + x} - a.$$ 

Ben is interested in finding the zero to $k$ digits of precision. Assume that $a$ has $n$ digits of precision where $n >> k$.

(a) [5 points] Describe the Newton’s Method update rule for $f$.

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

$$= x_i - \frac{2(\sqrt{1 + x_i} - a)}{(1 + x_i)^{0.5}}$$

$$= x_i - 2(1 + x_i) + 2a\sqrt{1 + x_i}$$

$$= -2 - x_i + a\sqrt{1 + x_i}$$

(b) [5 points] Assume that calculating multiplication, division, and square root operations to $p$ digits of precision all take $O(M(p))$ time where $M(p) = \Omega(p)$. Ben knows that the update rule you derived displays quadratic convergence. Ben’s initial guess for the zero of $f$ is correct to 1 digit of precision. Ben calculates that he’ll need to do $O(\lg k)$ updates to find the root of $f$ to $k$ digits of precision. Because Ben is interested in the final answer having $k$ digits of precision, he assumes that each update will take $O(M(k))$ time, and therefore, the entire procedure will take $O(M(k) \lg k)$ time. How can Ben modify his analysis to compute the zero of $f$ in $O(M(k))$ time?
Ben assumes that each iteration of Newton’s method will require computing estimates to $k$ digits of precision. However, since we know that in the $i$-th iteration, the estimates will only have $O(2^i)$ digits of precision. Therefore, we can reduce the precision of calculating updates to correspond to the expected precision of the estimate. If $T(k)$ represents the amount of time needed to compute a zero to $k$ digits of precision, then our new algorithm has the recurrence:

$$T(k) = T(k/2) + O(M(k))$$

By the master theorem, since $M(p) = \Omega(p)$, the new recurrence has the solution $T(k) = O(M(k))$.

(c) [5 points] Ben realizes that, when $f(x) = 0$, he can write $x = a^2 - 1 = (a+1)(a-1)$. Assume that multiplying two $p$-digit-precision numbers will give us a $p$-digit-precision answer. Describe a modification to the Karatsuba algorithm that will allow Ben to compute $(a+1)(a-1)$ to $k$ digits of precision in $O(k \log_2 3 + n)$ time.

Naive karatsuba will compute the value of $(a+1)(a-1)$ exactly in $O(n \log_2 3)$ time. However, since $n \gg k$, we need a shortcut. We can drop the last $n - k$ bits from $(a+1)$ and $(a-1)$ via a right bit-shift operation, and call these truncated values $a^+$ and $a^-$. We can then compute an approximation of $(a+1)(a-1)$ as $a^+ \times a^- \times 2^{2(n-k)}$. The first multiplication is between $k$ digit numbers and takes $O(k \log_2 3)$ time using Karatsuba. The second multiplication is a bit-shift and takes $O(n)$ time. Therefore, the total operation takes $O(k \log_2 3 + n)$ time.