CS 473G: Combinatorial Algorithms, Fall 2005
Homework 3

Due Tuesday, October 18, 2005, at midnight

Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:

Starting with Homework 1, homeworks may be done in teams of up to three people. Each team turns in just one solution, and every member of a team gets the same grade.

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Staple this sheet to the top of your homework.

---

1. Consider the following greedy approximation algorithm to find a vertex cover in a graph:

```
GreedyVertexCover(G):
  C ← ∅
  while G has at least one edge
      v ← vertex in G with maximum degree
      G ← G \ v
      C ← C ∪ v
  return C
```

   In class we proved that the approximation ratio of this algorithm is $O(\log n)$; your task is to prove a matching lower bound. Specifically, prove that for any integer n, there is a graph G with n vertices such that GreedyVertexCover(G) returns a vertex cover that is $\Omega(\log n)$ times larger than optimal.

2. Prove that for *any* constant k and *any* graph coloring algorithm A, there is a graph G such that $\boldsymbol{A(G) > OPT(G) + k}$, where $A(G)$ is the number of colors generated by algorithm A for graph G, and $OPT(G)$ is the optimal number of colors for G.

   [Note: This does not contradict the possibility of a constant ***factor*** *approximation algorithm.]*

3. Let R be a set of rectangles in the plane, with horizontal and vertical edges. A *stabbing set* for R is a set of points S such that every rectangle in R contains at least one point in S. The *rectangle stabbing* problem asks, given a set R of rectangles, for the smallest stabbing set S.

   (a) Prove that the rectangle stabbing problem is NP-hard.

   (b) Describe and analyze an efficient approximation algorithm for the rectangle stabbing problem. Give bounds on the approximation ratio of your algorithm.

4. Consider the following approximation scheme for coloring a graph G.

```
TreeColor(G):
  T ← any spanning tree of G
  Color the tree T with two colors
  c ← 2

  for each edge (u, v) ∈ G \ T
      T ← T ∪ {(u, v)}

      if color(u) = color(v)    ⟨⟨Try recoloring u with an existing color⟩⟩
          for i ← 1 to c
              if no neighbor of u in T has color i
                  color(u) ← i

      if color(u) = color(v)    ⟨⟨Try recoloring v with an existing color⟩⟩
          for i ← 1 to c
              if no neighbor of v in T has color i
                  color(v) ← i

      if color(u) = color(v)    ⟨⟨Give up and use a new color⟩⟩
          c ← c + 1
          color(u) ← c

  return c
```

   (a) Prove that this algorithm correctly colors any bipartite graph.

   (b) Prove an upper bound C on the number of colors used by this algorithm. Give a sample graph and run that requires C colors.

   (c) Does this algorithm approximate the minimum number of colors up to a constant factor? In other words, is there a constant α such that $\textsc{TreeColor}(G) < \alpha \cdot OPT(G)$ for any graph G? Justify your answer.

5. In the *bin packing* problem, we are given a set of n items, each with weight between 0 and 1, and we are asked to load the items into as few bins as possible, such that the total weight in each bin is at most 1. It's not hard to show that this problem is NP-Hard; this question asks you to analyze a few common approximation algorithms. In each case, the input is an array $W[1..n]$ of weights, and the output is the number of bins used.

```
NextFit(W[1..n]):
  b ← 0
  Total[0] ← ∞

  for i ← 1 to n
      if Total[b] + W[i] > 1
          b ← b + 1
          Total[b] ← W[i]
      else
          Total[b] ← Total[b] + W[i]

  return b
```

```
FirstFit(W[1..n]):
  b ← 0

  for i ← 1 to n
      j ← 1; found ← False

      while j ≤ b and found = False
          if Total[j] + W[i] ≤ 1
              Total[j] ← Total[j] + W[i]
              found ← True
          j ← j + 1

      if found = False
          b ← b + 1
          Total[b] = W[i]

  return b
```

(a) Prove that NextFit uses at most twice the optimal number of bins.

(b) Prove that FirstFit uses at most twice the optimal number of bins.

(c) Prove that if the weight array W is initially sorted in decreasing order, then FirstFit uses at most $(4 \cdot OPT + 1)/3$ bins, where OPT is the optimal number of bins. The following facts may be useful (but you need to prove them if your proof uses them):

- In the packing computed by FirstFit, every item with weight more than $1/3$ is placed in one of the first OPT bins.
- FirstFit places at most $OPT - 1$ items outside the first OPT bins.