1. A set \( B = \{b_1, b_2, \ldots, b_n\} \) 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.

(5 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 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 \( S \)

(13 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 since the algorithm first packs the boxes of width 2, then 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. 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 \( B_3 \).

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 \( area(B_{k-1}) + area(B_k) \leq 1 \) 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

\[
\text{total area of the boxes in } B = \sum_{i=1}^{k} area(B_i) \\
= \sum_{i=1}^{k-2} area(B_i) + area(B_{k-1}) + area(B_k) \\
> \sum_{i=1}^{k-2} (1 + 1) = k - 1, \text{ by Claim 1} \quad \text{(1)}
\]

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

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

The last inequality follows from (1). Hence,

\[
\frac{SOL}{OPT} < \frac{k}{k-1} = \frac{2k}{k+1} = 2 + \frac{2}{k-1}. \quad \text{(2)}
\]
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.

**Additional Notes. Tighter Analysis**

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

(3 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 \( O(n) \) time as there are only two box widths and so the time complexity of the algorithm is \( O(n^2) \).
2. Let $S$ be an array storing $n$ different integer values; the array is not sorted. Consider the algorithm in the assignment for finding the $k$-largest value in $S$, for a given value $k$.

(12 marks) Compute the expected running time of this algorithm. Assume that a random value $x$ from $S$ can be selected in $O(1)$ time.

Let $X =$ number of iterations of the “repeat” loop. Since every iteration of the loop needs $O(n)$ time, then the running time of the algorithm is $O(nX)$. This value is random, though, so we want to find $E[X]$ first to get the expected running time of the algorithm.

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

Note that $X = i$ if in the first $i - 1$ iterations the algorithm does not find the $k$th largest element, but in the $i$th iteration it does find this element. As all the elements in $S$ are different, the probability that the random element $x$ picked by the algorithm in one iteration of the loop is the $k$-th largest element is $1/n$. Hence

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

and

$$E[X] = \sum_{i=1}^{\infty} i \frac{1}{n} \left( 1 - \frac{1}{n} \right)^{i-1} = \frac{1}{n} \sum_{i=1}^{\infty} i \left( \frac{n-1}{n} \right)^i \left( \frac{n}{n-1} \right) = \frac{1}{n-1} \sum_{i=1}^{\infty} i \left( \frac{n-1}{n} \right)^i$$

Since

$$\sum_{i=1}^{\infty} i a^i = \frac{a}{(1-a)^2}$$

for any $a < 1$, then

$$E[X] = \frac{1}{n-1} \frac{(n-1)/n}{1 - (n-1)/n} = \frac{1}{n} \frac{n^2}{(n-(n-1))^2} = n.$$ 

Therefore, the expected running time of the algorithm is $O(n^2)$.

3. Consider a graph $G = (V, E)$ for which we want to color each node $u$ of $G$ with one of 4 possible colors: $c_2, c_3, c_4$. We say that an edge $(u, v)$ is satisfied if the colors assigned to $u$ and $v$ are different. We wish to assign colors to the nodes to maximize the number of satisfied edges.
• (5 marks) Write a randomized approximation algorithm that assigns a color to the nodes of $G$ in such a way that the expected number of satisfied edges is at least $\frac{3}{4}OPT$ where $OPT$ is the maximum number of edges that can be satisfied.

• (12 marks) Show that the expected number of satisfied edges is at least $\frac{3}{4}OPT$.

**Algorithm** label($G = (V, E)$)

**In:** Graph $G = (V, E)$.

**Out:** Number of satisfied edges.

**for** each node $u$ of $G$ **do**

- Color $u$ with a randomly selected color from $\{c_1, c_2, c_3, c_4\}$

**Output** number of satisfied edges.

Let $X$ be the number of satisfied edges in the solution produced by the algorithm. For every edge $e_i$ let $X_i = 1$ if $e_i$ is satisfied 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 $4^2$ equally likely combinations $(c_u, c_v)$ for the colors assigned of the endpoints of an edge $(u, v)$. Among these possible combinations of colors, 4 of them assign the same color to $u$ and $v$: $(c_1, c_1)$, $(c_2, c_2)$, $(c_3, c_3)$, $(c_4, c_4)$, and $4^2 - 4$ assign them different colors. Then an edge $e_i$ is satisfied with probability $\frac{4^2 - 4}{4^2} = \frac{3}{4}$. Therefore, $E[X_i] = \Pr(X_i = 1) = \frac{3}{4}$ and

$$E[X] = \sum_{i=1}^{m} \frac{3}{4} = \frac{(3)m}{4}.$$

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

$$\frac{3m}{4} \geq \frac{3}{4}OPT$$

and thus the approximation ratio of the algorithm is

$$\frac{OPT}{SOL} \leq \frac{m}{\frac{3}{4}m} = \frac{4}{3}.$$