CS4445/CS9544 Analysis of Algorithms II
Assignment 3.

1. A set $B = \{b_a, b_2, \ldots, b_n\}$ of rectangular boxes must be stored in a set of rectangular bins. Each box b_i has length h_i and width w_i. The length h_i of a box is a positive number no larger than 1, i.e. $0 < h_i \leq 1$, and its width w_i can be either 1 or 2. Each bin has length 1 and width 2. The bin-box-packing problem is to determine the minimum number of bins needed to store all the boxes in B. This problem is NP-hard.

   (10 marks) Write a polynomial time approximation algorithm for the problem with constant approximation ratio.

   The algorithm first packs the boxes of width 2 and then the boxes of width 1. When trying to pack a box of width 1 in a bin, the empty space in the bin is divided into two parts or sub-bins of width 1 each; we will call these parts the *left part* and the *right part*. A box of width 1 will first try to be packed in the left part of the bin and if it does not fit there it will try to be packed in the right part of the bin. Boxes are packed in the bins without leaving space between a box and the box beneath it or between a box and the bottom of the bin. The algorithm is as follows.

   Algorithm BoxPacking(B)
   In: Set B of boxes.
   Out: Packing for B into bins of width 2 and length 1.

   Sort the boxes in B in non-increasing order of width
   $S \leftarrow \{\}$ // Set of bins selected by the algorithm
   for each box $b_i \in B$ **do**
   if b_i fits in any bin of S **then**
   Pack b_i in the first bin of S where it fits. If b_i has width 2 it is packed without leaving any space beneath it. If b_i has width 1, we first try to pack it in the left part of the bin; if it does not fit there then we pack it in the right part of the bin.
   else {
   Pack b_i in an empty bin b.
   Add b to S
   }
   Output S

   (25 marks) Compute the approximation ratio of your algorithm. It must be constant.

   First we prove the following claim.

   Claim 1 *In the solution S produced by the algorithm at most two bins store boxes of total area at most 1.*

   Proof: The proof is by contradiction. Assume that 3 bins, B_1, B_2, and B_3 each store boxes of total area at most 1. Let the bins be indexed in the order in which they were

added to the solution S, so bin B_1 was added to S first, bin B_2 was added second and B_3 was the last bin added to S.

Then note that B_2 cannot store any boxes of width 2. This is because if a box b_i of width 2 is stored in B_2, the height of b_i must be at most $1/2$ as the total area of the boxes in B_2 is no larger than 1. But this means that when the algorithm was packing the boxes of width 2, box b_i would have been placed in bin B_1 as the total height of the boxes of width 2 that had been packed in B_1 was at most $1/2$ (this follows because the total area of the boxes in B_1 is at most 1).

The same above argument can be applied to bin B_3. Hence, B_2 and B_3 can only store boxes of width 1. Since the total area of the boxes in B_2 is at most 1, then all the boxes in B_2 fit in its left side. But then, since the total area of the boxes in B_3 is also at most 1, all these boxes fit in the right side of B_2 and thus the algorithm would have packed them there instead of packing them in B_3. □

Note that there is no algorithm that can pack the boxes in the bins such that at most one of the bins stores boxes of total area at most 1. To see this consider two boxes b_1, b_1, where b_1 has width 2 and height $\frac{1}{2} - \epsilon$, for some small value $\epsilon > 0$ and b_2 has width 1 and height $\frac{1}{2} + 2\epsilon$. The two boxes do not fit in a single bin, so they must be packed in 2 bins, each of which will store boxes of total area smaller than 1.

Let $B_1, B_2, \ldots, B_k$ be the bins in the set S computed by the algorithm; then $SOL = k$. Let the bins be indexed so that the last two bins are the only ones that could store boxes of total area at most 1. Let area(B_i) be the total area of the boxes packed in B_i. Then by Claim 1, area(B_i) > 1 for all $i = 1, 2, \ldots, k-2$.

Note that area(B_{k-1})+area(B_k) > 1 as otherwise the boxes in B_{k-1} and B_k would fit in a single bin, and so the algorithm would not have used the two bins. To see this observe that if area(B_{k-1})+area(B_k) ≤ 1 then if we stack the boxes in B_{k-1} and B_k on top of each other the total height of the stack would be at most 1, so all the boxes would fit in a single bin.

Since all the boxes in B are packed in the bins of S then

$$\begin{aligned} \text{total area of the boxes in } B &= \sum_{i=1}^{k} \text{area}(B_i) \\ &= \sum_{i=1}^{k-2} \text{area}(B_i) + B_{k-1} + B_k \\ &> \sum_{i=1}^{k-2} 1 + 1 = k - 1 \end{aligned} \tag{1}$$

Each bin can store boxes of total area 2 and hence if OPT is the minimum number of bins where all the boxes can be packed, then

$$2OPT \geq \text{total area of the boxes in } B > k - 1.$$

Hence,

$$\frac{SOL}{OPT} < \frac{k}{\frac{k-1}{2}} = \frac{2k}{k-1} = 2 + \frac{2}{k-1}. \tag{2}$$

Notice that if $k = 1$ then $SOL = 1$ and the algorithm uses only one bin. In that situation $OPT = 1$ also, and so the approximation ratio for this case is 1.

If $k > 1$ then according to (2) the approximation ratio has its maximum value when $k = 2$ and thus $SOL/OPT < 2 + \frac{2}{2-1} = 4$. Since $SOL/OPT < 4$ regardless of the value of k, then the approximation ratio of the algorithm is 4.

Note. We can tighten the above analysis to get a better bound for the approximation ratio. Note that when $k = 2$ the algorithm uses 2 bins and the optimum solution uses at least one bin. So when $k = 2$, $SOL/OPT < 2$. By (2) when $k > 2$ the largest value of the approximation ratio is $SOL/OPT < 2 + \frac{2}{3-1} = 3$. Therefore, the approximation ratio of the algorithm is no more than 3.

We can tighten the analysis a bit more. If $k = 3$ then the algorithm uses 3 bins and the optimum solution needs to use at least 2 bins. To see this, assume that $OPT = 1$, so all the boxes fit in one bin. If all the boxes have width 2 then the algorithm will also put all the boxes in one bin and k would be equal to 1. Hence, there must be boxes of width 1 and the total area of these boxes is at most 2. Note that when the algorithm packs boxes of width 1, we assume that each bin is split into a left part and a right part. The algorithm fills each part of a bin to at least half of its capacity, so boxes of total area 2 will be packed by the algorithm in at most 4 sub-bins, or in other words in at most 2 bins. Hence, when $k = 3$, $SOL/OPT < 3/2$.

When $k > 3$, by (2) the approximation ratio is at most $SOL/OPT < 2 + \frac{2}{4-1} = 2 + \frac{2}{3}$. Therefore, regardless of the value of k the approximation ratio of the algorithm is at most $2 + \frac{2}{3}$.

(5 marks) Compute the time complexity of your algorithm

To determine whether a box b_i fits in a bin the algorithm proceeds as follows:

- If the box b_i has width 2, then if the total height of the boxes packed in the bin is at most $1 - h_i$ the box fits, otherwise it does not fit; this condition can be tested in $O(1)$ time.
- If the box b_i has width 1, then if the total height of the boxes packed in the left part of the bin or if the total height of the boxes packed in the right part of the bin is at most $1-h_i$ the box fits in the bin, otherwise it does not fit; this condition can be tested in $O(1)$ time.

In each iteration of the **for** loop a box b_i is packed in a bin. In the worst case all the bins in S need to be checked to determine whether b_i fits in any of them. Since the number of bins in S cannot be larger than n, then each iteration of the loop needs $O(n)$ time. The loop is repeated n times, so the time complexity of the **for** loop is $O(n^2)$. Initializing set S and producing the output require $O(1)$ time. Sorting the boxes requires $O(n \log n)$ time and so the time complexity of the algorithm is $O(n^2)$.

2. (*10 marks*) What is the expected number of times that 2 dice need to be tossed until both of them show the same number (i.e. both of them are 1, or 2, or ..., or 6)?

   Let X = number of dice rolls until both dice land on same number. Then,

$$E[X] = \sum_{i=1}^{\infty} i \Pr(X = i).$$

   Now, $X = i$ when the first $i-1$ rolls show two different numbers on the dice –which happens with probability $\frac{30}{36} = \frac{5}{6}$ for each roll–, and the last roll shows the same number on the two dice –which happens with probability $\frac{6}{36} = \frac{1}{6}$. Therefore,

$$\Pr(X = i) = \left(\frac{5}{6}\right)^{i-1} \frac{1}{6}$$

   and so

$$\begin{aligned} E[X] &= \sum_{i=1}^{\infty} i \left(\frac{5}{6}\right)^{i-1} \frac{1}{6} = \frac{1}{6}\frac{6}{5} \sum_{i=1}^{\infty} i \left(\frac{5}{6}\right)^{i} \\ &= \frac{1}{5} \frac{5/6}{(1-5/6)^2} = \frac{1}{5} \times \frac{5/6}{1/36} = \frac{1}{5} \times \frac{5}{6} \times 36 = 6. \end{aligned}$$

3. (*15 marks*) Assume that you have 2 bags, one containing n screws and the other containing n nuts, where n is a multiple of 3. There are 3 types of screws and 3 types of nuts; the first bag contains $n/3$ screws of each type. Each screw has a matching nut in the other bag. The screws and nuts look very much alike, so the only way of deciding whether a screw b_i and a nut n_j match is by trying to screw the nut into the screw.

   Take the screws off the bag in random order and place each one of them in a different bucket. Do the same thing with the nuts so that each bucket has one nut and one screw. Take the screw and nut from the first bucket and see whether they match. Then, take the screw and nut from the second bucket and determine whether they match. Continue doing this with all screws and nuts.

   What is the expected number of screws and nuts that will match? You must explain your answer.

   Place the buckets in a line and index them as $b_1, b_2, \ldots, b_n$. Let X = number of screws and nuts that match, so we now need to compute $E[X]$. For each bucket b_i introduce the indicator random variable X_i, which takes value 1 if b_i contains a matching pair and takes value 0 otherwise. Then

$$X = \sum_{i=1}^{n} X_i$$

and

$$E[X] = \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} \Pr(X_i = 1)$$

So, we need to find $\Pr(X_i = 1)$. Observe that the screw and nut in bucket b_i match if both of them are of the same type; they do not match if they are of different type. Since there are $n/3$ screws and nuts of each type, then the probability that the screw and nut in bucket b_i match is $1/3$. This is because the screw in bucket b_i is of some type t and the nut either has type t or it has any of the other two types; hence, the probability that the nut has type t is $1/3$. Therefore,

$$E[X] = \sum_{i=1}^{n} \frac{1}{3} = \frac{n}{3}.$$

4. Suppose we are given a graph $G = (V, E)$, and we want to label each node u of G with one of k possible labels: $\ell_1, \ell_2, \ldots, \ell_k$, where k is a **constant**. We wish to assign labels to the nodes to maximize the number of edges $(u, v) \in E$ for which the endpoints, u and v, have different labels. We say that an edge (u, v) is *cross* if the colors assigned to u and v are different.

   - (*10 marks*) Write a randomized approximation algorithm that labels the nodes of G in such a way that the expected number of cross edges is at least $\frac{k-1}{k}OPT$ where OPT is the maximum number of edges that can be satisfied.
   - (*25 marks*) Show that the expected number of satisfied edges is at least $\frac{k-1}{k}OPT$.

   Algorithm label$(G = (V, E), \ell)$

   In: Graph $G = (V, E)$ and set ℓ of k labels $\ell_1, \ell_2, \ldots, \ell_k$.

   Out: Number of cross edges.

   for each node u of G **do**

   Label u with a randomly selected label from ℓ

   return number of cross edges.

   Note that selecting a random label for a vertex requires the generation of $\log k$ random bits.

   Let X be the number of cross edges in the solution produced by the algorithm. For every edge e_i let $X_i = 1$ if e_i is cross and $X_i = 0$ otherwise. Then,

$$X = \sum_{i=1}^{m} X_i, \text{ and } E[X] = \sum_{i=1}^{m} E[X_i]$$

   where m is the number of edges. There are k^2 equally likely combinations (ℓ_u, ℓ_v) for the labels of the endpoints of an edge (u, v). Among these possible k^2 combinations of labels k of them assign the same labels to u and v: (ℓ_1, ℓ_1), (ℓ_2, ℓ_2), $\ldots, (\ell_k, \ell_k)$,

and $k^2 - k$ assign them different labels. Then an edge e_i is cross with probability $\frac{k^2-k}{k^2} = \frac{k-1}{k}$. Therefore, $E[X_i] = \Pr(X_i = 1) = \frac{k-1}{k}$ and

$$E[X] = \sum_{i=1}^{m} \frac{k-1}{k} = \frac{(k-1)m}{k}.$$

As for the optimum solution, the maximum number of edges that can be cross is m, so $OPT \leq m$. Hence, the expected number of cross edges in the solution produced by the algorithm is

$$\frac{(k-1)m}{k} \geq \frac{k-1}{k} OPT$$

and thus the approximation ratio of the algorithm is

$$\frac{OPT}{SOL} \leq \frac{m}{\frac{(k-1)m}{k}} = \frac{k}{k-1}.$$