These are practice problems for the upcoming final exam. You will be given a sheet of notes for the exam. Also, go over your homework assignments. **Warning:** This does not necessarily reflect the length, difficulty, or coverage of the actual exam.

Problem 1. Assume that you developed an algorithm to find the (index of the) $n/3$ smallest element of a list of n elements in $2n$ comparisons.

(a) Using the algorithm (as a black box), give an algorithm, efficient in the worst case, to find the kth smallest element of a list.

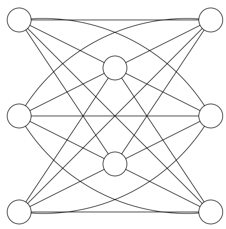
(b) Write down a recurrence for (a bound on) the number of comparisons it executes in the worst case.

(c) Solve the recurrence (using constructive induction). Find the high order term exactly (but you do not need any low order terms).

(d) Using the (black box) algorithm for finding the $n/3$ smallest element and using the ideas and results of Parts (a), (b), and (c), give an efficient algorithm to find (the index of) two elements, the k_1th smallest and the k_2 smallest (for inputs k_1 and k_2). The algorithm description can be very high level and brief.

(e) How many comparisons does it use? Find the high order term exactly (but you do not need any low order terms). Give a brief justification.

Problem 2. A graph is tripartite if the vertices can be partitioned into three sets so that there are no edges internal to any set. The *complete* tripartite graph, $K(a, b, c)$, has three sets of vertices with sizes a, b, and c and all possible edges between each pair of sets of vertices. $K(3, 2, 3)$ is pictured below. A *Hamiltonian* cycle in a graph is a cycle that traverses every vertex exactly once.



(a) For which values of n does $K(1, 1, n)$ have a Hamiltonian cycle. Justify your answer.

Solution: For $n = 1, 2$.

(b) For which values of n does $K(1, n, n)$ have a Hamiltonian cycle. Justify your answer.

Solution: For all $n \geq 1$.

(c) For which values of n does $K(n, n, n)$ have a Hamiltonian cycle. Justify your answer.

Solution: For all $n \geq 1$.

Problem 3. Let $G = (V, E)$ be an undirected graph. A *triangle* is a set of three vertices such that each pair has an edge.

(a) Give an efficient algorithm to find all of the triangles in a graph.

(b) How fast is your algorithm?

Problem 4. Show that you can convert a formula in Conjuctive Normal Form (CNF) where every clause has *at most* three literals, into a new formula where every clause has *exactly* three literals, so that the new formula is satisfiable if and only if the original formula is satisfiable. No variable may occur twice in the same clause.

Solution: Convert $A \vee B$ into $(A \vee B \vee Z)(A \vee B \vee \bar{Z})$
Convert A into $(A \vee Y \vee Z)(A \vee Y \vee \bar{Z})$ $(A \vee \bar{Y} \vee Z)(A \vee \bar{Y} \vee \bar{Z})$

Problem 5. In a graph $G = (V, E)$ edge (x, y) *touches* vertices x and y.

A *newtonian cluster* in a graph $G = (V, E)$ is a subset of the edges such that every vertex is touched by at least one edge. The *size of a newtonian cluster* is the number of edges in the subset.

(a) Give an example of a graph that has a newtonian cluster of size four but not of size three.

Solution: Four isolated edges.

(b) Let C be a newtonian cluster of G. What can you say about the minimum size of C as a function of the number of vertices n? Justify.

Solution: It must be at least $\lceil n/2 \rceil$ since an edge can touch at most two vertices.

(c) A *minimal newtonian cluster* is a newtonian cluster such that if any edge is removed it is no longer a newtonian cluster. Let C be a minimal newtonian cluster of G. What can you say about the maximum size of C as a function of the number of vertices n? Justify.

Solution: It has size $n - 1$. Consider a star graph.

(d) The (decision version of) *newtonian cluster problem* is given a graph $G = (V, E)$ and an integer k does G have a newtonian cluster of size (at most) k. Show that the newtonian cluster problem is in **NP**. What is the certificate?

> **Solution:** The certificate is the set of at most k edges that form the newtonian cluster. Call them $(x_1, y_1), (x_2, y_2), \ldots, (x_p, y_p)$
>
> ```
> function CheckNewtonianCluster
>     {Check that the size of the newtonian cluster is small enough}
>     if p > k then return(FALSE)
>     {Check that every vertex is covered}
>     for i = 1 to n do VertexCovered[i] ← FALSE
>     for i = 1 to p do
>         VertexCovered[x[i]] ← TRUE
>         VertexCovered[y[i]] ← TRUE
>     end for
>     for i = 1 to n do
>         if not VertexCovered[i] then return(FALSE)
>     end for
>     return(TRUE)
> end function
> ```
>
> It is $O(n + k)$ time, which is $O(n^2)$. This is polynomial.

Problem 6. A *vertex cover* in a graph $G = (V, E)$ is a subset of vertices such every edge is incident on at least one vertex of the subset. The *Weighted Vertex Cover Problem (WVCP)* is, given a graph $G = (V, E)$ with integer weights on the vertices, find a vertex cover whose sum of weights is as small as possible. You can assume that the weights are between 1 and n (inclusive).

(a) WVCP is an optimization problem. Define a decision version of WVCP.

> **Solution:** Given a graph $G = (V, E)$ with integer weights on the vertices, and a targe T, is there a vertex cover whose sum of weights on the vertices is at most T?

(b) Show that the decision version is in **NP**. Make sure to state the certificate and give the pseudo code.

> **Solution:** Assume that $A[i, j]$ is the adjacency matrix and $weight[i]$ is the weight of vertex i.
>
> The certificate is the set of vertices in the weighted vertex cover. It can be represented by an array of size n where $incover[i]$ is TRUE if vertex i is in the vertex cover.
>
> ```
> function CheckVertexCover
>     {Check that the weight of the vertex cover is small enough}
>     W ← 0
> ```

```
        for i = 1 to n do
            if incover[i] then W ← W + weight[i]
        end for
        if W > T then return(FALSE)
        {Check that every edge is covered}
        for i = 1 to n do
            for j = i+1 to n do
                if A[i,j] and NOT (incover[i] OR incover[j]) then return(FALSE)
            end for
        end for
        return(TRUE)
    end function
```

It is $O(n)$ time to check that the size of the vertex cover is at most the target T, and $O(n^2)$ time to check that every edge is covered. This is $O(n^2)$, which is polynomial.

(c) Show that if you could solve the optimization version in polynomial time that you could also solve the decision version in polynomial time.

Solution: Run the optimization algorithm on the graph. Sum the weights on the vertices in the vertex cover and check that if the sum is at most T.

(d) Show that if you could solve the decision version in polynomial time that you could also solve the optimization version in polynomial time. HINT: First find the weight of an optimal weighted vertex cover.

Solution:

```
    function SolveOpt
        L ← 0       {Lower bound}
        {Sum the weights of the vertices to get upper bound}
        U ← 0
        for i = 1 to n do U ← U + weight[i]
        {Do binary search to find optimal weight}
        while L<U do
            W ← L+U div 2
            if SolveDecision(G,W) then U ← W
                else L ← W+1
            end if
        end while
        W ← L

        {Find vertex cover by removing vertices from G}
        VC ← ∅
        for i = 1 to n do
```

```
            if SolveDecision(G - vertex i, W - weight[i]) then
                VC ← VC ∪ {i}
                G ← G - vertex i
                W ← W - weight[i]
            end if
        end for
        return(VC)
    end function
```

Problem 7. This problem is more open-ended than you would see on an exam: If you do not know how to play Sudoku, look it up. Normally, Sudoku is played on a 9×9 grid.

(a) Generalize Sudoku to larger grids.

(b) State the (generalized) Sudoku game as a decision problem.

(c) Show that the decision version of (generalized) Sudoku is in NP.

(d) Show that if you can solve the decision version of (generalized) Sudoku in polynomial time, you can solve a (generalized) Sudoku puzzle in polynomial time.