Homework #8

Analysis of Algorithms

Illya Starikov

Due Date: November 27th, 2017

Contents

Listings

Question #1

Theorem 1. *We take the definition of $\mathcal{O}$ as such.*

For $f(n) \in \mathcal{O}(g(n))$, there must exist constants c, where $c > 0$, and there must exist $n_0 \in \mathbb{N}$, where $n_0 \geq 1$, such that

$$|f(n)| \leq c|g(n)| \ \forall n \geq n_0$$

Problem #1.1

Problem Statement. ***Without using limits, but only the definition of*** $\mathcal{O}$ *prove that $81n^3 + 1300n^2 + 300n \in \mathcal{O}(n^5 - 15000n^4 - 10n^3)$, but that $n^5 - 15000n^4 - 10n^3 \notin \mathcal{O}(81n^3 + 1300n^2 + 300n)$. Show all work.*

Proof. In this instance, $f(n) = 81n^3+1300n^2+300n$ and $g(n) \in \mathcal{O}(n^5 - 15000n^4 - 10n^3)$. To prove that $f(n) \in \mathcal{O}(g(n))$, first we must find constants c and n_0.

Take the following, $c = 1$, $n_0 = 15\,000$. Then, for all $n \geq 15000$,

$$81n^3 + 1300n^2 + 300n \ll n^5 - 15000n^4 - 10n^3$$

We see this is true, because

$$\begin{aligned} f(n) &= 81n^3 + 1300n^2 + 300n \\ &\leq 81n^3 + 1300n^3 + 300n^3 \\ &\leq 1681n^3 \\ g(n) &= n^5 - 15000n^4 - 10n^3 \\ &\geq n^3 - 15000^3 - 10n^3 \\ &\geq -14991n^3 \end{aligned}$$

From this we, we see that

$$|f(n)| \leq c|g(n)| \ \forall n \geq n_0$$

Therefore, $f(n) \in \mathcal{O}(g(n))$, and $81n^3+1300n^2+300n \in \mathcal{O}(n^5 - 15000n^4 - 10n^3)$. □

Proof. Suppose not. That is, suppose that $\exists c, n_0$ such that

$$|f(n)| \leq c|g(n)| \ \forall n \geq n_0$$

Therefore, c, n_0 must satisfy the equation

$$\begin{aligned}
81n^3 + 1300n^2 + 300n &\geq c \times (n^5 - 15000n^4 - 10n^3) \\
n^5 &\leq 15000n^4 - 10n^3 + c \times (81n^3 + 1300n^2 + 300n) \\
&\leq \frac{15000}{n} + \frac{10}{n^2} + c \times \left(\frac{81}{n^2} + \frac{1300}{n^3} + \frac{300}{n^4}\right) \\
&\approx 1 + 3.6 \times 10^{-7} c
\end{aligned}$$

From this, the inequality must be satisfied:

$$n \leq \max\left(n_0, \delta + 1 + 3.6 \times 10^{-7} c\right)$$

As we see, there is no values of n_0 and c that will make this inequality hold for all n. This has led us to our contradiction. Therefore

$$n^5 - 15000n^4 - 10n^3 \notin \mathcal{O}\left(81n^3 + 1300n^2 + 300n\right)$$

$\square$

Problem #1.2

Problem Statement. *Let $f(x) = 2\sin^3(x) - 4\sin^2(x)$ and $g(x) = 2\cos^4(x) + 5\cos(x)$. Determine whether $f(x) \in \mathcal{O}(g(x))$ or $g(x) \in \mathcal{O}(f(x))$. You must show all work and base it directly on the definition of $\mathcal{O}$.*

Proof. Because both $\sin x$ and $\cos x$ are sinusoidal, we cannot compare them directly. For the purposes of this problem, we will use a Taylor Series expansions (Equation 1).

$$\sum_{n=1}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n \tag{1}$$

For $f(x)$ and $g(x)$, we get the following expansions.

$$\begin{aligned}
\sum_{n=1}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n &= -4x^2 + 2x^3 + \frac{4x^4}{3} - x^5 - \frac{8x^6}{45} + \frac{13x^7}{60} + \frac{4x^8}{315} + \mathcal{O}\left(x^9\right) \\
\sum_{n=1}^{\infty} \frac{g^{(n)}(a)}{n!}(x-a)^n &= 7 - \frac{13x^2}{2} + \frac{85x^4}{24} - \frac{1093x^6}{720} + \frac{3329x^8}{8064} - \frac{263173x^{10}}{3628800} + \frac{839681x^{12}}{95800320} + \mathcal{O}\left(x^{14}\right)
\end{aligned}$$

From this, we can clearly see that starting at the third term, $g(x)$ grows much more rapidly than $f(x)$. Furthermore, we see that the exponents of $g(x)$ grow by increments of 2, while $f(x)$ only grows by an increments of 1.

Therefore, choosing $c = 1$ and $n_0 = 1$,

$$\begin{aligned}
|f(x)| \leq c|g(x)|\ \forall n \geq n_0 &\implies f(x) \in \mathcal{O}(g(x)) \\
&\implies 2\sin^3(x) - 4\sin^2(x) \in \mathcal{O}\left(2\cos^4(x) + 5\cos(x)\right)
\end{aligned}$$

$\square$

Question #2

For the proceeding problems, the source code is as follows.

Listing 1: Problem #2

```python


class Graph:
    __adjacency_list = {}
    is_directed = False

    def __init__(self, is_directed=False):
        self.__adjacency_list = {}
        self.is_directed = is_directed

    @property
    def vertices(self):
        vertices = set()

        for key, value in self.__adjacency_list.items():
            vertices.add(key)
            vertices = vertices.union(value)

        return vertices

    @property
    def edges(self):
        edges = set()

        for key, values in self.__adjacency_list.items():
            for value in values:
                edges.add((key, value))

        return edges

    @property
    def adjacency_list_raw(self):
        return self.__adjacency_list

    @property
    def adjacency_list(self):
        max_dimension = max(self.vertices) + 1
        adj_list = [[0 for i in range(max_dimension)] for j in range(
    max_dimension)]

        for start, value in self.__adjacency_list.items():
            for destination in value:
                adj_list[start][destination] = 1

        return adj_list

    @property
    def number_of_edges(self):
        return len(self.edges)

    @property
    def number_of_vertices(self):
        return len(self.vertices)

    @property
    def number_of_connected_components(self):
        return len(self.connected_components)

    @property
    def number_of_components_with_euler_path(self):
        connected_components = self.connected_components

        number_of_connected_components_with_euler_path = 0
        for connected_component in connected_components:
            number_of_vertices_with_odd_degree = 0

            for vertex in connected_component:

                if vertex in self.__adjacency_list.keys():
                    if len(self.__adjacency_list[vertex]) % 2 != 0:
                        number_of_vertices_with_odd_degree += 1

            if number_of_vertices_with_odd_degree == 0 or number_of_vertices_with_odd_degree == 2:
                number_of_connected_components_with_euler_path += 1

        return number_of_connected_components_with_euler_path

    @property
    def number_of_components_with_euler_circuits(self):
        connected_components = self.connected_components

        number_of_connected_components_with_euler_circuit = 0
        for connected_component in connected_components:
            number_of_vertices_with_odd_degree = 0

            for vertex in connected_component:

                if vertex in self.__adjacency_list.keys():
                    if len(self.__adjacency_list[vertex]) % 2 != 0:
                        number_of_vertices_with_odd_degree += 1

            if number_of_vertices_with_odd_degree == 0:
                number_of_connected_components_with_euler_circuit += 1

        return number_of_connected_components_with_euler_circuit

    @property
    def connected_components(self):
```

```python
        colors = {}
        for vertex in self.vertices:
            colors[vertex] = "WHITE"

        connected_components = []
        for vertex in self.vertices:
            if colors[vertex] is "WHITE":
                colors[vertex] = "GREY"
                connected_components += [self.__dfs_connected_components_visit(vertex, colors, [])]
                colors[vertex] = "BLACK"

        return connected_components

    def __dfs_connected_components_visit(self, vertex, colors, connected_components):
        connected_components += [vertex]

        if vertex in self.__adjacency_list.keys():
            for adjacent_vertex in self.__adjacency_list[vertex]:
                if colors[adjacent_vertex] is "WHITE":
                    colors[adjacent_vertex] = "GREY"
                    self.__dfs_connected_components_visit(adjacent_vertex, colors, connected_components)
                    colors[adjacent_vertex] = "BLACK"

        return list(set(connected_components))

    def add_edge(self, start, destination):
        if start not in self.__adjacency_list.keys():
            self.add_vertex(start)

        self.__adjacency_list[start].add(destination)

        if not self.is_directed:
            if destination not in self.__adjacency_list.keys():
                self.add_vertex(destination)

            self.__adjacency_list[destination].add(start)

    def add_vertex(self, vertex):
        if vertex not in self.__adjacency_list:
            self.__adjacency_list[vertex] = set()


def read_edges_from_file(filename):
    with open(filename, 'r') as content_file:
        content = content_file.read()

        content = content.replace("(", "").replace(")", "").replace(" ", "")
        vertices_string = content.split("\n")

        vertices = []
        for vertex in vertices_string:
            if vertex != "":
                vertices += [tuple([int(v) for v in vertex.split(",")])]

        return vertices

    return []


def add_list_of_edges_to_graph(graph, edges):
    for (start, destination) in edges:
        graph.add_edge(start, destination)


def dictionary_to_string_description(dictionary):
    result = ""
    for key, value in dictionary.items():
        value_printable = str(value).replace("{", "").replace("}", "")
        result += "{}: {}\n".format(key, value_printable)

    return result


def main():
    np.set_printoptions(threshold=np.nan)

    filename = "input/graph-data.txt"
    vertices = read_edges_from_file(filename)

    graph = Graph(False)
    add_list_of_edges_to_graph(graph, vertices)

    print("The number of vertices in the graph is {}".format(graph.number_of_vertices))
    print("The number of edges in the graph is {}".format(graph.number_of_edges))
    print("Below is the adjacency list for this graph with the vertices sorted.")
    print(np.matrix(graph.adjacency_list))
    print("Below is the adjacency list with of the form Vertex: Neighbor1, Neighbor2, ...")
    print(dictionary_to_string_description(graph.adjacency_list_raw))
    print("The number of connected components of this graph is {}".format(graph.number_of_connected_components))
    print("The number of connected components of the graph that have an Euler path is {}".format(graph.number_of_components_with_euler_path))
    print("I could not do this part.")
    print("The number of connected components of the graph that have an
     Euler circuit is {}".format(graph.
    number_of_components_with_euler_circuits))
    print("I could not do this part.")


if __name__ == "__main__":
    main()
```

Question #3

Problem #3.1

Problem Statement. *Let function q be as below:*

```
def q(n):
   if n <= 0:
       return 1
   elif n < 2:
       return 7
   else:
       return q(n - 1) + q(n - 2)
```

Let function sq be as below:

```
def sq(n):
   if n < 0:
       return 0
   else:
       return sq(n - 1) + q(n)
```

Conjecture a very simple linear relationship between q and sq.

After careful inspection of the sequences, it became quite apparent that there was a relationship between the two sequences is a constant and two terms in the sequence. In other words,

$$s(q) = q(n+2) - 7 \tag{2}$$

These findings can be summarized by Table 1.

Table 1: The values of $q(n)$, $sq(n)$, and $q(n+2)-7$

$q(n)$	$sq(n)$	$q(n+2)-7$
1	1	1
7	8	8
8	16	16
15	31	31
23	54	54
38	92	92
61	153	153
99	252	252
160	412	412
259	671	671
419	1090	1090
678	1768	1768
1097	2865	2865
1775	4640	4640
2872	7512	7512

Problem #3.2

Problem Statement. *Prove, using induction, that the relationship that you conjectured in the previous part is correct. Be sure to set your proof up correctly and to list explicitly the steps of a proof by induction.*

Proof. We will prove the following with induction. To prove this, first we must take into consideration that:

$$sq(n) = \sum_{i=0}^{n+1} q(n) = q(n+2) - 7 \tag{3}$$

Define The Problem For this problem, we wish to prove that $p \underset{\sim}{\propto} q$ by the relation modeled by Equation 2. We map this relationship $\forall n \in \mathbb{Z}^+$.

Check Base Case Two Other Values Refer to Table 1 for the first three values, along with 12 more values.

Prove for all $n > s$, that if $P(n-1)$ is true, then $P(n)$ is true Assume the following inductive hypothesis:

$$\sum_{i=0}^{n-1} q(n) = q(n+1) - 7$$

Then we are going to prove

$$\sum_{i=0}^{n} q(n) = q(n+2) - 7$$

We prove so by such:

$$\begin{aligned}\sum_{i=0}^{n} q(n) &= \sum_{i=0}^{n-1} q(n) + q(n) \\ &= (q(n+1) - 7) + q(n) \\ &= q(n+2) - 7\end{aligned}$$

Conclude The proof Because we have proved the base case and the inductive step, we use inducion to conclude $sq(n) = q(n+2) - 7$.

□

Question #4

Problem Statement. *Let Vec_n be the set of all vectors of length n each component of which comes from range(n). For example, $(3, 0, 2, 2) \in Vec_4$. If $v \in Vec_n$, a quirk is defined as a pair (i, j) such that $0 \leq i < j \leq n-1$, but $v[i] > v[j]$.*

Problem #4.1

Problem Statement. *Create a sample space consisting of the elements of Vec_3. List all the elements of this space. Turn it into a probability space by using the uniform distribution. Finally, compute the average number of quirks in members of Vec_3.*

The sample space S as follows,

$$\begin{gathered}
\text{S} = (0,\ 0,\ 0),\ (0,\ 0,\ 1),\ (0,\ 0,\ 2),\ (0,\ 0,\ 3),\\
(0,\ 1,\ 0),\ (0,\ 1,\ 1),\ (0,\ 1,\ 2),\ (0,\ 1,\ 3),\ (0,\ 2,\ 0),\\
(0,\ 2,\ 1),\ (0,\ 2,\ 2),\ (0,\ 2,\ 3),\ (0,\ 3,\ 0),\ (0,\ 3,\ 1),\\
(0,\ 3,\ 2),\ (0,\ 3,\ 3),\ (1,\ 0,\ 0),\ (1,\ 0,\ 1),\ (1,\ 0,\ 2),\\
(1,\ 0,\ 3),\ (1,\ 1,\ 0),\ (1,\ 1,\ 1),\ (1,\ 1,\ 2),\ (1,\ 1,\ 3),\\
(1,\ 2,\ 0),\ (1,\ 2,\ 1),\ (1,\ 2,\ 2),\ (1,\ 2,\ 3),\ (1,\ 3,\ 0),\\
(1,\ 3,\ 1),\ (1,\ 3,\ 2),\ (1,\ 3,\ 3),\ (2,\ 0,\ 0),\ (2,\ 0,\ 1),\\
(2,\ 0,\ 2),\ (2,\ 0,\ 3),\ (2,\ 1,\ 0),\ (2,\ 1,\ 1),\ (2,\ 1,\ 2),\\
(2,\ 1,\ 3),\ (2,\ 2,\ 0),\ (2,\ 2,\ 1),\ (2,\ 2,\ 2),\ (2,\ 2,\ 3),\\
(2,\ 3,\ 0),\ (2,\ 3,\ 1),\ (2,\ 3,\ 2),\ (2,\ 3,\ 3),\ (3,\ 0,\ 0),\\
(3,\ 0,\ 1),\ (3,\ 0,\ 2),\ (3,\ 0,\ 3),\ (3,\ 1,\ 0),\ (3,\ 1,\ 1),\\
(3,\ 1,\ 2),\ (3,\ 1,\ 3),\ (3,\ 2,\ 0),\ (3,\ 2,\ 1),\ (3,\ 2,\ 2),\\
(3,\ 2,\ 3),\ (3,\ 3,\ 0),\ (3,\ 3,\ 1),\ (3,\ 3,\ 2),\ (3,\ 3,\ 3)
\end{gathered}$$

The average number of quirks for $Vec_3 = 1$.

Problem #4.2

Problem Statement. *Generalize the results of Part (a) to determine the average number of quirks in members of Vec_n. You do not need to list the elements of Vec_n.*

For Vec_n, we have the following:

$$\text{Number of } quirks \in Vec_n = 0.25(n-1)^2$$

Question #5

Problem Statement. *Let G be defined by the following equations: $G(0) = 5$, $G(1) = 15$, $G(2) = 40$, and for $n > 2$, $G(n) = G(n-1) + G(n-2) + G(n-3)$. Write a Python program that implements G directly from the definition. Submit this program as a* `.py` *in your ZIP file. Try to compute $G(500)$ with this program. If you can't compute $G(500)$ directly show how to use dynamic programming to write a more efficient program. Submit this program in your ZIP file.*

Let H be defined by the following equations: $H(0) = 6$, $H(1) = 7$, $H(2) = 8$, and for $n > 2$, $H(n) = H(n-1) - H(n-2) + H(n-3)$. Write a Python program that implements H directly from the definition. Submit this program as a `.py` *in your ZIP file. Try to compute $H(500)$ with this program. If you can't compute $H(500)$ directly show how to use dynamic programming to write a more efficient program. Once you compute $H(500)$ see if you can discover a more efficient program. Submit all of these programs in your ZIP file.*

For the sample input, the following is generated.

$$\begin{aligned} G(500) &= 213\,546\,417\,395\,738\,934\,772\,794\,111\,784\,493\,777\,375\,698\,990\,926\,394\,537\,758\,958\,705\,709 \\ &\quad 75\,006\,915\,873\,346\,065\,610\,819\,405\,691\,957\,713\,899\,246\,806\,099\,708\,361\,322\,154\,885\,470\,915 \\ H(500) &= 6 \end{aligned}$$

For a $\mathcal{O}(c)$ solution to $H(n)$, the following was produced:

$$H(n) = \begin{cases} 6 & \iff n \mod 4 = 0 \\ 7 & \iff n \mod 2 \neq 0 \\ 8 & \iff (n-2) \mod 4 = 0 \end{cases}$$

Listing 2: Problem #5

```python
def static_vars(**kwargs):
    def decorate(func):
        for k in kwargs:
            setattr(func, k, kwargs[k])
        return func
    return decorate


@static_vars(values=[])
def G(n):
    if n > len(G.values) - 1:
        G.values += [None] * (n - len(G.values) + 1)

    if G.values[n] is not None:
        return G.values[n]

    if n == 0:
        return 5
    elif n == 1:
        return 15
    elif n == 2:
        return 40
    elif n > 2:
        G.values[n] = G(n - 1) + G(n - 2) + G(n - 3)
        return G.values[n]


@static_vars(values=[])
def H(n):
    if n > len(H.values) - 1:
        H.values += [None] * (n - len(H.values) + 1)

    if H.values[n] is not None:
        return H.values[n]

    if n == 0:
        return 6
    elif n == 1:
        return 7
    elif n == 2:
        return 8
    elif n > 2:
        H.values[n] = H(n - 1) - H(n - 2) + H(n - 3)
        return H.values[n]


def H_fast(n):
    if n % 2 != 0:
        return 7
    elif (n - 2) % 4 == 0:
        return 8
    elif n % 4 == 0:
        return 6


def main():
    n = 500

    print("G({}) = {}".format(n, G(n)))
    print("H({}) = {}".format(n, H(n)))
    print("H_fast({}) = {}".format(n, H_fast(n)))


if __name__ == "__main__":
    main()
```

Question #6

Problem Statement. *Suppose you are dealing with a dynamic table that follows the following rules:*

1. *The table size doubles when the table is full and another element is added.*
2. *The table contracts to ²⁄₃rds of its size when its load factor falls below ¹⁄₃.*

Using the potential function

$$\Phi(T) = |2 \times T.num - T.size|$$

show that the amortized cost of a TABLE-DELETE for this strategy is bounded above by a constant. For the definition of all terms, consult your textbook.

Proof. We know the amortized cost of the ith operation to be

$$\hat{c}_i = c_i + \Phi_i - \Phi_{i-1} \tag{4}$$

Assuming the ith operation does not trigger a contraction, using Equation 4,

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (2 \times T.num_i - T.size_i) - (2 \times T.num_{i-1} - T.size_{i-1}) \\
&= 1 + (2 \times T.num_i - T.size_i) - (2 \times T.num_i - T.size_i) + 2 \\
&= 3
\end{aligned}$$

However, if the ith operation does trigger a contraction,

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= T.num_i + 1 + (2 \times T.num_i - T.size_i) - (2 \times T.num_{i-1} - T.size_{i-1}) \\
&= T.num_i + 1 + \left(\frac{2}{3}T.size_{i-1} - 2 \times T.num_i) - (T.size_{i-1} - 2 \times T.num_i - 2\right) \\
&= 2
\end{aligned}$$

We see in either situations, the amortized cost of a TABLE-DELETE operation is bounded by $2 \leq \hat{c}_i \leq 3$.

□

Question #7

Problem #7.1

Problem Statement. *Consider the following two sets of coin denominations: {1 cent, 8 cents, 20 cents}, {1 cent, 6 cents, 18 cents}. Describe a greedy algorithm that will express any sum given in pennies in terms of the denominations given in a set. Determine whether the greedy algorithm always produces the optimal solution for these two sets or not. Give a convincing reason for your conclusion.*

A greedy algorithm that always produces a solution could be described as follows:

1. Take $S = \varnothing$ to be the solution set, *coins* to be the input of coins, and T to be the target to reach.
2. Sort *coins* in ascending order.
3. While $\sum_{x \in S} x \neq T$
   a) Take the difference δ to be $T - \sum_{x \in S} x$.
   b) Pick the largest coin c such that $c \leq \delta$[1].
   c) Add this coin to S.

Although this always produces a solution, it does not always produce an optimal solution. For example, suppose our target $T = 24$. With the algorithm described above, then it would take 5 coins ($1 \times 20¢ + 4 \times 1¢$), while the optimal solution would take 4 coins ($4 \times 8¢$).

[1] Because 1 is a valid coin, there will always be a coin to choose from

Problem #7.2

Problem Statement. *Suppose we have an unlimited number of rooms and a finite number of activities, each of which can be staged in any of the rooms. Give an efficient algorithm that can schedule all the activities using the smallest number of rooms.*

A greedy algorithm that always produces solution is as follows:

1. Sort all of the activities in respect to their finish times (i.e., the job that finishes first is the first element). Call this sorted set of activities A.
2. Take a particular room that is not preoccupied R_i and map a particular schedule S_i to it.
3. While not at the end of A:
   a) Pick the first activity as just the first element in the A. Remove this element from A and add it the schedule S_i.
   b) Search A from beginning, finding the first activity who's start time is after the finish time of the previous job. Add this to schedule S_i, remove it from A.
   c) Repeat the last step.
4. If A is empty, the algorithm is finished. If not, repeat Step 2.

Question #8

Problem #8.1

Problem Statement. *Consider the following problem: given a graph, determine whether it can be colored using exactly* 4 *colors. Prove that this coloring problem is NP-Complete.*

Proof. To prove the Four Coloring Problem is NP-Complete, first we must prove that the Four Color Problem is NP. To check a solution in polynomial time, we simple iterate through all edges, and check:

- Make sure that the graph is colored by ≤ 4 colors.
- Iterate though all edges, ensuring that every edge's neighbor is colored by a different color.

Because this is a $\mathcal{O}\left(n^2\right)$ algorithm, we can check a solution in polynomial time.

To prove that this is NP-Complete, we will reduce it to the three coloring problem, as follows.

Supposing we have a graph G, we wish to create a new graph G', such that if G is colorable in three colors, G' can be colored in four. To make G', then we simply add a new vertex and attach it to all of the verticies in G. Therefore, if G is three colorable, then we know G' to be four colorable.

Because we know the Four Coloring Problem is NP and is reducable from the Three Coloring Problem, we conclude that the Four Coloring Problem is NP-Complete. □

Problem #8.2

Problem Statement. *Prove that for all $k > 4$, determining whether a graph can be colored using exactly k colors is NP-Complete.*

Proof. To prove the Four Coloring Problem is NP-Complete, first we must prove that the k-Color Problem is NP. To check a solution in polynomial time, we simple iterate through all edges, and check:

- Make sure that the graph is colored by $\leq k$ colors.
- Iterate though all edges, ensuring that every edge's neighbor is colored by a different color.

Because this is a $\mathcal{O}\left(n^2\right)$ algorithm, we can check a solution in polynomial time.

To prove that this is NP-Complete, we will reduce it to the three coloring problem, as follows.

Supposing we have a graph G, we wish to create a new graph G', such that if G is colorable in three colors, G' can be colored in k. To make G', then we simply add attach k new vertices and attach it to all of the vertices in G. Therefore, if G is three colorable, then we know G' to be k colorable.

Because we know the k-Coloring Problem is NP and is reducable from the Three Coloring Problem, we conclude that the k-Coloring Problem is NP-Complete.

□