# PANCAKE FLIPPING AND SORTING ALGORITHM COMPLEXITY

Chanez ARROUM, Aleksandra KOVIAZINA, Justine XU, Melissa BENKHODJA and Lina CHENNI

## Contents

*Date*: March 23, 2024.

## 1. Abstract

*The pancake flipping problem* presents a unique challenge in sorting a stack of pancakes that vary in size, using only a spatula to flip sections of the stack.
For over three decades, researchers explored the complexities of the pancake problem. This exploration involves investigating optimal arrangements for any given stack of pancakes and analyzing the worst-case scenarios.

The pancake flipping problem has many applications in computer science, specifically in parallel computing to develop efficient sorting algorithms. In biology, it serves as a model for understanding genomics rearrangements. In robotics, the limited flipping operations can inspire algorithms for more effective object manipulation. Additionally, it has applications in education to teach practical algorithmic concepts beyond its theoretical aspect.

In the context of this research paper, we explored two different approaches. The first one focuses on the two largest pancakes in a stack, while the second is based on a stack of pancakes with sizes 0 and 1.

## 2. Introduction

The pancake sorting problem was first formulated in 1975 by mathematician *Jacob Eli Goodman* [3] and computer scientist *William Thomas Trotter* [4], who submitted it to the *American Mathematical Monthly* [5] under the pseudonym *"Harry Dweighter"* :

> *"The chef in our place is sloppy, and when he prepares a stack of pancakes, they come out all different sizes. Therefore, when I deliver them to a customer, on the way to the table I rearrange them (so that the smallest winds up on top, and so on, down to the largest at the bottom) by grabbing several from the top and flipping them over, repeating this (varying the number I flip) as many times as necessary. If there are n pancakes, what is the maximum number of flips (in terms of n) that I will ever have to use to rearrange them ? "*

The stack of pancakes is represented as permutations, where each flip results in the inversion of a prefix of varying lengths. This computational problem is recognized in computer science as prefix reversal sorting. It leads to two closely connected challenges: firstly, the development of an algorithm capable of efficiently sorting any permutation with the fewest possible flips, and secondly, determining the maximum number of flips necessary to sort a permutation of size n.

Many variations of this algorithmic problem have emerged, including the "burnt pancake" version proposed by *William Gates* [11] and *Christos Papadimitriou* [1] in 1979.

## 3. State of art

### 3.1. **William Gates [12] and Christos Papadimitriou [1].** [11]

In 1979, Gates and Papadimitriou defined an algorithm based on eight cases that describe the relative position and orientation of certain blocks and singletons in permutations.

"A block in a permutation $\pi = \pi_1, \pi_2, \ldots, \pi_n$ is a maximal length sublist $\pi_r, \pi_{r+1}, \ldots, \pi_s$ such that there is an adjacency between $\pi_i$ and $\pi_{i+1}$ for all $i$ $(r \leq i < s)$. A block has a distinct initial and final element, which we call its endpoints"

An element not occurring in a block is called a singleton. The consecutive element for each endpoint can either be a singleton or an endpoint of another block.

The eight cases are categorized by whether the initial object is a singleton or a block and whether elements consecutive with the initial object are singletons or blocks. Every permutation belongs to at least one of these eight cases.

To use the algorithm, we first determine which case(s) the permutation belongs to and then perform the prefix reversals indicated for the case. After performing the specified prefix reversals, one again has a permutation, and the process is repeated.

Let $s1$ be the first element of our list. At least one of the following 8 cases applies.

(1) $s1$ is free, and $s2$ is also free. Two singletons are eliminated, and one block is created in one step.

| | | | |
|---|---|---|---|
| $s1$ | $\ldots$ | $s2$ | $\ldots$ |
| $\ldots$ | $s1s2$ | $\ldots$ | $\ldots$ |

(2) $s1$ is free, and $s2$ is the first element of a block. Singleton at the beginning of the permutation is consecutive with the left endpoint of a block. One singleton is eliminated in one step.

| | | | |
|---|---|---|---|
| $s1$ | $\ldots$ | $s2b$ | $\ldots$ |
| $\ldots$ | $s1s2b$ | $\ldots$ | |

(3) $s1$ is free, but both $s2$ and $s3$ are the last elements of blocks. One singleton and one block are eliminated in 4 steps.

| | | | | | |
|---|---|---|---|---|---|
| $s1$ | $\ldots$ | $bs2$ | $\ldots$ | $bs3$ | $\ldots$ |
| $\ldots$ | $\ldots$ | $bs3s1s2b$ | $\ldots$ | | |

(4) $s1$ is in a block, and $s2$ is free. One singleton is eliminated in one step.

| | | |
|---|---|---|
| $s1b$ | $\ldots$ | $s2 \ldots$ |
| $\ldots$ | $bs1s2$ | $\ldots$ |

(5) $s1$ is in a block, and $s2$ is the first element of a block. One block is eliminated in one step.

| | | | |
|---|---|---|---|
| $s1b$ | $\ldots$ | $s2b$ | $\ldots$ |
| $\ldots$ | $bs1s2b$ | $\ldots$ | |

(6) $s1$ is the first element of a block that ends with $s2$ ,$s3$ is the last element of another block and $s4$ is free. One singleton and one block are eliminated in 4 steps.

| | | | | |
|---|---|---|---|---|
| $s1bs2$ | $\ldots$ | $bs3$ | $\ldots$ | $s4$ |
| $\ldots$ | $\ldots$ | $s4bs1b$ | $\ldots$ | |

(7) $s1$ is in a block with last element $s2$, $s3$ is the first(or the last) element in a block. One block is eliminated in two steps.

| | | | |
|---|---|---|---|
| $s1bs2$ | $\dots$ | $s3b$ | $\dots$ |
| $\dots$ | $s1bs2s3b$ | $\dots$ | |

(8) None of the above. The list has $n-1$ adjacencies ; halt.

**Theorem 3.1.** *The Algorithm defined before creates a permutation with $n-1$ adjacencies by at most $\frac{5n-7}{3}$ moves.*

3.1.1. *proof Gates and Papadimitriou* [11].

*Proof.* Firstly, if we've got a permutation $\pi$ with fewer than $n-1$ adjacent pairs, then one of the situations 1 through 7 has to be happening. So, the algorithm won't stop until it's made $n-1$ adjacent pairs.

Of course, the algorithm will stop eventually. Because each time it goes through the main loop, it adds at least one new adjacent pair and doesn't take any away. So, it's a sure thing that it will stop.

Let's represent the number of actions of type 1 - $x_1$, of type 2 - $x_2$, and the number of actions of type 3 and 6 - $x_3$, because the number of flips and configurations are similar, to type 4 - $x_4$, and of type 5 and 7 - $x_5$ and $x_7$ respectively. Thus the total number of flips is given by

$$z = x_1 + x_2 + 4x_3 + x_4 + x_5 + 2x_7. \tag{1}$$

Where $x_i$ is multiplied by the number of flipping involved in the action of type $i$. We noticed that the author made a little error in the algorithm by switching the values of the number of flips for actions of types 5 and 7. Below, in the algorithm, there is 1 flip for type 5 and 2 for the 7. So, we corrected this mistake in our work. Anyway, it does not influence the final conjecture.

Cases 3 and 6 can be divided into four special cases, according to what happens in the flipping. In the following figure, we give you an example of the third case :

| | | | | | |
|---|---|---|---|---|---|
| $s1$ | M | $bs2$ | $\dots$ | $bs3$ | L |
| $s2b$ | M | $s1$ | $\dots$ | $bs3$ | L |
| M | $bs2s1$ | $\dots$ | $bs3$ | L | |
| $s3b$ | $\dots$ | $s1s2b$ | M | L | |
| $\dots$ | $bs3s1s2b$ | M | L | | |

Where M and L are the same things that "...". But we need to pay attention because when we do the third flip, M and L find themselves side by side. It can influence the number of adjacencies, so the author separates 4 special cases :

(1) be non-adjacent,
(2) form a new block,
(3) merge a block with a singleton,
(4) merge two blocks.

Accordingly, we distinguish among these subcases by writing $x_3 = x_{31} + x_{32} + x_{33} + x_{34}$.

The following table resume information for all actions

Now, since each action increases the number of adjacencies as indicated in the Table, the total number of adjacencies in the conclusion of the algorithm is

$$n-1 = a + x_1 + x_2 + 2x_3 + 3x_{31} + 3x_{32} + 3x_{33} + 3x_{34} + x_4 + x_5 + x_7. \tag{2}$$

| Action | 1 | 2 | 31 | 32 | 33 | 34 | 4 | 5 | 7 |
|---|---|---|---|---|---|---|---|---|---|
| Number of flips | 1 | 1 | 4 | 4 | 4 | 4 | 1 | 1 | 2 |
| Increase in adjacencies | 1 | 1 | 2 | 3 | 3 | 3 | 1 | 1 | 1 |
| Increase in number of blocks | 1 | 0 | -1 | 0 | -1 | -2 | 0 | -1 | -1 |

Where $a$ is the number of adjacencies in the initial permutation $\pi$.

Finally, if $b$ is the number of blocks in $\pi$, we have

$$b + x_1 - x_{31} - x_{33} - 2x_{34} - x_5 - x_7 = 1. \tag{3}$$

Because each type of action increases or decreases the number of blocks as indicated in the table, we start with $b$ blocks and end up with 1 block. Also, notice that $b \leq a$, whereby (2) becomes

$$n - 1 \geq b + x_1 + x_2 + 2x_{31} + 3x_{32} + 3x_{33} + 3x_{34} + x_4 + x_5 + x_7. \tag{4}$$

Thus, any possible application of the algorithm would, at worst, maximize (1) subject to (3) and (4) above. The author claims that the maximum is achieved for the values

$$x_1 = \frac{n+1}{3}, \quad x_2 = 0, \quad x_3 = x_{31} = \frac{n-2}{3}, \quad x_4 = x_5 = x_7 = b = 0,$$

yielding a value of $z$ equal to $\frac{5n-7}{3}$. To show the claim, the author recalls the duality theorem [2], stating essentially that this maximum value equals the minimum value of the dual linear program :

Maximize z = $\sum_{j=1}^{n} c_j * x_j$, subject to :

$$\sum_{i=1}^{n} a_{ij} * x_j \leq b_i$$

Associated with this primal problem there is a corresponding dual problem given by Minimize v = $\sum_{j=1}^{m} b_i * y_i$, subject to :

$$\sum_{i=1}^{m} a_{ij} * y_i \geq c_j$$

So for our problem, we have the following dual problem.

Minimize $\omega = \epsilon_2 + (n-1)\epsilon_3$, subject to the inequalities

$$\begin{aligned}
\epsilon_2 + \epsilon_3 &\geq 1 \\
\epsilon_3 &\geq 1 \\
-\epsilon_2 + 2\epsilon_3 &\geq 4 \\
3\epsilon_3 &\geq 4 \\
-\epsilon_2 + 3\epsilon_3 &\geq 4 \\
-2\epsilon_2 + 3\epsilon_3 &\geq 4 \\
\epsilon_3 &\geq 1 \\
-\epsilon_2 + \epsilon_3 &\geq 1 \\
-\epsilon_2 + \epsilon_3 &\geq 2 \\
\epsilon_2 + \epsilon_3 &\geq 0
\end{aligned}$$

Thus, in order to prove the claim, author just have to exhibit a pair $(\epsilon_2, \epsilon_3)$ satisfying these inequalities and having $\omega = \epsilon_2 + (n-1)\epsilon_3 = \frac{5n-7}{3}$. And such a pair is $\epsilon_2 = -\frac{2}{3}$, $\epsilon_3 = \frac{5}{3}$.

The bound $f(n) \leq \frac{5n+5}{3}$ now follows directly since it takes four more moves to transform a permutation with $n-1$ adjacencies to the identity permutation. In any event, the constant term of the bound can be improved quite easily by stopping the algorithm when $n-k$, for some $k$, adjacencies have been formed, and then optimally putting together the $k+1$ pieces.

□

3.2. **Burnt pancakes.** In the case of burnt pancakes, there has been particular research on constraining the diameter. In 1995, initial bounds on the diameter, along with a conjecture, were introduced : a lower bound of $3n/2$ and an upper bound of $2(n-1)$. Subsequent bounds on the diameter were established in 1997. Additionally, a polynomial-time algorithm designed for addressing ”simple” burnt pancake problems was unveiled in 2011.

During the same year, Josef Cibulka illustrated that a minimum of $7n/4$ flips was imperative for effectively sorting stacks of burnt pancakes, considering stacks of size $n$ on average. Furthermore, he refuted the conjecture proposed by Cohen and Blum. In his work, Josef Cibulka introduced compelling concepts, including anti-adjacency and clan.

3.3. **An (18/11)n upper bound for sorting by prefix reversals** [10]**.** The Pancake Problem, which involves sorting permutations using prefix reversals, has practical applications in parallel processing, particularly in the pancake network. There is a notable interest in improving the upper bound for this problem. Gates and Papadimitriou previously demonstrated that permutations of length n can be sorted with at most (5n + 5)/3 prefix reversal operations. The authors propose an enhanced upper bound of (18/11)n + O(1), surpassing the current lower bound of (15/14)n.

Permutations are represented as lists, and consecutive elements are defined based on their numerical differences being ±1 (mod n). The concepts of blocks and singletons in a permutation are introduced, with blocks being maximal length sublists with adjacencies between consecutive elements, and singletons being elements not in a block. Exactly like Bill Gates described it.

The authors denote the number of blocks and singletons in a permutation as $b(\pi)$ and $s(\pi)$, respectively. A prefix reversal of size j transforms a permutation, and a potential function $\phi(\pi) = (18/11)s(\pi) + (24/11)b(\pi)$ is defined, serving as an upper bound for the number of prefix reversal operations needed to transform a permutation into a single block.

Cases (1) to (3) of the Gates and Papadimitriou algorithm are explained, focusing on permutations with an initial singleton element, where the initial object B is a singleton, and consecutive elements A and C are determined based on specific criteria.

Now, let's consider a permutation where the initial element is a block, denoted by BC. This leads to the remaining six cases in the Gates and Papadimitriou algorithm, distinguished by the type of elements (singletons or endpoints of blocks) that are consecutive with the endpoints B and C of the initial block.

Case (4) applies when the element consecutive with the beginning endpoint B is a singleton (denoted by A). Case (5) applies when the consecutive element is the beginning endpoint of another block (denoted by A∼). For the situation where the consecutive element with B is the terminating endpoint of another block (denoted by ∼A), Cases (6) and (7) are introduced. These are resolved by considering the element D consecutive with the terminating endpoint C of the initial block. Case (6) applies when D is the beginning endpoint of another block (denoted by D∼), and Case (7) applies when D is the terminating endpoint of another block (denoted by ∼D). The only remaining possibility is that D is a singleton, leading to Case (8) when ∼A occurs to the left of D and Case (9) when ∼A occurs to the right of D. All of these cases can be resolved in an acceptable number of steps.

They found it worth noting that Cases (6)–(9) deal with situations where the initial object is a block B∼C, and B's consecutive element is the terminating endpoint of another block (∼A). This single configuration was expanded into four cases (6)–(9) to provide more information about the permutation's structure.

To improve the upper bound, a similar strategy is employed. The authors design an enhanced algorithm by delving into cases in more detail, resulting in a further division of the nine cases. Starting with the original nine cases that form a complete solution for the previous upper bound, a refined set of cases is constructed through compounding.

In this approach, if Case (i), where 1 i 9, applies to a permutation , the prefix reversals defined by the Gates and Papadimitriou algorithm for Case (i) yield a new permutation $\pi$' to which Case (j) applies. This two-case sequence is termed a compound case, denoted as compound Case (i)–(j), and it applies to $\pi$. As every permutation belongs to such a compound case, this perspective provides a complete set of cases, except when $\pi$ becomes a single block by the steps of the initial Case (i). By considering these compound cases, more efficient prefix reversal sequences can be designed.

The authors have found shorter flip sequences for most compound cases, although some exceptions exist. To address cases where shorter sequences are not found, additional expansion is carried out. This involves considering additional elements adjacent to previously identified elements and exploring all possible positions of objects consecutive with the new elements, a method referred to as expansion by breadth.

For instance, one challenging sub-case of the compound case (3-3) is any permutation of the form B H∼ J∼ ∼A I ∼C, denoted as (3-3a). This sub-case involves an initial Case (3), where the initial singleton is denoted by B, and B's two consecutive elements are the terminal endpoints of blocks (∼A and ∼C). The subsequent Case (3) involves a singleton I next to ∼A, and somewhere between the singleton B and the block ∼A are the blocks H∼ and J∼ containing the elements H and J, consecutive with I.

To efficiently sequence flips for this case, an expansion of the permutation into several sub-cases is considered, represented by the generating string B H∼ J∼ ∼A I S∼CT*. The generating string denotes all sub-cases, where the beginning endpoint of the block ∼C is denoted by the symbol S and the element consecutive to S is denoted by the symbol T, considering all possible positions of T, whether as a singleton or endpoint of a block. The compact notation T* indicates all sub-cases based on possible positions of the element T. This expansion ensures coverage for every permutation of the compound (3-3a) case.

While the flip sequences are not explicitly presented, table 15 in the Appendix indicates the existence of an efficient flip sequence for all but one of these sub-cases. The exception is the sub-case denoted by BH∼ J∼ ∼A I S∼C ∼T, where the element T is the terminating endpoint of a block appearing after the block S∼C. To find a good flip sequence for this exceptional sub-case, a further expansion is conducted into B H∼ J∼ U∼A I S∼C ∼T V*.

Among the 93 compound cases where good sequences must be found 11 fail. That is, for 11 of the 93 sub-cases, no flip sequence exists that is sufficient to satisfy our potential function. These are indicated by the word deficiency in the tables. For each of these eleven failures, they investigate sub-cases based on expansion by breadth.

Expansion by breadth increases the number of total cases dramatically because it introduces new elements, and we must consider all possible positions for their consecutive elements. The number of possible arrangements increases significantly. Consequently, the description in the paper does not explicitly list all of the sub-cases. Instead, they use generating strings to describe expansions that are sufficient to obtain a complete set of cases. Generating strings can be used to describe a complete list of all 2220 cases more succinctly. they have verified the correctness of the potential function $\phi(\pi) = (18/11)s(\pi) + (24/11)b(\pi)$ in all cases. Thus, all failed sub-cases have been resolved.

## 3.4. NP-hardness. [9]

In 2011, Laurent Bulteau, Guillaume Fertin, and Irena Rusu proved that the problem of finding the shortest sequence of flips for a given stack of pancakes is NP-hard. This work isn't the most important for our research, but it gives us information about the problem's computational complexity.

**Definition 3.1.** ***P class*** *- the complexity class of decision problems which can be solved in polynomial-time*

***NP class*** *- the complexity class of decision problems for which we aren't sure to have a polynomial algorithm but for every instance, where the answer is "yes", we have proofs that the instance can be solved in a polynomial-time.*

***NP-hard*** *- the problem K is NP-hard when for every problem M in NP, there is a polynomial-time many-one reduction (reduction that converts an instance of one problem to another) from M to K.*

To prove that the pancake flipping problem (also known as Sorting By Prefix Reversals) is NP-hard, authors will reduce 3-SAT [13] instance to Pancake instance. 3-SAT is a boolean satisfiability problem that determines the satisfiability of a formula in conjunctive normal form where each clause is limited to at most three literals. This problem is NP-complete which means that all the problems of a complexity class NP, are at most as difficult to solve as SAT.

For the author of this paper efficient flips are the flips that decrease the number of breakpoints, where x considers a breakpoint if x isn't adjacent to the next element. For the last element of the sequence, we can call it a breakpoint if it isn't the maximum.

In the first part, the author defines some structures which are called *gadgets* to simulate boolean variables and clauses with subsequences. They are organized in two levels. For each defined gadget, we derive a property characterizing the efficient paths that can be followed if some part of the gadget appears at the head of a sequence.

In the first level each gadget is like a function that takes one parameter and gives you the sequence of integers. There are 4 gadgets at this level : Dock, Lock, Fork, and Hook.

Then the author defines 3 more gadgets using the patterns of level-1 gadgets.There are level-2 gadgets :

- Literals - used only once in the reduction. It contains the locks corresponding to all literals of the formula.
- Variable - a gadget simulating a Boolean variable $x_i$.
- Clause - gadget simulates a 3-clause in a Boolean formula. It holds the test elements for three locks, corresponding to three literals.

In the second part, the authors start the reduction. Let $\phi$ - a Boolean formula over l variables in conjunctive normal form, such that each clause contains exactly three literals. It's a 3-SAT instance. $S_\phi$ - a permutation of $[|\ 1..l\ |]$ based on gadgets which were defined earlier. It's a Sorting By Prefix Reversals instance. Then they describe three steps of reduction

**Step 1. Variable assignment** In this step, the authors assign all variables of $S_\phi$ to either the P class or the N class. The ones with index in P(resp. N) open locks to Literals of form $x_i$(resp $\neg x_i$)

Thus, they introduce lemme 1, which ensures that any sequence of efficient flips begins with a full assignment of the variables, and every possible assignment can be reached using only efficient flips (Flips that don‘t increase the number of breakpoints in the sequence)

**Step 2. Going through clauses** After the first step, we need to check if our assignment satisfies the formula $\phi$. For each clause, we select one true literal and we try to open the correspondent lock.

Here the authors introduce lemme 2, which ensures that after the assignment, every efficient path starting from $S_\phi$ needs to select a literal in each clause (set of these literals we note $\alpha$), under the constraint that the selection is compatible with the assignment.

**Step 3. Beyond clauses** In this step, the author poses lemme 3, which ensures that from the $S_\phi$ after assignment and going through clauses, we can reach the sequence 1,2,3..n (identity) using efficient flips only.

Now we move to the final theorem

**Theorem 3.2.** *$S_\phi$ has an efficient pass to the sorted sequence if and only if $\phi$ is satisfiable.*

Proving this theorem will show us the possibility of reducing the 3-SAT instance to a Pancake instance.

3.4.1. *proof NP-hard.*

*Proof.* $\Rightarrow$ Firstly, the authors assume that there is efficient pass from $S_\phi$ to a sorted sequence. lemme 1 proves that there are two classes P and N using which we can do a full assignment. So there is surely some path from $S_\phi$ to identity uses assigned $S_\phi$. And by the lemme 2, we selected a set $\alpha$ and there is some path from assigned $S_\phi$ to identity such that uses assigned $S_\phi$ with selected $\alpha$. So in every clause of $\phi$ we have at least one literal that is true. It's the exact definition of $\phi$ is satisfiable.

$\Leftarrow$ Secondly, the authors assume that $\phi$ is satisfiable. Consider any truth assignment making $\phi$ true, such $x_i \in$ P the set of indices. So, if our formula contains l variables set of indices N is $[| \ 1..l \ |] \setminus$ P. And we note $\alpha$ the selection one literal being true under this assignment for each clause of $\phi$.

So by lemme 1, there is an efficient path from $S_\phi$ to assigned $S_\phi$.

Then by lemme 2, there is an efficient path from assigned $S_\phi$ to assigned $S_\phi$ with selected $\alpha$

And finally, by lemme 3 there is a path from $S_\phi$ to assigned $S_\phi$ with selected $\alpha$ to identity. Thus sequence $S_\phi$ is efficiently sortable.

□

So by reducing from the 3-SAT instance to Sorting By Prefix Reversals instance, the authors conclude that the pancake problem is *NP-hard*.

## 4. Our work

We had different approaches during our work. So in this part, we would like to present some of them along with our conjectures.

### 4.1. **Approach 1 : n and n - 1.**

Here we use an algorithm of a complexity of 2n to sort the stack of pancakes. The table below describes the stack of pancakes with its number of flips.

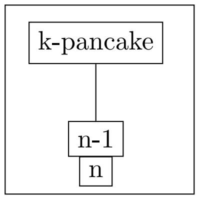
| n-pancake | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| flip | 0 | 0 | 1 | 3 | 4 | 5 | 7 | 8 | 9 | 10 |

[11]

Description of the number of flips

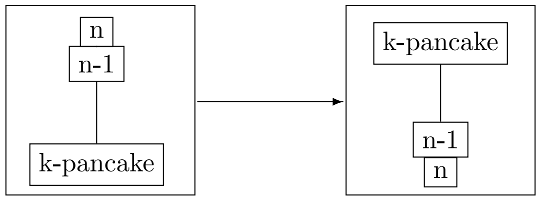
Where n is the number of pancakes and f is the number of flips.
Each stack has the following structure : (k-pancake + n + n-1) where n is the largest pancake, n-1 is the second largest and k-pancake is a stack of pancakes that has the same structure. We have listed a total of five different cases depending on f.

**Case f = 0 :**



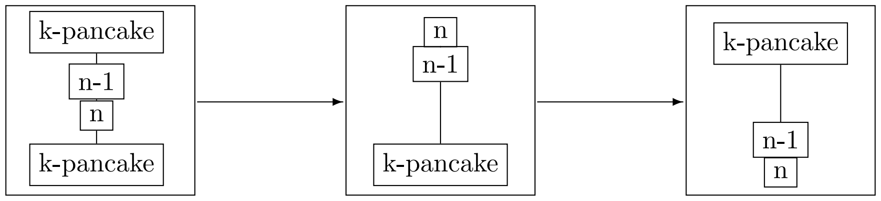
The two largest pancake are the bottom of the stack. Then f = 0.

**Case f = 1 :**



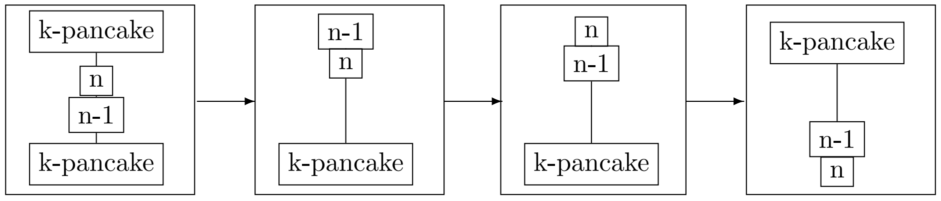
On the top of the stack are the two largest pancakes. To place the largest pancake at the bottom of the stack, we return it. Consequently, f = 1.

**Case f = 2 :**



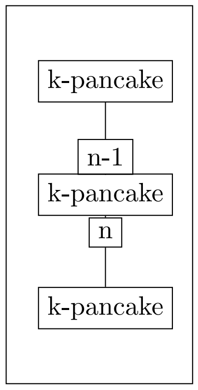
The two largest pancakes are in the middle of the stack where n-1 is just above n. We flip the pancakes one time starting from the largest (n). Now we have k-pancakes above k-pancakes that we merge to have one. Finally, we reverse the pancake stack to have the largest at the bottom. leading to f = 2.

**Case f = 3 :**

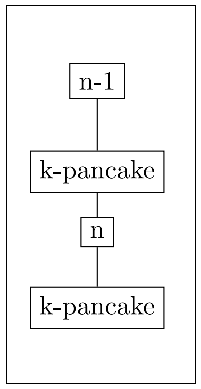


For this case, the two largest pancakes are in the middle of the stack, but now n is above n-1. So we first flip the stack starting from n-1, then we flip again reversing n-1 and n that are now on the top. Finally, we reverse the whole stack. In the end, it took us 3 flips to sort this stack. Thus, f = 3.

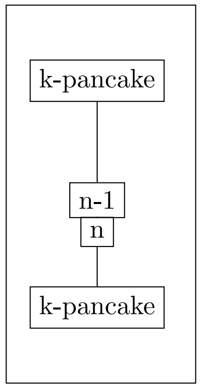
**Case f = 4 :**
The two largest pancakes are in the middle of the stack where n-1 is below n, but this time they are separated. The example below shows the different steps making the total number of flips equal to four (f = 4).
For example [8] :



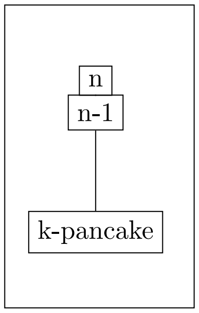
In the initial step, we have a stack of k-pancakes where n is the first largest pancake and n-1 is the second largest pancake. They are placed in the middle of the stack. The counter of flips is zero.



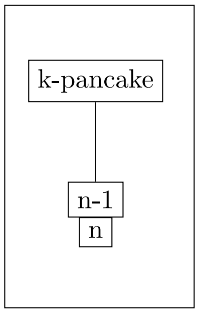
In the first step, we have to place the pancake n-1 on the top The counter of flips is one.



In the second step, we have to place the pancake n-1 just above the pancake n. The counter of flips levels up by one. Therefore, the counter is now two.



In the third step, we have to flip the pancakes n and n-1 to the top. We can see the first pancake is n and the second is n-1. The counter of flip level up by one. Then, the counter is three.



In step four, we overturn to have the two largest in the bottom of the pancake stack. The counter of flip level up by one. Then, the counter is four.

Throughout this section, we have described a total of 5 cases depending on the number of flips we make in a stack where n and n-1 are the two largest pancakes. We have found a result of 4 maximum flips with a complexity of 2n.

## 4.2. **Approach 2 : permutation of 0 and 1.**

In this approach, we suppose that our pancakes have just two possible sizes. We can denote it by 0 and 1. And we will try to make a conjecture for this type of permutation.

**Permutations with the size n = 1**

We only have two possible situations : 0 and 1. Since they are already sorted, we need 0 flips.

**Permutations with the size n = 2**

Now it is 4 possible permutations : 00,01,11,10. We notice that the permutation 00 can be reduced to $\sim 0$ with a size n = 1. Same thing for the permutation 11 $\sim$ 1. So like for all sequences with n = 1, we need 0 flips

The permutation 01 is already sorted so the number of flips is 0. But 10 is not sorted and we note that 1 flip is enough to get from 10 to 01.

So the maximum number of flips needed for permutations with n = 2 is 1.

**Permutations with the size n = 3**

Before we make some conclusion we would like to analyze the permutations of size 3. We have the 8 following possibilities : 000,111,001,011,110,100,010,101,

For the first 4 cases, we see that they are already sorted. But also we can see that the first 6 can be reduced to sequences of sizes 1 and 2, because when we have two "0"s or two "1"s together it‘s obvious that we need to flip them together two, thus we can merge them because they form a sub-sequence which is already sorted so we can reduce 00 0 and 11 1.

Now we can introduce the first useful definition :

**Definition 4.1.** *Meaningful permutation - A permutation that cannot be reduced to a smaller size permutation by merging two consecutive identical numbers as one.*

For the sequences with a size of 3, we have 2 meaningful permutations. And the number minimum of flips is 2 for 010 ($010 \to 100 \sim 10 \to 01$) and 1 for 101 ($101 \to 011 \sim 01$). So the maximum number of flips for n = 3 is 2.

**Meaningful permutation**

We can see directly that this type of permutation has some proprieties :

- The permutation of size n which consists of n distinct numbers is always meaningful (it's true because all elements are different).
- For permutations of 0 and 1 of the size n there are only two meaningful permutations.

We can prove the second point by recursion

*Proof.* **Induction step**. We have already seen that for the permutation of size 1, we have two possible situations : 0 and 1. Because of the first property, they are meaningful.

**Recursion step**. We assume that the permutations of 0 and 1 of the size n-1 have two meaningful permutations. (IH)

How can we build a permutation of the size n ? By adding 0 or 1 to any permutation of size n-1. We notice that if we add 0 or 1 to NOT meaningful permutation, it will remain not meaningful because there are already consecutive identical numbers in this permutation, so we can reduce it. But if we add 0 or 1 to a meaningful permutation, there is a possibility to obtain a new meaningful permutation of size n.

By our assumption (IH) there are two meaningful permutations of size n-1. If we add 0 to the meaningful permutation that ends by 0, we could reduce it. So we add 0 to the sequence that ends by 1 and 1 to the sequence that ends by 0. And from each meaningful permutation of size n-1, we can construct one meaningful permutation of size n. By IH there are two sequences for n - 1 $\Rightarrow$ there are two sequences for n.

So we demonstrated by recursion that there are exactly two meaningful permutations for all sequences of 0 and 1 of size n. □

Actually, we can also understand it by looking at some meaningful permutations of 0 and 1. It‘s always under the form : 10101010... or 01010101... .

**Conjecture for pancakes of two sizes**

Now we will try to prove the following Theorem by recursion :

**Theorem 4.1.** *For a stack of n pancakes of just two sizes, we need a maximum of n - 1 flips to get it sorted.*

*Proof.* **Induction step**. We have already seen that for n = 2, we need exactly 1 flip.

**Reduction step**. We assume that for a stack of n - 1 pancakes of two sizes we have a maximum of n-2 flips to sort any permutation. (IH)

For sequences of size n that are not meaningful, we can reduce them to the sequences of size n-1 so the maximum number of flips by IH will be n-2 for this type of permutation.

But for meaningful permutations, we need another approach. If this type of sequence begins with 0 then after we have 1 and then 0 again. So in one flip(by exchanging first 0 and following 1) we can create a non-meaningful permutation that can be reduced to size n-1 and by IH be sorted in n-2 flips. It means that for initial meaningful permutation, we need n-2+1 = n - 1 flips. The same strategy can be used if our meaningful permutation begins with 1. Using the property that there are only two meaningful permutations for the sequence of 0 and 1 of size n, we conclude that we have seen all the possible permutations of size n.

By recursion, our theorem has been proved. □

**Python code for sorting permutation of 0 and 1**

```
# Function that checks if our permutation is already sorted.
def isSorted(s):
    is0 = True
    for c in s:
        if c == '0' and not(is0):
            return False
        if c == '1' and is0:
            is0 = False
    return True

# Function that reverses the stack at the point.
def reversalByPrefix(s, point):
    return s[:point][::-1]+s[point:]

# Function that sorts permutation of 0 and 1.
def pacake01_sort(s):
    n = len(s)
    sorted_s = s[::]
    numberFlips = 0

    while not(isSorted(sorted_s)):
        j = 1
        # This while do "reduction" by assuming
        # the first consecutive identical numbers like the one element
    .
        while j < n and sorted_s[j] == sorted_s[0]:
            j += 1
        # We search for the occurrence of
        # the same element that the first one in our permutation.
        for i in range(j, n):
            # When we found it we do one flip.
            if sorted_s[i] == sorted_s[0]:
                sorted_s = reversalByPrefix(sorted_s, i)
                numberFlips += 1
                break
            # If we in the end and we did not found it,
            # it means that our permutation is sorted but in the
    reverse order.
            # So we perform one flip to get the right order.
            if i == n-1 and not(isSorted(sorted_s)):
                sorted_s = sorted_s[::-1]
                numberFlips += 1

    return sorted_s, numberFlips
```

In addition, we have prepared a test for the sequences from size 1 to 100. The code can be found below and a table containing the results can be found in the appendix.

```
meaningfulSeqOne = "0"
meaningfulSeqTwo = "1"

for i in range(1,101):
    print("Number of elements is : ", i)
    print("\t Sequence: ", meaningfulSeqOne)
    print("\t Number of flips: ",pacake01_sort(meaningfulSeqOne)[1])
    print("\t Sequence: ", meaningfulSeqTwo)
    print("\t Number of flips: ", pacake01_sort(meaningfulSeqTwo)[1],"
    \n")

    meaningfulSeqOne += "1"
    meaningfulSeqTwo += "0"
    meaningfulSeqOne, meaningfulSeqTwo = meaningfulSeqTwo,
    meaningfulSeqOne
```

## 5. Conclusion and future Work

Our research revolves around the intricate challenges posed by the pancake problem, a captivating area within permutation sorting by reversals. While the unsigned permutation domain presents formidable difficulties, the signed permutation domain benefits from an efficient polynomial-time solver. However, the pancake problem, both in its unburnt and burnt variants, introduces unique complexities that have yet to be fully unraveled.

Our contribution is experimental using n and n-1 approaches sought to elucidate our insights by providing examples using pancake stacks of minimal sizes, such as 0 and 1. These examples served as a foundation for a conjecture aimed at generalizing solutions for unburnt pancake stacks. Our experimentation extended to larger stacks, with a notable attempt involving a stack of 20 pancakes. However, challenges emerged as we ventured into stacks of greater magnitude.

To augment the clarity of our findings, we have implemented a Python code to articulate the problem and the corresponding tests. This coding approach not only serves as a practical demonstration but also provides a tangible means for fellow researchers to engage with and understand our work.

Despite the challenges encountered, particularly in dealing with larger pancake stacks, our research has shed light on the inherent complexities of the problem. These challenges serve as stepping stones for future investigations, prompting us to refine our methodologies and algorithms.

Looking ahead, our research trajectory involves an in-depth exploration of the burnt pancake problem. This problem, in a fascinating twist, is intimately connected to the unburnt pancake problem.

## 6. Appendix

Au cours de ce projet, les élèves suivants ont traduit les articles wikipedia suivants :
Chanez ARROUM : Tri par fusion-insertion
Aleksandra KOVIAZINA et Melissa BENKHODJA : Tri par comparaison
Justine XU : Graphe des pancakes
Lina CHENNI est dispensée de EVC

### 6.1. **Result of the test for sequences of 0 and 1.** :

| Size | First sequence | Number of flips | Second sequence | Number of flips |
|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 |
| 2 | 01 | 0 | 10 | 1 |
| 3 | 010 | 2 | 101 | 1 |
| 4 | 0101 | 2 | 1010 | 3 |
| 5 | 01010 | 4 | 10101 | 3 |
| 6 | 010101 | 4 | 101010 | 5 |
| 7 | 0101010 | 6 | 1010101 | 5 |
| 8 | 01010101 | 6 | 10101010 | 7 |
| 9 | 010101010 | 8 | 101010101 | 7 |
| 10 | 0101010101 | 8 | 1010101010 | 9 |
| 11 | 01010101010 | 10 | 10101010101 | 9 |
| 12 | 010101010101 | 10 | 101010101010 | 11 |
| 13 | 0101010101010 | 12 | 1010101010101 | 11 |
| 14 | 01010101010101 | 12 | 10101010101010 | 13 |
| 15 | 010101010101010 | 14 | 101010101010101 | 13 |
| 85 | 010101010101010... | 84 | 101010101010101... | 83 |
| 86 | 010101010101010... | 84 | 101010101010101... | 85 |
| 87 | 010101010101010... | 86 | 101010101010101... | 85 |
| 88 | 010101010101010... | 86 | 101010101010101... | 87 |
| 89 | 010101010101010... | 88 | 101010101010101... | 87 |
| 90 | 010101010101010... | 88 | 101010101010101... | 89 |
| 91 | 010101010101010... | 90 | 101010101010101... | 89 |
| 92 | 010101010101010... | 90 | 101010101010101... | 91 |
| 93 | 010101010101010... | 92 | 101010101010101... | 91 |
| 94 | 010101010101010... | 92 | 101010101010101... | 93 |
| 95 | 010101010101010... | 94 | 101010101010101... | 93 |
| 96 | 010101010101010... | 94 | 101010101010101... | 95 |
| 97 | 010101010101010... | 96 | 101010101010101... | 95 |
| 98 | 010101010101010... | 96 | 101010101010101... | 97 |
| 99 | 010101010101010... | 98 | 101010101010101... | 97 |
| 100 | 010101010101010... | 98 | 101010101010101... | 99 |

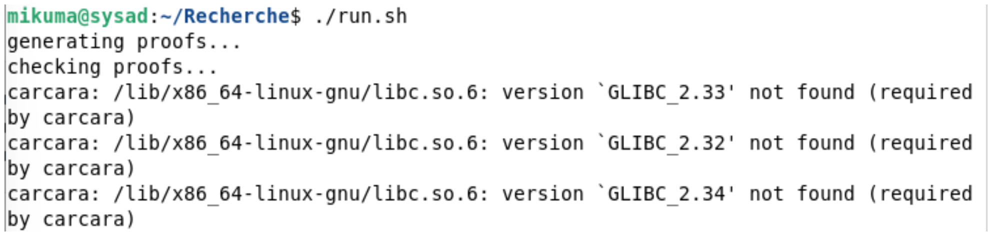
Table 1. Result of test for the sequences of 0 and 1

### 6.2. **Reproducibility of dataset.**

The goal of this part of the paper is to try to reproduce the results of another article. We have been given a research paper titled “Carcara : an Efficient Proof Checker and Elaborator for Alethe Proofs”[7] which presents Carcara, an independent proof checker and elaborator for Alethe, implemented in Rust. It aims to increase the adoption of the format by providing push-button proof-checking for Alethe proofs, focusing on efficiency and usability.

We tried to reproduce the *table 1* and the figures 5a, 6a, 6b, and 7 of this work. In this table, the authors present total solving and proof-checking time per logic for veriT and Carcara. When running on the full benchmark set, on a machine with 64 cores and 512 GB of memory, the total runtime of the experiment is approximately 10 hours. Since we do not have such a powerful machine, we have reproduced the experiment on a smaller version of the data set in a machine with 4 cores and 16 GB of memory, the total runtime is approximately 10 minutes. We ran it on the provided benchmark sample only, because the total runtime is smaller.

First, we tried to work with a virtual machine that was already pre-installed on our machine. We made sure to install a Rust compiler and toolchain which were necessary for our work. But we encountered the following problem :



```
mikuma@sysad:~/Recherche$ ./run.sh
generating proofs...
checking proofs...
carcara: /lib/x86_64-linux-gnu/libc.so.6: version `GLIBC_2.33' not found (required
by carcara)
carcara: /lib/x86_64-linux-gnu/libc.so.6: version `GLIBC_2.32' not found (required
by carcara)
carcara: /lib/x86_64-linux-gnu/libc.so.6: version `GLIBC_2.34' not found (required
by carcara)
```

To get rid of it, we took 2 hours to understand the article, download the official virtual machine TACAS 2023, which is 5.8 GB, and the Carcara artifact[6], which is 367.3 MB. We read the file "README.md" to have the instructions for the data reproducibility. The instructions were clear, we hadn't run across any issues, and we did not need to install any additional bibliographies. We used Linux to execute the "run.sh" program that generated 7 photos and one file to reflect table 1 and different figures in the article. This experiment was tested ten times, although each test took 20 minutes. Nine tests turned out successful and one failed. The acquired results (Figure 2) differ from those in the publication (Figure 1), as shown below.

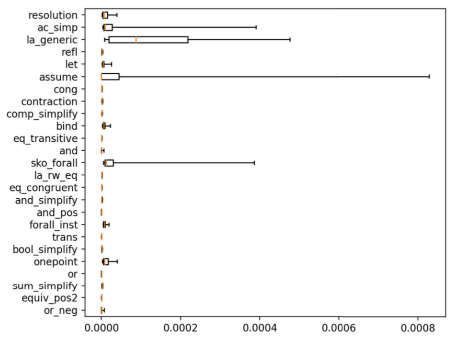


Figure 1. Paper data

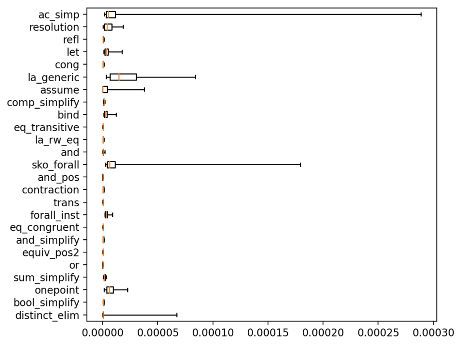
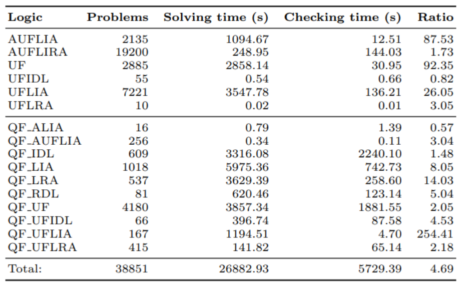


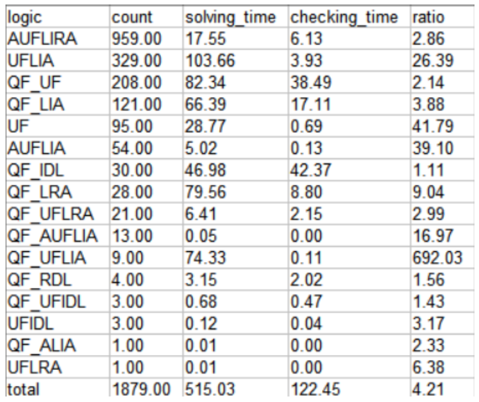
Figure 2. Data obtained

In the obtained results, we notice that the execution time is different because the number of problems is not the same. When running the benchmark sample, we have 1879 problems in total (Figure 4), while in table 1 from the article, we have 38851 problems (Figure 3).



| Logic | Problems | Solving time (s) | Checking time (s) | Ratio |
|---|---|---|---|---|
| AUFLIA | 2135 | 1094.67 | 12.51 | 87.53 |
| AUFLIRA | 19200 | 248.95 | 144.03 | 1.73 |
| UF | 2885 | 2858.14 | 30.95 | 92.35 |
| UFIDL | 55 | 0.54 | 0.66 | 0.82 |
| UFLIA | 7221 | 3547.78 | 136.21 | 26.05 |
| UFLRA | 10 | 0.02 | 0.01 | 3.05 |
| QF_ALIA | 16 | 0.79 | 1.39 | 0.57 |
| QF_AUFLIA | 256 | 0.34 | 0.11 | 3.04 |
| QF_IDL | 609 | 3316.08 | 2240.10 | 1.48 |
| QF_LIA | 1018 | 5975.36 | 742.73 | 8.05 |
| QF_LRA | 537 | 3629.39 | 258.60 | 14.03 |
| QF_RDL | 81 | 620.46 | 123.14 | 5.04 |
| QF_UF | 4180 | 3857.34 | 1881.55 | 2.05 |
| QF_UFIDL | 66 | 396.74 | 87.58 | 4.53 |
| QF_UFLIA | 167 | 1194.51 | 4.70 | 254.41 |
| QF_UFLRA | 415 | 141.82 | 65.14 | 2.18 |
| Total: | 38851 | 26882.93 | 5729.39 | 4.69 |

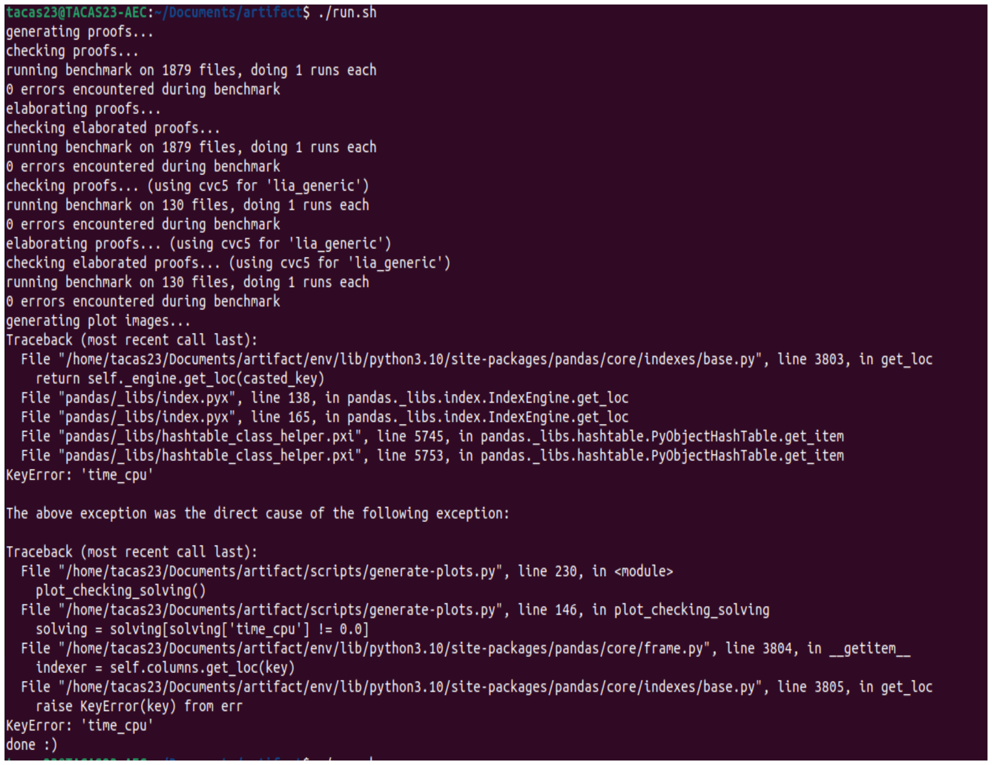
FIGURE 3. Paper data



| logic | count | solving_time | checking_time | ratio |
|---|---|---|---|---|
| AUFLIRA | 959.00 | 17.55 | 6.13 | 2.86 |
| UFLIA | 329.00 | 103.66 | 3.93 | 26.39 |
| QF_UF | 208.00 | 82.34 | 38.49 | 2.14 |
| QF_LIA | 121.00 | 66.39 | 17.11 | 3.88 |
| UF | 95.00 | 28.77 | 0.69 | 41.79 |
| AUFLIA | 54.00 | 5.02 | 0.13 | 39.10 |
| QF_IDL | 30.00 | 46.98 | 42.37 | 1.11 |
| QF_LRA | 28.00 | 79.56 | 8.80 | 9.04 |
| QF_UFLRA | 21.00 | 6.41 | 2.15 | 2.99 |
| QF_AUFLIA | 13.00 | 0.05 | 0.00 | 16.97 |
| QF_UFLIA | 9.00 | 74.33 | 0.11 | 692.03 |
| QF_RDL | 4.00 | 3.15 | 2.02 | 1.56 |
| QF_UFIDL | 3.00 | 0.68 | 0.47 | 1.43 |
| UFIDL | 3.00 | 0.12 | 0.04 | 3.17 |
| QF_ALIA | 1.00 | 0.01 | 0.00 | 2.33 |
| UFLRA | 1.00 | 0.01 | 0.00 | 6.38 |
| total | 1879.00 | 515.03 | 122.45 | 4.21 |

FIGURE 4. Data obtained

In the failed result, we have 4 images, but the processor seems unable to execute the Time file. Furthermore, we do not have any file that contains the time according to the number of problems.



```
tacas23@TACAS23-AEC:~/Documents/artifact$ ./run.sh
generating proofs...
checking proofs...
running benchmark on 1879 files, doing 1 runs each
0 errors encountered during benchmark
elaborating proofs...
checking elaborated proofs...
running benchmark on 1879 files, doing 1 runs each
0 errors encountered during benchmark
checking proofs... (using cvc5 for 'lia_generic')
running benchmark on 130 files, doing 1 runs each
0 errors encountered during benchmark
elaborating proofs... (using cvc5 for 'lia_generic')
checking elaborated proofs... (using cvc5 for 'lia_generic')
running benchmark on 130 files, doing 1 runs each
0 errors encountered during benchmark
generating plot images...
Traceback (most recent call last):
  File "/home/tacas23/Documents/artifact/env/lib/python3.10/site-packages/pandas/core/indexes/base.py", line 3803, in get_loc
    return self._engine.get_loc(casted_key)
  File "pandas/_libs/index.pyx", line 138, in pandas._libs.index.IndexEngine.get_loc
  File "pandas/_libs/index.pyx", line 165, in pandas._libs.index.IndexEngine.get_loc
  File "pandas/_libs/hashtable_class_helper.pxi", line 5745, in pandas._libs.hashtable.PyObjectHashTable.get_item
  File "pandas/_libs/hashtable_class_helper.pxi", line 5753, in pandas._libs.hashtable.PyObjectHashTable.get_item
KeyError: 'time_cpu'

The above exception was the direct cause of the following exception:

Traceback (most recent call last):
  File "/home/tacas23/Documents/artifact/scripts/generate-plots.py", line 230, in <module>
    plot_checking_solving()
  File "/home/tacas23/Documents/artifact/scripts/generate-plots.py", line 146, in plot_checking_solving
    solving = solving[solving['time_cpu'] != 0.0]
  File "/home/tacas23/Documents/artifact/env/lib/python3.10/site-packages/pandas/core/frame.py", line 3804, in __getitem__
    indexer = self.columns.get_loc(key)
  File "/home/tacas23/Documents/artifact/env/lib/python3.10/site-packages/pandas/core/indexes/base.py", line 3805, in get_loc
    raise KeyError(key) from err
KeyError: 'time_cpu'
done :)
```

## References

1. *Christos Papadimitriou — Wikipédia — fr.wikipedia.org*, https://fr.wikipedia.org/wiki/Christos_Papadimitriou, [Accessed 06-12-2023].
2. *Duality in linear programming*, https://web.mit.edu/15.053/www/AMP-Chapter-04.pdf.
3. *Jacob E. Goodman — Wikipédia — fr.wikipedia.org*, https://fr.wikipedia.org/wiki/Jacob_E._Goodman, [Accessed 06-12-2023].
4. *William T. Trotter - Wikipedia — en.wikipedia.org*, https://en.wikipedia.org/wiki/William_T._Trotter, [Accessed 06-12-2023].
5. *The american mathematical monthly*, https://maa.org, Dec 2023.
6. Bruno Andreotti, Hanna Lachnitt, and Haniel Barbosa, *Carcara artifact*, https://zenodo.org/records/7574451.
7. ———, *Carcara: An efficient proof checker and elaborator for smt proofs in the alethe format*, Springer Cham **13993** (2023), no. 1, 367–386.
8. Saúl A. Blanco, Charles Buehrle, and Akshay. Patidar, *On the number of pancake stacks requiring four flips to be sorted*, Discrete Math. Theor. Comput. Sci. **21** (2019), no. 2, 27 (English), Id/No 3.
9. Laurent Bulteau, Guillaume Fertin, and Irena Rusu, *Pancake flipping is hard*, Journal of Computer and System Sciences **81** (2015), no. 8, 1556–1574.
10. B. Chitturi, W. Fahle, Z. Meng, L. Morales, C.O. Shields, I.H. Sudborough, and W. Voit, *An (18/11)n upper bound for sorting by prefix reversals*, Theoretical Computer Science **410** (2009), no. 36, 3372–3390, Graphs, Games and Computation: Dedicated to Professor Burkhard Monien on the Occasion of his 65th Birthday.
11. William H. Gates and Christos H. Papadimitriou, *Bounds for sorting by prefix reversal*, Discrete Mathematics **27** (1979), no. 1, 47–57.
12. Wikipedia, *Bill Gates — Wikipedia, the free encyclopedia*, http://fr.wikipedia.org/w/index.php?title=Bill%20Gates&oldid=208093723, 2023, [Online; accessed 22/November/2023].
13. Wikipedia contributors, *Boolean satisfiability problem — Wikipedia, the free encyclopedia*, 2023, [Online; accessed 22-November-2023].