

- Question 1. Find some good rational numbers for  $a$  and  $b$  such that  $2 = a^2b^2$ . For the purposes of this problem, a pair  $a$  and  $b$  are good if (1) both  $a$  and  $b$  are small and (2) if the fraction representation of  $a$  and  $b$  use integers with a small number of digits for the tops and bottom of the fractions. These two requirements are in partial conflict with each other. For this question, two answers are considered equally good if each answer can beat the other on one of these two considerations. If one answer beats the other on both considerations, then it is the better answer. My calculator claims that  $2^{1/4}$  is 1.189207115.
- Question 2. You roll  $n$  six-sided dice. You get to keep all the dice that have the number 6 up. What is the probability that you get to keep  $k$  dice?
- Question 3. You roll three six-sided dice (each has the integers 1 to 6 on its various faces). You get to keep the dice if the numbers up sum to 6. What is the probability that you get to keep the dice?
- Question 4. Simplify  $\sum_{1 \leq i \leq n} i \lceil \lg i \rceil$ .

The following algorithm follows pointers (represented as integers) in the array  $A$ . Normally, the integer zero is used to represent the empty pointer, but to keep things simple, this algorithm treats zero just like any other integer. Whatever numbers are stored in the array  $A$ , the path starting from index zero will have an initial segment (of length zero or more) followed by a loop (since there are only a limited number of integers that can be stored in a computer word). The algorithm exits as soon as it is sure that the variable  $i$  has gotten to the part of the path containing the loop. In the algorithm,  $d$  is some fixed integer constant that is at least 2.

- Step 1. [Initialize] Set  $i := 0, j := 0$ .
- Step 2. [Outer loop] Set  $i := A[i], j := A[j]$ .
- Step 3. [Inner loop count] Set  $k := d - 1$ .
- Step 4. [Inner loop] Set  $j := A[j]$ . If  $j = i$  then go to Step 7.
- Step 5. [Done inner?] Set  $k := k - 1$ . If  $k > 0$  then go to Step 4.
- Step 6. [Continue outer loop] Go to Step 2.
- Step 7. [Done] Exit.

For example, suppose  $d = 2, A[0] = 1, A[1] = 2, A[2] = 3, A[3] = 4, A[4] = 5$ , and  $A[5] = 4$ . Then the algorithm will do the following:

- Step 1:  $i = 0, j = 0$ ; Step 2:  $i = 1, j = 1$ ; Step 3:  $k = 1$ ; Step 4:  $j = 2$ ; Step 5:  $k = 0$ ;
- Step 6; Step 2:  $i = 2, j = 3$  Step 3:  $k = 1$ ; Step 4:  $j = 4$ ; Step 5:  $k = 0$ ; Step 6; Step
- 2:  $i = 3, j = 5$ ; Step 3:  $k = 1$ ; Step 4:  $j = 4$ ; Step 5:  $k = 0$ ; Step 6; Step 2:  $i = 4,$
- $j = 5$ ; Step 3:  $k = 1$ ; Step 4:  $j = 4$ ; Step 7.

There are six questions about this algorithm. They start out easy and rapidly become harder.

Question 5. For each time the algorithm does Step 3, it will do Step 4 some number of times. Give upper and lower bounds on this number of times. Your answer may depend on  $d$ , but it should be independent of the numbers in  $A$ . (Your upper and lower bounds are with respect to the possible numbers in  $A$ .)

To discuss the running time of the algorithm for a particular value of  $d$  and a particular array  $A$ , define  $i_{n+1}$  to be  $A[i_n]$ , with  $i_0 = 0$ . In the above example we have  $i_0 = 0, i_1 = 1, i_2 = 2, i_3 = 3, i_4 = 4, i_5 = 5, i_6 = 4, i_7 = 5$ , etc. For this example  $i_n = i_{n+2}$  for all  $n \geq 4$ . The sequence  $i_n$  has an initial part that does not repeat (called the tail) and a final part that repeats (called the cycle). Let  $t$  be the length of the tail and  $c$  be the length of the cycle. For the example,  $t = 3$  and  $c = 2$ .

Question 6. When  $m$  is a large enough integer, we have  $i_{m+c} = i_m$ . This in turn implies that  $i_{m+nc} = i_m$  for any positive integer  $n$  (still subject to the condition that  $m$  is large enough). What is the smallest value of  $m$  that is large enough?

Question 7. The first time the algorithm finishes Step 2, the algorithm has  $i = i_1$  and  $j = i_1$ . The second time the algorithm finishes Step 2, it has  $i = i_2$  and  $j = i_{c+1}$ . What are the values of  $i$  and  $j$  the  $m$ -th time that the algorithm finishes Step 2. (Don't worry about the correctness of your formula for the case that  $m$  is too large.)

Question 8. The first time the algorithm finishes Step 2, it has  $i = i_1, j = i_2$ , and  $k = d - 1$ . When  $d = 2$ , the second time the algorithm finishes Step 2, it has  $i = i_2, j = i_4$ , and  $k = d - 1 = 1$ . When  $d \geq 3$ , the second time the algorithm finishes Step 2, it has  $i = i_1, j = i_3$ , and  $k = d - 2$ . Write formulas for  $i, j$ , and  $k$  that are correct for  $m$ -th time that the algorithm finishes Step 4. For final formula should work for any value of  $d$ . You may want to first consider various small values of  $d$  if you find that to be an easier way to organize your thinking.

Question 9. Write an equation whose solution gives the number of times that the value of  $j$  is changed during the running of the algorithm. (This is a proxy for the running time.) Presumably you will want to build on Questions 6 and 8.

Question 10. Solve the equation from Question 9 to obtain the proxy for the running time.