Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Please also tell us whether you are an undergraduate or a grad student by circling U or G, respectively. Staple this sheet to the top of your homework.
Required Problems

1. Free lunch. [10 Points]

(a) [3 Points] Provide a detailed description of the procedure that computes the longest ascending subsequence in a given sequence of \( n \) numbers. The procedure should use only arrays, and should output together with the length of the subsequence, the subsequence itself.

(b) [4 Points] Provide a data-structure, that store pairs \((a_i, b_i)\) of numbers, such that an insertion/deletion operation takes \( O(\log n) \) time, where \( n \) is the total number of elements inserted. And furthermore, given a query interval \([\alpha, \beta]\), it can output in \( O(\log n) \) time, the pair realizing

\[
\max_{(a_i, b_i) \in S, a_i \in [\alpha, \beta]} b_i,
\]

where \( S \) is the current set of pairs.

(c) [3 Points] Using (b), describe an \( O(n \log n) \) time algorithm for computing the longest ascending subsequence given a sequence of \( n \) numbers.

2. Greedy algorithm does not work for independent set. [20 Points]

A natural algorithm, \textsc{GreedyIndependent}, for computing maximum independent set in a graph, is to repeatedly remove the vertex of lowest degree in the graph, and add it to the independent set, and remove all its neighbors.

(a) [5 Points] Show an example, where this algorithm fails to output the optimal solution.

(b) [5 Points] Let \( G \) be a \((k, k + 1)\)-uniform graph (this is a graph where every vertex has degree either \( k \) or \( k + 1 \)). Show that the above algorithm outputs an independent set of size \( \Omega(n/k) \), where \( n \) is the number of vertices in \( G \).

(c) [5 Points] Let \( G \) be a graph with average degree \( \delta \) (i.e., \( \delta = 2|E(G)|/|V(G)| \)). Prove that the above algorithm outputs an independent set of size \( \Omega(n/\delta) \).

(d) [5 Points] For any integer \( k \), present an example of a graph \( G_k \), such that \textsc{GreedyIndependent} outputs an independent set of size \( \leq \lfloor \text{OPT}(G_k)/k \rfloor \), where \( \text{OPT}(G_k) \) is the largest independent set in \( G_k \). How many vertices and edges does \( G_k \) have? What is the average degree of \( G_k \)?
3. **Greedy Algorithm does not work for Vertex Cover.**  
   **[10 Points]**  
   Extend the example shown in class for the greedy algorithm for Vertex Cover. Namely, for any \( n \), show a graph \( G_n \) with \( n \) vertices, for which the greedy Vertex Cover algorithm, outputs a vertex cover which is of size \( \Omega(\text{Opt}(G_n) \log n) \), where \( \text{Opt}(G_n) \) is the cardinality of the smallest Vertex Cover of \( G_n \).

4. **Greedy algorithm does not work for TSP with the triangle inequality.**  
   **[10 Points]**  
   In the greedy Traveling Salesman algorithm, the algorithm starts from a starting vertex \( v_1 = s \), and in \( i \)-th stage, it goes to the closest vertex to \( v_i \) that was not visited yet.

   (a) **[5 Points]** Show an example that prove that the greedy traveling salesman does not provide any constant factor approximation to the TSP.  
   Formally, for any constant \( c > 0 \), provide a complete graph \( G \) and positive weights on its edges, such that the length of the greedy TSP is by a factor of (at least) \( c \) longer than the length of the shortest TSP of \( G \).

   (b) **[5 Points]** Show an example, that prove that the greedy traveling salesman does not provide any constant factor approximation to the TSP with triangle inequality.  
   Formally, for any constant \( c > 0 \), provide a complete graph \( G \), and positive weights on its edges, such that the weights obey the triangle inequality, and the length of the greedy TSP is by a factor of (at least) \( c \) longer than the length of the shortest TSP of \( G \). (In particular, prove that the triangle inequality holds for the weights you assign to the edges of \( G \).)

5. **Yes. Greedy algorithm does not work for coloring. Really.**  
   **[10 Points]**  
   Let \( G \) be a graph defined over \( n \) vertices, and let the vertices be ordered: \( v_1, \ldots, v_n \). Let \( G_i \) be the induced subgraph of \( G \) on \( v_1, \ldots, v_i \). Formally, \( G_i = (V_i, E_i) \), where \( V_i = \{v_1, \ldots, v_i\} \) and  
   \[
   E_i = \left\{ uv \in E \mid u, v \in V_i \text{ and } uv \in E(G) \right\}.
   \]

   The greedy coloring algorithm, colors the vertices, one by one, according to their ordering. Let \( k_i \) denote the number of colors the algorithm uses to color the first \( i \) vertices.

   In the \( i \)-th iteration, the algorithm considers \( v_i \) in the graph \( G_i \). If all the neighbors of \( v_i \) in \( G_i \) are using all the \( k_{i-1} \) colors used to color \( G_{i-1} \), the algorithm introduces a new color (i.e., \( k_i = k_{i-1} + 1 \)) and assigns it to \( v_i \). Otherwise, it assign \( v_i \) one of the colors \( 1, \ldots, k_{i-1} \) (i.e., \( k_i = k_{i-1} \)).

   Give an example of a graph \( G \) with \( n \) vertices, and an ordering of its vertices, such that even if \( G \) can be colored using \( O(1) \) (in fact, it is possible to do this with two) colors, the greedy algorithm would color it with \( \Omega(n) \) colors. (Hint: consider an ordering where the first two vertices are not connected.)
6. **Greedy coloring does not work even if you do it in the right order.**

    [10 Points]

    Given a graph $G$, with $n$ vertices, let us define an ordering on the vertices of $G$ where the min degree vertex in the graph is last. Formally, we set $v_n$ to be a vertex of minimum degree in $G$ (breaking ties arbitrarily), define the ordering recursively, over the graph $G \setminus v_n$, which is the graph resulting from removing $v_n$ from $G$. Let $v_1, \ldots, v_n$ be the resulting ordering, which is known as **min last ordering**.

    (a) **[5 Points]** Prove that the greedy coloring algorithm, if applied to a planar graph $G$, which uses the min last ordering, outputs a coloring that uses $6$ colors.\footnote{There is a quadratic time algorithm for coloring planar graphs using 4 colors (i.e., follows from a constructive proof of the four color theorem). Coloring with 5 colors requires slightly more cleverness.}

    (b) **[5 Points]** Give an example of a graph $G_n$ with $O(n)$ vertices which is 3-colorable, but nevertheless, when colored by the greedy algorithm using min last ordering, the number of colors output is $n$. (Hint: Extend your solution to 5.)
Practice Problems

1. [10 Points] Even More on Vertex Cover
   (Based on CLRS 35.1-1 and 35.1-4)
   (a) [3 Points] Give an example of a graph for which Approx-Vertex-Cover always yields a suboptimal solution.
   (b) [2 Points] Give an efficient algorithm that finds an optimal vertex cover for a tree in linear time.
   (c) [5 Points] (Based on CLRS 35.1-3) Professor Nixon proposes the following heuristic to solve the vertex-cover problem. Repeatedly select a vertex of highest degree, and remove all of its incident edges. Give an example to show that the professor’s heuristic does not have an approximation ratio of 2. [Hint: Try a bipartite graph with vertices of uniform degree on the left and vertices of varying degree on the right.]

2. [10 Points] Greedy Traveling
   (Based on CLRS 35.2-3) Consider the following closest-point heuristic for building an approximate traveling-salesman tour. Begin with a trivial cycle consisting of a single arbitrary chosen vertex. At each step, identify the vertex \( u \) that is not on the cycle but whose distance to any vertex on the cycle is minimum. Suppose that the vertex on the cycle that is nearest \( u \) is vertex \( v \). Extend the cycle to include \( u \) by inserting \( u \) just after \( v \). Repeat until all vertices are on the cycle. Prove that this heuristic returns a tour whose total cost is not more than twice the cost of an optimal tour.

3. [10 Points] Bin Packing
   (Based on CLRS35-1) Suppose that we are given a set of \( n \) objects, where the size \( s_i \) of the \( i \)-th object satisfies \( 0 < s_i < 1 \). We wish to pack all the objects into the minimum number of unit-size bins. Each bin can hold any subset of the objects whose total size does not exceed 1.
   (a) [4 Points] Prove that the problem of determining the minimum number of bins required is NP-hard.
   (b) [6 Points] Give a heuristic that has an approximation ratio of 2. And give an \( O(n \log n) \) time algorithm for the heuristic.

4. [10 Points] Maximum Clique (Based on CLRS 35-2)
   Let \( G = (V, E) \) be an undirected graph. For any \( k \geq 1 \), define \( G^{(k)} \) to be the undirected graph \( (V^{(k)}, E^{(k)}) \), where \( V^{(k)} \) is the set of all ordered \( k \)-tuples of vertices from \( V \) and \( E^{(k)} \) is defined so that \((v_1, v_2, ..., v_k) \) is adjacent to \((w_1, w_2, ..., w_k) \) if and only if for each \( i \) \( (1 \leq i \leq k) \) either vertex \( v_i \) is adjacent to \( w_i \) in \( G \), or else \( v_i = w_i \).
   (a) [5 Points] Prove that the size of the maximum clique in \( G^{(k)} \) is equal to the \( k \)-th power of the size of the maximum clique in \( G \).
(b) [5 Points] Argue that if there is an approximation algorithm that has a constant approximation ratio for finding a maximum-size clique, then there is a fully polynomial time approximation scheme for the problem.

5. [10 Points] Approx Partition

Problem: Approx Partition

Instance: A finite set $A$ and a “size” $s(a)$ for each $a \in A$, an approximation parameter $\varepsilon > 0$.

Question: Is there a subset $A' \subseteq A$ such that

$$\left| \sum_{a \in A'} s(a) - \sum_{a \in A \setminus A'} s(a) \right| < \varepsilon \sum_{a \in A} s(a)?$$

(a) [5 Points] Suppose $s(a) \in \mathbb{Z}^+$ and $s(a) \leq k$ for each $a \in A$. Give an $O(\frac{nk}{\varepsilon^2})$ time algorithm to find a $\varepsilon$-partition.

(b) [5 Points] Suppose $s(a) \in \mathbb{R}^+$. Give a polynomial time algorithm to find a $\varepsilon$-partition.

6. [10 Points] Just a little bit more about graph coloring

(a) [2 Points] Prove that a graph $G$ with a chromatic number $k$ (i.e., $k$ is the minimal number of colors needed to color $G$), must have $\Omega(k^2)$ edges.

(b) [2 Points] Prove that a graph $G$ with $m$ edges can be colored using $4\sqrt{m}$ colors.

(c) [6 Points] Describe a polynomial time algorithm that given a graph $G$, which is 3-colorable, it computes a coloring of $G$ using, say, at most $n/(5 \log n)$ colors.

7. [10 Points] Splitting and splicing

Let $G = (V, E)$ be a graph with $n$ vertices and $m$ edges. A splitting of $G$ is a partition of $V$ into two sets $V_1, V_2$, such that $V = V_1 \cup V_2$, and $V_1 \cap V_2 = \emptyset$. The cardinality of the split $(V_1, V_2)$, denoted by $m(V_1, V_2)$, is the number of edges in $G$ that has one vertex in $V_1$, and one vertex in $V_2$. Namely,

$$m(V_1, V_2) = \left| \left\{ e \mid e = (uv) \in E(G), u \in V_1, v \in V_2 \right\} \right|.$$

Let $sn(G) = \max_{V_1 \subseteq V} m(V_1, V_2)$ be the maximum cardinality of such a split. Describe a deterministic polynomial time algorithm that computes a splitting $(V_1, V_2)$ of $G$, such that $m(V_1, V_2) \geq sn(G)/2$. (Hint: Start from an arbitrary split, and continue in a greedy fashion to improve it.)

8. Vertex Cover

Problem: Vertex Cover

Instance: A graph $G = (V, E)$ and a positive integer $K \leq |V|$.

Question: Is there a vertex cover of size $K$ or less for $G$, that is, a subset $V' \subseteq V$ such that $|V'| \leq K$ and for each edge $\{u, v\} \in E$, at least one of $u$ and $v$ belongs to $V'$?
(a) Show that VERTEX COVER is NP-Complete. Hint: Do a reduction from INDEPENDENT SET to VERTEX COVER.

(b) Show a polynomial approximation algorithm to the VERTEX-COVER problem which is a factor 2 approximation of the optimal solution. Namely, your algorithm should output a set $X \subseteq V$, such that $X$ is a vertex cover, and $|C| \leq 2K_{opt}$, where $K_{opt}$ is the cardinality of the smallest vertex cover of $G$.

(c) Present a linear time algorithm that solves this problem for the case that $G$ is a tree.

(d) For a constant $k$, a graph $G$ is $k$-separable, if there are $k$ vertices of $G$, such that if we remove them from $G$, each one of the remaining connected components has at most $(2/3)n$ vertices, and furthermore each one of those connected components is also $k$-separable. (More formally, a graph $G = (V, E)$ is $k$-separable, if for any subset of vertices $S \subseteq V$, there exists a subset $M \subseteq S$, such that each connected component of $G_{S \setminus M}$ has at most $(2/3)|S|$ vertices, and $|M| \leq k$.)

Show that given a graph $G$ which is $k$-separable, one can compute the optimal VERTEX COVER in $n^{O(k)}$ time.

9. Bin Packing

**Problem:** BIN PACKING

*Instance:* Finite set $U$ of items, a size $s(u) \in \mathbb{Z}^+$ for each $u \in U$, an integer bin capacity $B$, and a positive integer $K$.

*Question:* Is there a partition of $U$ into disjoint sets $U_1, \ldots, U_K$ such that the sum of the sizes of the items inside each $U_i$ is $B$ or less?

(a) Show that the BIN PACKING problem is NP-Complete.

(b) In the optimization variant of BIN PACKING one has to find the minimum number of bins needed to contain all elements of $U$. Present an algorithm that is a factor two approximation to optimal solution. Namely, it outputs a partition of $U$ into $M$ bins, such that the total size of each bin is at most $B$, and $M \leq k_{opt}$, where $k_{opt}$ is the minimum number of bins of size $B$ needed to store all the given elements of $U$.

(c) Assume that $B$ is bounded by an integer constant $m$. Describe a polynomial algorithm that computes the solution that uses the minimum number of bins to store all the elements.

(d) Show that the following problem is NP-Complete.

**Problem:** TILING

*Instance:* Finite set $R$ of rectangles and a rectangle $R$ in the plane.

*Question:* Is there a way of placing the rectangles of $R$ inside $R$, so that no pair of the rectangles intersect, and all the rectangles have their edges parallel to the edges of $R$?

---

2It was very recently shown (I. Dinur and S. Safra. On the importance of being biased. Manuscript. http://www.math.ias.edu/~iritd/mypapers/vc.pdf, 2001.) that doing better than 1.3600 approximation to VERTEX COVER is NP-Hard. In your free time you can try and improve this constant. Good luck.
(e) Assume that $\mathcal{R}$ is a set of squares that can be arranged as to tile $\mathcal{R}$ completely. Present a polynomial time algorithm that computes a subset $T \subseteq \mathcal{R}$, and a tiling of $T$, so that this tiling of $T$ covers, say, 10% of the area of $\mathcal{R}$.

10. **Minimum Set Cover**

**Problem:** Minimum Set Cover

| Instance: | Collection $C$ of subsets of a finite set $S$ and an integer $k$. |
| Question: | Are there $k$ sets $S_1, ..., S_k$ in $C$ such that $S \subseteq \bigcup_{i=1}^{k} S_i$? |

(a) Prove that Minimum Set Cover problem is NP-Complete

(b) The greedy approximation algorithm for Minimum Set Cover, works by taking the largest set in $X \in C$, remove all the elements of $X$ from $S$ and also from each subset of $C$. The algorithm repeat this until all the elements of $S$ are removed. Prove that the number of elements not covered after $k_{opt}$ iterations is at most $n/2$, where $k_{opt}$ is the smallest number of sets of $C$ needed to cover $S$, and $n = |S|$.

(c) Prove the greedy algorithm is $O(\log n)$ factor optimal approximation.

(d) Prove that the following problem is NP-Complete.

**Problem:** Hitting Set

| Instance: | A collection $C$ of subsets of a set $S$, a positive integer $K$. |
| Question: | Does $S$ contain a hitting set for $C$ of size $K$ or less, that is, a subset $S' \subseteq S$ with $|S'| \leq K$ and such that $S'$ contains at least one element from each subset in $C$. |

(e) Given a set $I$ of $n$ intervals on the real line, show a $O(n \log n)$ time algorithm that computes the smallest set of points $X$ on the real line, such that for every interval $I \in I$ there is a point $p \in X$, such that $p \in I$. 


11. k-CENTER Problem

**Problem:** k-CENTER

**Instance:** A set $P$ of $n$ points in the plane, and an integer $k$ and a radius $r$.

**Question:** Is there a cover of the points of $P$ by $k$ disks of radius (at most) $r$?

(a) Describe an $n^{O(k)}$ time algorithm that solves this problem.

(b) There is a very simple and natural algorithm that achieves a 2-approximation for this cover: First it select an arbitrary point as a center (this point is going to be the center of one of the $k$ covering disks). Then it computes the point that it furthest away from the current set of centers as the next center, and it continue in this fashion till it has $k$-points, which are the resulting centers. The smallest $k$ equal radius disks centered at those points are the required $k$ disks.

Show an implementation of this approximation algorithm in $O(nk)$ time.

(c) Prove that the above algorithm is a factor two approximation to the optimal cover. Namely, the radius of the disks output $\leq 2r_{\text{opt}}$, where $r_{\text{opt}}$ is the smallest radius, so that we can find $k$-disks that cover the point-set.

(d) Provide an $\varepsilon$-approximation algorithm for this problem. Namely, given $k$ and a set of points $P$ in the plane, your algorithm would output $k$-disks that cover the points and their radius is $\leq (1 + \varepsilon)r_{\text{opt}}$, where $r_{\text{opt}}$ is the minimum radius of such a cover of $P$.

(e) Prove that dual problem r-DISK-COVER problem is NP-Hard. In this problem, given $P$ and a radius $r$, one should find the smallest number of disks of radius $r$ that cover $P$.

(f) Describe an approximation algorithm to the r-DISK COVER problem. Namely, given a point-set $P$ and a radius $r$, outputs $k$ disks, so that the $k$ disks cover $P$ and are of radius $r$, and $k = O(k_{\text{opt}})$, where $k_{\text{opt}}$ is the minimal number of disks needed to cover $P$ by disks of radius $r$.

12. MAX 3SAT

**Problem:** MAX SAT

**Instance:** Set $U$ of variables, a collection $C$ of disjunctive clauses of literals where a literal is a variable or a negated variable in $U$.

**Question:** Find an assignment that maximized the number of clauses of $C$ that are being satisfied.

(a) Prove that MAX SAT is NP-Hard.

(b) Prove that if each clause has exactly three literals, and we randomly assign to the variables values 0 or 1, then the expected number of satisfied clauses is $(7/8)M$, where $M = |C|$.

(c) Show that for any instance of MAX SAT, where each clause has exactly three different literals, there exists an assignment that satisfies at least $7/8$ of the clauses.
(d) Let \((U, C)\) be an instance of MAX SAT such that each clause has \(\geq 10 \cdot \log n\) distinct variables, where \(n\) is the number of clauses. Prove that there exists a satisfying assignment. Namely, there exists an assignment that satisfy all the clauses of \(C\).