KØBENHAVNS UNIVERSITET

Diskret Matematik og Formelle Sprog: Problem Set 2

Due: Monday February 27 at 12:59 CET .
Submission: Please submit your solutions via *Absalon* as a PDF file. State your name and e-mail address close to the top of the first page. Solutions should be written in LaTeX or some other math-aware typesetting system with reasonable margins on all sides (at least 2.5 cm). Please try to be precise and to the point in your solutions and refrain from vague statements. Make sure to explain your reasoning. *Write so that a fellow student of yours can read, understand, and verify your solutions.* In addition to what is stated below, the general rules for problem sets stated on *Absalon* always apply.
Collaboration: Discussions of ideas in groups of two to three people are allowed and indeed, encouraged—but you should always write up your solutions completely on your own, from start to finish, and you should understand all aspects of them fully. It is not allowed to compose draft solutions together and then continue editing individually, or to share any text, formulas, or pseudocode. Also, no such material may be downloaded from or generated via the internet to be used in draft or final solutions. Submitted solutions will be checked for plagiarism.
Grading: A score of 120 points is guaranteed to be enough to pass this problem set.
Questions: Please do not hesitate to ask the instructor or TAs if any problem statement is unclear, but please make sure to send private messages sometimes specific enough questions could give away the solution to your fellow students, and we want all of you to benefit from working on, and learning from, the problems. Good luck!

1 (70 p) Use mathematical induction to solve the following two problems.

1a (30 p) For which positive integers n does the inequality $3^n > n^3$ hold?

Solution: We can start by trying out some values. For $n = 1$ we have $3^1 = 3 > 1 = 1^3$ and for $n = 2$ we have $3^2 = 9 > 8 = 2^3$. For $n = 3$ strict inequality does not hold, since $3^3 = 3^3$, but for $n = 4$ we have $3^4 = 91 > 64 = 4^3$. We conjecture that strict inequality will keep holding for $n > 4$ and try to prove this using induction.

Base case ***($n = 4$):*** We already verified above for $n = 4$ that $3^n > n^3$ holds.

Induction step: Suppose that $3^n > n^3$ for $n \geq 4$ and consider $n + 1$. We wish to prove that $3^{n+1} > (n + 1)^3 = n^3 + 3n^2 + 3n + 1$. Jumping straight into the calculations, we get

$$\begin{aligned}
3^{n+1} &= 3 \cdot 3^n \\
&> 3 \cdot n^3 && \text{[by the induction hypothesis]} \\
&= n^3 + n \cdot n^2 + n^2 \cdot n \\
&> n^3 + 3n^2 + (3n + 1) && \text{[since } n \geq 4\text{]} \\
&= (n + 1)^3
\end{aligned}$$

which is the desired inequality (and note that when you do calculations like this, we do want to see where you use the induction hypothesis and how, so please make sure to explain this). This concludes the induction step.

It follows by the principle of mathematical induction that the inequality $3^n > n^3$ holds for all positive integers n except $n = 3$.

1b (40 p) For which positive integers n does the inequality

$$1 + \frac{1}{\sqrt{2}} + \frac{1}{\sqrt{3}} + \ldots + \frac{1}{\sqrt{n}} = \sum_{i=1}^{n} \frac{1}{\sqrt{i}} > \sqrt{n}$$

hold?

Solution: We start by investigating this for a couple of small values of n. For $n = 1$ we have $\sum_{i=1}^{1} \frac{1}{\sqrt{i}} = 1 = \sqrt{1}$, so strict inequality does not hold. For $n = 2$ we get that the inequality

$$\sum_{i=1}^{2} \frac{1}{\sqrt{i}} = 1 + \frac{1}{\sqrt{2}} > \sqrt{2} \tag{1}$$

holds if and only if

$$\sqrt{2} + 1 > 2 \tag{2}$$

(which is (1) multiplied by $\sqrt{2}$), and the inequality in (2) clearly holds since $\sqrt{2} > 1$. Since we are in an optimistic mood, we decide to attempt a proof by induction that the inequality also holds for all larger integers.

Base case ***($n = 2$):*** It certainly holds that $\sum_{i=1}^{2} \frac{1}{\sqrt{i}} > \sqrt{2}$, since this is what we just verified.

Induction step: Suppose that $\sum_{i=1}^{n} \frac{1}{\sqrt{i}} > \sqrt{n}$ and consider $n + 1$. By using the induction hypothesis we get

$$\sum_{i=1}^{n+1} \frac{1}{\sqrt{i}} = \sum_{i=1}^{n} \frac{1}{\sqrt{i}} + \frac{1}{\sqrt{n+1}} > \sqrt{n} + \frac{1}{\sqrt{n+1}} \ , \tag{3}$$

and the induction step will clearly follow from (3) if we can prove that

$$\sqrt{n} + \frac{1}{\sqrt{n+1}} > \sqrt{n+1} \tag{4}$$

(in fact, this last inequality more or less has to hold, because otherwise it is not clear what else we could do). If we multiply by (the positive number) $\sqrt{n+1}$, we find that the claimed inequality (4) is equivalent to

$$\sqrt{n(n+1)} + 1 > n + 1 \ , \tag{5}$$

and now it is easy to see that the inequality holds, since for all positive integers n we have $n(n+1) > n^2$ (and taking square roots preserves this inequality). Hence, the induction step goes through.

Appealing to the principle of mathematical induction, we conclude that $\sum_{i=1}^{n} \frac{1}{\sqrt{i}} > \sqrt{n}$ holds for all positive integers $n \geq 2$.

2 (50 p) When Jakob started preparing this course last autumn, he ran the following interesting experiment several times. He picked a random set S consisting of 1012 distinct integers between 1 and 2022 (i.e., in mathematical notation he had $|S| = 1012$ and $1 \leq i \leq 2022$ for all $i \in S$), and then studied the numbers in this set S. Every time, he found that there was some number $c \in S$ that was the sum $c = a + b$ of two other numbers $a, b \in S$ (not necessarily distinct). Can you prove that this was not a coincidence, but that this must always happen?

(*Bonus problem:* What if we require $a, b, c \in S$ to be all different?)

Solution: We are interested in finding numbers $a, b, c \in S$ such that $a + b = c$. This smells a bit like a pigeonhole principle problem, so let us think about what the pigeons and pigeonholes could be.

The pigeonholes tend to be some kind of "fixed" part of the problem, so in this case our working hypothesis could be that they should be the numbers between 1 and 2022, and that the numbers chosen in the set S are the "pigeons" flying to different pigeonholes. But then we need to find at least 2023 pigeons, and we only have 1012 numbers which in addition we are promised will be distinct. A more fundamental problem, perhaps, it also that it is not clear how we would be collecting information about sums $a + b$ of pairs of numbers $a, b \in S$, which would seem to be crucial information for solving the problem.

Another idea might be to let the "pigeons" be all sums $a + b$ of pairs of numbers $a, b \in S$, but then the problem is that these sums could be (much) larger than 2022. It would be nice to have some kind of summation, but to make sure to stay below 2022. Thoughts like these might finally lead us to a solution as follows.

Let us write $S = \{a_1, a_2, a_3, \ldots, a_{1012}\}$ for the numbers a_i ordered so that $a_i < a_{i+1}$ for all i. Now we can note that the numbers $a_i - a_1$ for $i = 2, 3, \ldots, a_{1012}$, are all distinct, positive, and smaller than 2022. This means that if we consider all numbers

$$\{a_i \mid i = 1, 2, \ldots, 1012\} \cup \{a_i - a_1 \mid i = 2, 3, \ldots, 1012\} \ , \tag{6}$$

we have $1012 + 1011 = 2023$ numbers between 1 and 2022. That is, by the pigeonhole principle we must get a collision. Since $\{a_i \mid i = 1, 2, \ldots, 1012\}$ and $\{a_i - a_1 \mid i = 2, 3, \ldots, 1012\}$ are both sets of distinct elements, a colliding pair must have one element from each set. In other words, there must exist indices $j, k \in \{1, 2, \ldots, 1012\}$, $j < k$, such that

$$a_k - a_1 = a_j \tag{7}$$

holds. But then by rewriting (7) we get that $a_k = a_1 + a_j$, which is precisely the type of sum we are looking for.

We draw the conclusion that for Jakob's experiment it has to be the case that one can always find some number $c \in S$ that is the sum $c = a + b$ of two other numbers $a, b \in S$.

Regarding the bonus problem, if we pick the set $S = \{1011, 1012, 1013, \ldots, 2021, 2022\}$, then we see that for all sums of distinct numbers $a, b \in S$, $a \neq b$, we have $a + b > 2022$, so this is a counter-example.

3 (50 p) Another problem that Jakob got interested in when planning the DMFS course last autumn was to find all solutions to the equation

$$x^2 = 2022 + y^2 \tag{8}$$

for integer-valued x and y. Jakob actually worked on this pretty hard, but in the end had to give up because he was not able to solve this equation (and was planning to keep this secret, until he finally had to cobble together a second problem set for the DMFS course and realized that this might be an opportunity to get some help from the students).

Can you help Jakob by proving that in fact there are no solutions to Equation (8) if we insist that x and y should be integers?

Hint: Rewrite the equation and factor it.

Solution: Let us make use of the hint and rewrite the equation as

$$x^2 - y^2 = 2022 \tag{9}$$

which can in turn be factored as

$$(x+y)(x-y) = 2022 \ . \tag{10}$$

Since Equation (10) specifies that 2022 can be written as a particular product of integers, it seems relevant to factor this number, and it is straightforward to compute that

$$2022 = 2 \cdot 3 \cdot 337 \ . \tag{11}$$

Let us make a case analysis depending on whether x and y are odd or even:

Exactly one of** x **and** y **is odd: Suppose that x is odd and y is even. Then $x+y$ and $x-y$ are both odd numbers, and their product could not possibly be even. The same conclusion holds if x is even and y is odd. Hence, there can be no such solutions.

Both of** x **and** y **are either odd or even: Then $x+y$ and $x-y$ are both even numbers, i.e., there are integers m and n such that $x+y = 2m$ and $x-y = 2n$. But if so, we can write the product as $(x+y)(x-y) = 2m \cdot 2n = 4mn$, meaning that it has to be divisible by 4. However, we know from (11) that the number 2022 is not divisible by 4, so this is impossible. Hence there can be no solutions in this case either.

It follows that the equation $x^2 - y^2 = 2022$ has no integral solutions.

Another way of using Equations (10) and (11), which in some sense is of a similar flavour to the solution above, but relies on the concrete prime factorization of 2022 to clinch the argument, is as follows. Without loss of generality, we can assume that x and y are both non-negative (why?), so that $x+y \geq x-y$. In order for (10) to hold, we must have

$$x + y = N_1 \tag{12a}$$

$$x - y = N_2 \tag{12b}$$

for integers N_1 and N_2 such that $N_1 \geq N_2$ and $N_1 \cdot N_2 = 2022$. By adding (12a) and (12b) and dividing by 2 we get

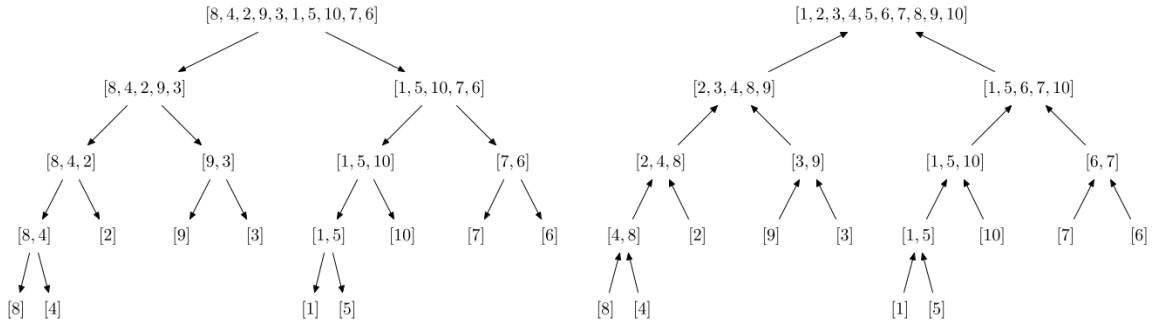
$$x = \frac{N_1 + N_2}{2} \ , \tag{13}$$

which has to be an integer. But since $N_1 \cdot N_2 = 2022 = 2 \cdot 3 \cdot 337$, we see that the only possible cases are $N_1 = 337$, $N_2 = 6$; or $N_1 = 2 \cdot 337$, $N_2 = 3$; or $N_1 = 3 \cdot 337$, $N_2 = 2$; or $N_1 = 2022$, $N_2 = 1$; and in all of these cases $N_1 + N_2$ is odd, meaning that x cannot be an integer.

Yet another possible solution, which is not as elegant but is not wrong if argued with some care, is as follows. Suppose that $x = m$ and $y = n$ is a solution, where we assume without loss of generality that $m > n \geq 0$. Then we have that

$$m^2 - n^2 \geq m^2 - (m-1)^2 = m^2 - (m^2 - 2m + 1) = 2m - 1 \ . \tag{14}$$

Clearly, we cannot have such solutions if $2m - 1 > 2022$, which is the case if $m \geq 1012$. So we can solve the equation by brute force by considering the set $\{(m,n) \mid m, n \in \mathbb{N},\ n < m < 1012\}$ (using a computer, for instance), and showing that no such pair can be a solution (and the search space can be narrowed down further by observing facts like that $m \geq 45$, since we must have $m^2 \geq 2022$). This approach might not give a full score, though (unless you have managed to prove in some very convincing way that you really have checked all possible combinations, and that your calculations can be trusted), and, perhaps more importantly, it will be quite hard to anything similar at the exam...



(a) Tree of recursive calls splitting list.

(b) Bottom-up sorting of split lists.

Figure 1: Merge sort of array $A = [8, 4, 2, 9, 3, 1, 5, 10, 7, 6]$.

4 (60 p) This problem is about the merge sort algorithm that we learned about during the first week of lectures.

4a (30 p) Run merge sort by hand on the array

$$A = [8, 4, 2, 9, 3, 1, 5, 10, 7, 6] \tag{15}$$

as in the handwritten notes for the lectures. Show in every step of the algorithm what recursive calls are made and how the results from these recursive calls are combined. Make sure to explain the merge steps carefully, in particular, the final (and most interesting) merge. (Any clear way of explaining is fine—you do not have to learn how to draw pictures in LaTeX if you do not want to.)

Solution: See Figure 1 for an illustration of what happens in the algorithm.

We first recursively split the list in two part, where the first part is one element larger if the list size is odd, until all lists have size 1. This leads to the split lists in the leaves of the recursion tree in Figure 1a.

In the merge phase we work bottom up from the level above the leaves. Every parent node P takes its two children lists C_1 and C_2, which have previously been sorted, and merges them. We do so by placing to pointers e_1 and e_2 at the start of C_1 and C_2, respectively, and then adding the smallest of these elements to the list under construction after which the corresponding pointer is advanced.

Let us give some examples of how this works, but ignoring the merging of lists of size 1 which is not so exciting (since either the elements are in order or we should just swap them—and, in fact, a more reasonable way to implement merge sort would be to have size ≤ 2 as base case and then swap elements if needed for lists of size 2 rather than making a call to the merge method). For instance, for the two leftmost vertices three levels down we have $e_1 = 4 > 2 = e_2$, so 2 is added first and e_2 advances, after which we discover that there is nothing left in the right list C_2. Therefore, we just append C_1 to obtain the list t $[2, 4, 8]$. When $[2, 4, 8]$ is merged with $[3, 9]$, then 2 is the smallest element and goes first, but then the next element 3 is taken from the other list. After this, we take 4 and 8, which are both smaller than 9 from the right list. This empties the left list, and so we append everything left in the right list, i.e., the element 9. As a result, we get the list

$[2, 3, 4, 8, 9]$. The recursive calls in the right subtree of the root are dealt with in the same way, although the only rearrangement than happens is that 6 and 7 swap places, since all other elements are already in order.

In the final step, we need to merge $[2, 3, 4, 8, 9]$ and $[1, 5, 6, 7, 10]$—note that we are specifically asked in the problem statement to explain this merge step carefully, and so we will make sure to do so. Here we start with $e_1 = 2 > 1 = e_2$, so 1 goes first, updating e_2 to 5. Now we have $e_1 = 2 < 5 = e_2$, so we add 2 to the list under construction, updating e_1 to 3. In the same way we add 3 and 4 from the left-hand side. After this we reach $e_1 = 8 > 5 = e_2$, however, and so we add 5 from the right list, followed by 6 and 7, after which $e_2 = 10 > 8 = e_1$. Since 10 is the largest element, we will now empty the left list, and finally add 10 from the right list. This produces the sorted list at the root of the tree in Figure 1b.

4b (30 p) The reason merge sort runs in time essentially $n \log_2 n$, as explained in class, is that the list of elements to be sorted is split into two lists of half the size in each recursive call, which means that after $\log_2 n$ recursive calls we get lists of size 1 that are already (vacuously) sorted. Merging all the lists takes a linear amount of work for each level of recursion, meaning that the total running time scales like $n \log_2 n$.

When Jakob thought more about this, he realized that if we would make a three-way split in each recursive call into three lists of a third of the size, then the recursion tree would terminate after $\log_3 n < \log_2 n$ levels, and the merging would still cost a linear amount of work per level, resulting in running time $n \log_3 n < n \log_2 n$. Or if in every recursive call we subdivided the list into four lists of equal size, then the recursion tree would only be of depth $\log_4 n$, and the total work of splitting and merging together the sorted lists would be $n \log_4 n < n \log_3 n$.

Pushing this idea to the limit, after a lot of calculations Jakob found that if in every level we do a $\sqrt{n}$-wise split of the list of size n into $\sqrt{n}$ lists of size $\sqrt{n}$, then the total running time of splitting and merging would be $n \log_{\sqrt{n}} n = 2 \cdot n = \mathrm{O}(n)$, meaning that we have a linear-time algorithm! Are you buying Jakob's arguments for this, or can you see any problems with his linear-time sorting algorithm? Please explain clearly what arguments you have to support or refute Jakob's claims.

Solution: No, Jakob has not designed a linear-time sorting algorithm. (As you might learn in later courses, if we only use comparisons of elements to sort, then it is in fact impossible to do better than $\mathrm{O}(n \log n)$, but this is not part of what we learn at the DMFS course.)

It is true that for k-wise splits the depth of the recursion tree will be $\log_k n$ (rounded up), and this number gets smaller for larger k. But for constant $k > 2$ the difference is only a constant factor, and merging k lists at a time is more complicated than merging just two lists, so we will lose in the constant factors in the merge steps. Thus, it is not clear how this would make the algorithm run faster.

And this only gets worse if k is chosen to be growing merging k lists will mean keeping track of which of k list heads is smallest at every step. It is very hard to see how to do this in linear time.

5 (70 p) February March is the worst time of the year for Jakob's pet frog. Jakob is busy teaching in Copenhagen and the frog is all alone at home in Lund and has to find other entertainment during long, dark afternoons and evenings.

One day, the frog came up with a new game with the following rules: for some positive integer n, make jumps on a straight line of lengths $i = 1, 2, \ldots, n$, each jump being either to the

left or the right, and try to end up in the same position. The frog immediately got captivated by this fascinating game, and quickly figured out for $n = 3$ that it is possible to jump 1 step right, 2 more steps right, and then 3 steps left to get back to the starting position. For $n = 4$, one possible strategy is to jump 1 step right, 2 steps left, 3 more steps left, and then 4 steps right to get back to the starting position. For $n = 5$, however, things started to get a little bit too complicated for froggy...

At this point, the frog remembered what Jakob often says, namely that the proper way of approaching situations like this is to model the problem mathematically, design an algorithm for it, prove that the algorithm is correct, and finally analyse the complexity of the algorithm. But exhausted after a day full of intellectual hard labour, the frog was not able to make any further progress.

Can you help the frog to solve this problem? That is:

1. Design an algorithm that, given a positive integer n, prints out a sequence of left and right jumps of increasing length $1, 2, \ldots, n$ that will end up in the starting position. (If there is no such sequence for a given n, then the algorithm should of course report this.)

   Sample output for your algorithm could look something like

   ```
   Jumping sequence of length 3:
   Jump length 1 right
   Jump length 2 right
   Jump length 3 left
   ```

   or

   ```
   Jumping sequence of length 4:
   Jump length 1 right
   Jump length 2 left
   Jump length 3 left
   Jump length 4 right
   ```

   for the two examples above.

2. Prove that the algorithm that you have designed is correct.

3. Analyze the time complexity of your algorithm.

Partial results can also give points on this problem (as for any other problem). In particular, if you cannot quite design an algorithm, but still can say mathematically interesting things about this problem (like for which n it might or might not be possible to construct left-right jumping sequences that return to the starting point), then such insights will be rewarded generously if they are written down clearly and with proper explanations.

Hint: Continue where the frog left off and work out examples for some small values of n, and try to find some patterns.

Solution: Let us follow the hint and work out some more small cases, where we consider the jumping lengths in decreasing order—clearly, this does not matter, since if we will find a solution, then the frog can just perform the appropriate jumps in reverse order. Let us also think of jumping right as jumping in the positive direction and left as the negative direction.

***Length** 5:* We observe that the jumps of length 5 and 4 cannot be in the same direction, because if so the frog will be 9 steps away from the starting point, and $1+2+3=6$. So the longest two jumps are in different directions, say length 5 right and length 4 right, after which the frog is at position $+1$. The length-3 jump must then be left to position -2, since otherwise the frog would be at distance 4, which is to large, and the length-2 jump goes right to position 0. But then the final jump will take frog away from the starting point. Hence, there is no solution for $n=5$.

***Length** 6:* By reasoning in exactly the same way, we find that the jumps of length 6 and 5 must be in opposite directions, after which the frog is at position $+1$ without loss of generality. If the length-4 jump goes right to position $+5$, then the next two jumps have to go left, after which the frog is at position 0 just before the final jump. If instead the length-4 jump goes left to position -3, then the next jump has to go right to position 0. Again, there is no way that the final two jumps of length 2 and 1 will bring froggy back to position 0. We conclude that there is no solution for $n=6$ either.

***Length** 7:* For length 7, an interesting observation is that the frog can jump 7 steps right, then $6+5$ steps left, and finally 4 steps right to get back to where it started. And now we can use the solution for $n=3$.

***Length** 8:* In exactly the same way, for length 8 one can jump 8 steps right, then $7+6$ steps left, and finally 5 steps right to get back to the starting position, after which the solution for $n=4$ can be used.

If we think a little bit more about this, we realize that if we set $r = n \bmod 4$, then for $r=0$ we can use the idea for length 8 to go from n to $n-4, n-8, \ldots$, taking 4 jumps at a time as described above, all the way down to $n=0$ to find a solution. And for $r=3$ we do the same until we reach $n=3$, which has a solution. So there are solutions for all n such that $n \bmod 4 \in \{0,3\}$.

If we wish, we can prove this more formally by induction. This is not really needed if the explanation is clear enough, but let us write an inductive proof anyway just to show how this can be done.

Claim: If $n \bmod 4 \in \{0,3\}$, then the frog has a valid length-n jumping sequence.

***Base case** ($n \leq 3$):* For $n=0$ the claim is vacuously true—the frog cannot jump, but is already in the desired position. For $n=3$, the frog can jump $1+2$ steps right and then 3 steps left, as described above.

Induction step: Suppose the claim is true for all sequence of length strictly smaller than n, and suppose $n \bmod 4 \in \{0,3\}$. Then $(n-4) \bmod 4 = n \bmod 4 \in \{0,3\}$, so the frog can first use a valid length-$(n-4)$ jumping sequence. After this, the frog jumps $n-3$ steps right, $n-2$ steps left, $n-1$ steps left, and finally n steps right again. Since $(n-3)+n = (n-2)+(n-1)$, this will again bring the frog back to the starting position. This concludes the induction step, and the claim follows by induction.

But what about n such that $n \bmod 4 \in \{1,2\}$? To see that there cannot exist any solution for such n, we can argue as follows. If there is a solution, then there is some way to choose the signs so that we get the summation equality $\sum_{i=1}^{n} \pm i = 0$. But observe that if $n \bmod 4 \in \{1,2\}$, then this sum contains an odd number of odd numbers (and, obviously, $+i$ is odd iff and only if $-i$ is odd, so it does not matter for this argument how we choose the signs). But if we sum up a set of even numbers plus an odd number of odd numbers, then that sum is an odd number, and hence not 0. So this shows that there cannot exist any valid length-n jumping sequences for $n \bmod 4 \in \{1,2\}$.

Another way to see that there are no solutions for $n \bmod 4 \in \{1, 2\}$, which is really the same as above but expressed in a different way, is the following: The total length jumped by the frog is $\sum_{i=1}^{n} i = n(n+1)/2$. Exactly half of this should be in one direction, and half should be in the other direction. This means that

$$\frac{\sum_{i=1}^{n} i}{2} = \frac{n(n+1)}{4} \tag{16}$$

had better be an integer. But one of the numbers n and $n+1$ is guaranteed to be odd, meaning that the other one has to be divisible by 4. Requiring that $4|n$ or $4|(n+1)$ is the same thing as saying that $n \bmod 4 \in \{0, 3\}$.

To summarize what we have shown so far: **The frog can find a valid length-n jumping sequence for a positive integer n if and only if $n \bmod 4 \in \{0, 3\}$.**

It remains to provide an algorithm. Using the ideas above, we can construct a algorithm `construct_jumping_sequence` as follows, where just for the fun of it we do not present pseudo-code but actually executable Python code (which is essentially pseudo-code if you just ignore the percent signs for formatting in the print commands):

```
import sys

def construct_jumping_sequence (n):
    r = (n % 4)
    if (r == 1 or r == 2 or n <= 0):
        print ('No jumping sequence for length %d' % n)
    else:
        print ('Jumping sequence of length %d:' % n)
        jump_recursive (n)

def jump_recursive (n):
    if (n >= 4):
        jump_recursive (n - 4)
        print ('Jump length %d right' % (n - 3))
        print ('Jump length %d left ' % (n - 2))
        print ('Jump length %d left ' % (n - 1))
        print ('Jump length %d right' %  n)
    elif (n == 3):
        print ('Jump length 1 right')
        print ('Jump length 2 right')
        print ('Jump length 3 left ')

if __name__ == '__main__':
    n = int (sys.argv[1])
    construct_jumping_sequence (n)
```

It would also be eminently possible to write the code with an iterative loop instead, but we are taking this opportunity to illustrate how elegant recursive algorithms can be.

For the time complexity analysis, note that everything in `construct_jumping_sequence` takes constant time except possibly the call to `jump_recursive`. In every call to `jump_recursive` we do a constant amount of work, and we decrease n by 4. Hence there will be $O(n)$ calls to `jump_recursive` all in all, and since each call runs in constant time this means that the total running time is also $O(n)$.

Just for fun, let us conclude with some actual examples of outputs when we run the Python code above. If we ask for a jumping sequence of length 10, the result is

```
$ python frogjump-pset2.py 10
No jumping sequence for length 10
```

but for length 11 we get

```
$ python frogjump-pset2.py 11
Jumping sequence of length 11:
Jump length 1 right
Jump length 2 right
Jump length 3 left
Jump length 4 right
Jump length 5 left
Jump length 6 left
Jump length 7 right
Jump length 8 right
Jump length 9 left
Jump length 10 left
Jump length 11 right
```

and for length 12 the output is

```
$ python frogjump-pset2.py 12
Jumping sequence of length 12:
Jump length 1 right
Jump length 2 left
Jump length 3 left
Jump length 4 right
Jump length 5 right
Jump length 6 left
Jump length 7 left
Jump length 8 right
Jump length 9 right
Jump length 10 left
Jump length 11 left
Jump length 12 right
```