Midterm Exam 1

- Do not open this exam booklet until you are directed to do so. Read all the instructions on this page.

- When the exam begins, write your name on every page of this exam booklet.

- This exam contains 4 problems, some with multiple parts. You have 120 minutes to earn 80 points.

- This exam booklet contains 10 pages, including this one. Two extra sheets of scratch paper are attached. Please detach them before turning in your exam.

- This exam is closed book. You may use one handwritten $8\frac{1}{2} \times 11$ or A4 crib sheet. No calculators or programmable devices are permitted.

- Write your solutions in the space provided. If you need more space, you may use the scratch sheets at the back of the exam.

- Do not waste time and paper rederiving facts that we have studied. It is sufficient to cite known results.

- 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.

- 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!

Name: ___________________________ ID: __________

<table>
<thead>
<tr>
<th>Problem</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Points</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>20</td>
<td>80</td>
</tr>
<tr>
<td>Grade</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Problem 1 (Miscellaneous).

(a) Each of the items below describes a function of a parameter $n$. For each one, give a simple big-theta ($\Theta$) expression for the function, of the form $\Theta(2^an^b\log^cn)$ where $a, b, c \geq 0$.

(a) The running time of Kruskal’s algorithm, implemented using a binary heap, on graphs with $n$ vertices and $n^{0.5}$ edges.

(b) The solution to the recurrence given by $T(n) = 9T([n/3]) + n^2$ and $T(1) = 1$.

(c) The solution to the recurrence given by $T(n) = 10T([n/3]) + n^2$ and $T(1) = 1$.

(d) The solution to the recurrence given by $T(n) = T(n - 1) + 2^n$ and $T(1) = 1$.

(e) $\sum_{i=1}^{n} i3^i$

(f) $\sum_{i=1}^{n^2} \sqrt{i}$

(g) $\sum_{i=1}^{n} \frac{i}{3^i}$

(b) In an effort to simplify its more obscure practices, Flying Dutchman Airlines (FDA) has decided on a new pricing model: the cost of a trip in dollars will equal the sum of the lengths, in miles, of the two longest segments. So if the two longest segments are 200 and 3000 miles long, respectively, the trip will cost $3200 (simple doesn’t mean cheap). Trips with only one segment will cost the length of that segment.

A clause in FDA’s charter states that it can only use or modify algorithms originally designed by Dutch computer scientists. They have asked you to modify Dijsktra’s algorithm to compute the cheapest flight between two given airports $s$ and $t$. The input is a directed graph $G$ in which every vertex is an airport, and an edge of length $\ell(u, v)$ represents a flight they operate from $u$ to $v$ that covers $\ell$ miles.
Algorithm 1: Dijkstra(G = (V, E), ℓ(·), s, t)

1. for each vertex v ∈ V do
   2. dist[v] ← ∞ ; /* Unknown distance from source to v */
   3. previous[v] ← undefined ; /* Previous node in optimal path from s */
   4. dist[s] ← 0 ;
   5. Q ← empty priority queue keyed on dist[];
   6. Q ← V ;

7. while Q is not empty do
   8. u ← extract-min(Q);
   9. for each neighbor v of u do
      10. alt ← dist[u] + ℓ(u, v) ;
      11. if alt < dist[v] then
          12. dist[v] ← alt; /* Implicitly performs decrease-key in Q. */
          13. previous[v] ← u;

14. return dist[t] ;

Indicate in the pseudocode of Algorithm 1 which lines you would change, and how. [Hint: you’ll have to store some extra information at each node.]
(c) Let $G$ be a weighted graph $G = (V, E)$ with weight function $w : V \rightarrow \mathbb{R}$. Suppose you are given an MST $T$ for $G$. Now one of the weights in $G$ gets decreased, that is, one edge $e$ gets assigned weight $w' < w(e)$. Describe a $O(n)$ time algorithm that finds a new MST $T'$. (Hint: there are two cases: either $e$ is in $T$ already, or it is not.)

(d) Explain why your algorithm from the previous question is correct.
Problem 2. (Interval Sub-Covers) Given a family of open intervals \((a_i, b_i), i = 1, \ldots, n\), a sub-cover is subset of the intervals that covers the same area of the real line as the union of all the intervals. For example, in the following picture, the gray intervals form a sub-cover consisting of 5 intervals:

You want to design an efficient algorithm that finds a sub-cover with as few intervals as possible. In the following, you may assume that (i) all the endpoints \((a_i, b_i)\) are distinct, and (ii) the original set of intervals covers a contiguous segment of the real line.

(a) Your friend suggests the following greedy approach: at each stage, add the interval with the most length not covered by the intervals selected so far. Give an example showing that this approach will not find the smallest sub-cover.

(b) Give a polynomial-time algorithm to find a smallest sub-cover, given the \(a_i\)'s and \(b_i\)'s as input. Faster (correct) algorithms are worth slightly more points than slower ones.

(c) What is the running time of your algorithm?
(d) Explain carefully why your algorithm is correct. You may find it helpful to read the rest of the question first.

(e) Given a set of intervals, a spread is a set of points no two of which are contained by a single interval. Prove that if there exists a spread of size $k$, then every sub-cover must have size at least $k$.

(f) Show that for any collection of intervals, the size of the largest spread is exactly equal to the size of the smallest sub-cover. [Hint: can you find an algorithm that produces a sub-cover and a spread of the same size?]
Problem 3 (Recursion). Recall that the degree of a vertex in an undirected graph is the number of vertices adjacent to it; the maximum degree of a graph $G$ is the largest degree of any vertex in $G$.

Recall that a chain graph of size $k$ is a sequence of vertices $\{v_1, \ldots, v_k\}$ with edges $(v_i, v_{i+1})$ for $i = 1, \ldots, k - 1$. A cycle graph of size $k$ is a chain graph with an additional edge $(v_k, v_1)$.

(a) Show that a graph of maximum degree 2 or less is a union of disjoint chains and cycles.

(b) Give an algorithm that finds the size of the largest independent set in such a graph in time $O(n)$.

(c) Fill in the missing pieces of the following recursive algorithm for finding the largest independent set in a graph $G$:

<table>
<thead>
<tr>
<th>Algorithm 2: $MIS_3(G)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 if $G$ has maximum degree 2 then</td>
</tr>
<tr>
<td>2 run the algorithm from the previous question</td>
</tr>
<tr>
<td>3 else</td>
</tr>
<tr>
<td>4 pick a vertex $u$ of degree at least 3;</td>
</tr>
<tr>
<td>5 $withu \leftarrow \phantom{0}$</td>
</tr>
<tr>
<td>6 $withoutu \leftarrow \phantom{0}$</td>
</tr>
<tr>
<td>7 return $\phantom{0}$</td>
</tr>
</tbody>
</table>
(d) Explain how to represent an undirected graph in memory so that the running time of this algorithm, excluding the recursive calls, is $O(n)$.

(e) Give a recurrence for the worst-case running time of this algorithm. Justify your answer briefly.

(f) Give a polynomial $q(x)$ such that the algorithm above runs in time $O(\lambda^n n^2)$, where $\lambda$ is the largest real root of $q(x)$.

(g) Will this algorithm run in time $o(1.5^n)$? Justify your answer briefly.
Problem 4. (Dynamic programming and coffee shops) You decide to spend all Saturday studying for your algorithms midterm. You want to go to a coffee shop, since your roommates will be watching the football game. The problem is that there are two coffee shops you like: Saints’ and Devils’. At any given minute, you prefer to study in the quieter of the two coffee shops, but of course switching from one coffee shop to the other takes time: you lose ten minutes each time you switch.

You’ve figured out, for every minute $i$ of the day, the number of exercises you can get done at Saints’, $s_i$, and at Devils’, $d_i$ (this depends on the typical noise level at that time). So the total amount of studying you’ll get done is

$$\left(\sum_{\text{minutes spent working at Saints'}} s_i\right) + \left(\sum_{\text{minutes spent working at Saints'}} d_i\right).$$

The time you spend switching from one coffee shop to the other is lost. You can start the day at either coffee shop.

More formally, you’re given $s_1, ..., s_n$ and $d_1, ..., d_n$ as input. A plan is a choice of Saints’, Devils’, or “switching” for each of the $n$ minutes in the day. Switching takes a block of ten consecutive minutes. Your goal is to design a dynamic programming algorithm that finds an optimal studying plan.

(a) What are the subproblems for which your algorithm will compute solutions?

(b) Give a recursive formula for each subproblem in terms of other subproblems and the $s_i$’s and $d_i$’s. Justify the formulas briefly. Don’t forget the base cases.

(See over.)
(c) Give pseudocode for an algorithm for this problem. [For full credit, you should find an algorithm that runs in time $O(n)$.] It is probably easiest to separate the algorithm that finds the value of the optimal plan from one that finds the actual plan.

(d) State the running time and space complexity of your algorithm and justify your answer.
— Scratch paper. Do not hand in. —