1. We have seen two heap-build methods in class.
   
   (a) For each method, give one reason why we would prefer it instead of the other.
   
   (b) Given an array with distinct elements, will both methods give the same heap, not
give the same heap, or does it depend on the input? Justify your answer.
   
   (c) We say an algorithm is stable if any pair of elements with equal value appear in the
same order in both the input and output. Assume that both heap-building meth-
ods will not swap elements that have equal value. Which heap-building method
is stable, if any? What about the extraction phase of heapsort that follows heap-
building?

2. You are given $k$ coins, arranged in various stacks. In one “move”, you may choose
any stack and redistribute it into other stacks, but in doing so you can place at most
one coin in each other stack. During the move, you are allowed to make as many new
stacks as you like, but only one coin can go in each.
The reward of one move is precisely the number of coins in the stack that you choose
to redistribute.
   
   (a) Suppose that you get to make some huge number of moves (say, $n$). Develop a
strategy to maximize your total reward. Notice that any strategy provides a lower
bound on the maximum possible reward. The better the strategy, the better (higher)
the lower bound.
   
   (b) Use amortization to obtain an upper bound on the maximum possible reward. If it
helps to think in terms of cost, pretend that your friend is playing this game and you
must pay the reward. In this context, you want to find an upper bound on the worst-
case cost. For amortization, as usual, you want to show that “expensive” moves do not
happen that often, or in other words you want a function that offsets expensive moves,
making their amortized cost smaller. So, think about defining what an expensive move
is, and what a non-expensive move is. It will help to have a good strategy in part (a)
for this.