arXiv:2005.00168v1 [cs.DS] 1 May 2020

Cutting Bamboo Down to Size

Davide Bilò
Department of Humanities and Social Sciences, University of Sassari, Italy.
davide.bilo@uniss.it

Luciano Gualà
Department of Enterprise Engineering, University of Rome "Tor Vergata", Italy.
guala@mat.uniroma2.it

Stefano Leucci
Department of Information Engineering, Computer Science and Mathematics, University of L'Aquila, Italy.
stefano.leucci@univaq.it

Guido Proietti
Department of Information Engineering, Computer Science and Mathematics, University of L'Aquila, Italy.
Institute for System Analysis and Computer Science "Antonio Ruberti" (IASI CNR), Italy.
guido.proietti@univaq.it

Giacomo Scornavacca
Department of Humanities and Social Sciences, University of Sassari, Italy.
giacomo.scornavacca@graduate.univaq.it

Abstract

This paper studies the problem of programming a robotic panda gardener to keep a bamboo garden from obstructing the view of the lake by your house.

The garden consists of n bamboo stalks with known daily growth rates and the gardener can cut at most one bamboo per day. As a computer scientist, you found out that this problem has already been formalized in [Gąsieniec et al., SOFSEM'17] as the *Bamboo Garden Trimming (BGT) problem*, where the goal is that of computing a perpetual schedule (i.e., the sequence of bamboos to cut) for the robotic gardener to follow in order to minimize the *makespan*, i.e., the maximum height ever reached by a bamboo.

Two natural strategies are `Reduce-Max` and `Reduce-Fastest`(x). `Reduce-Max` trims the tallest bamboo of the day, while `Reduce-Fastest`(x) trims the fastest growing bamboo among the ones that are taller than x. It is known that `Reduce-Max` and `Reduce-Fastest`(x) achieve a makespan of $O(\log n)$ and 4 for the best choice of $x = 2$, respectively. We prove the first constant upper bound of 9 for `Reduce-Max` and improve the one for `Reduce-Fastest`(x) to $\frac{3+\sqrt{5}}{2} < 2.62$ for $x = 1 + \frac{1}{\sqrt{5}}$.

Another critical aspect stems from the fact that your robotic gardener has a limited amount of processing power and memory. It is then important for the algorithm to be able to *quickly* determine the next bamboo to cut while requiring at most linear space. We formalize this aspect as the problem of designing a *Trimming Oracle* data structure, and we provide three efficient Trimming Oracles implementing different perpetual schedules, including those produced by `Reduce-Max` and `Reduce-Fastest(x)`.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases bamboo garden trimming; trimming oracles; approximation algorithms; pinwheel scheduling

Funding *Davide Bilò*: This work was partially supported by the Research Grant FBS2016_BILO, funded by "Fondazione di Sardegna" in 2016.
Giacomo Scornavacca: This work was partially supported by Research Grant FBS2016_BILO, funded by "Fondazione di Sardegna" in 2016.

1 Introduction

You just bought a house by a lake. A bamboo garden grows outside the house and obstructs the beautiful view of the lake. To solve the problem, you also bought a robotic panda gardener which, once per day, can instantaneously trim a single bamboo. You have already measured the growth rate of every bamboo in the garden, and you are now faced with programming the gardener with a suitable schedule of bamboos to trim in order to keep the view as clear as possible.

This problem is known as the *Bamboo Garden Trimming (BGT) Problem* [11] and can be formalized as follows: the garden contains n bamboos $b_1, \ldots, b_n$, where bamboo b_i has a known daily growth rate of $h_i > 0$, with $h_1 \geq \ldots \geq h_n$ and $\sum_{i=1}^{n} h_i = 1$. Initially, the height of each bamboo is 0, and at the end of each day, the robotic gardener can trim at most one bamboo to instantaneously reset its height to zero. The height of bamboo b_i at the end of day $d \geq 1$ and before the gardener decides which bamboo to trim is equal to $(d - d')h_i$, where $d' < d$ is the last day preceding d in which b_i was trimmed (if b_i was never trimmed before day d, then $d' = 0$). See Figure 1 for an example.

The main task in BGT is to design a perpetual trimming schedule that keeps the tallest bamboo ever seen in the garden as short as possible. In the literature of scheduling problems, this maximum height is called *makespan*.

A simple observation shows that the makespan must be at least 1 for every instance. Indeed, for any $\epsilon > 0$, a makespan of $1 - \epsilon$ would imply that the daily amount of bamboo cut from the garden is at most $1 - \epsilon$, while the overall daily growth rate of the garden is 1. This is a contradiction. Furthermore, there are instances for which the makespan can be made arbitrarily close to 2. Consider, for example, two bamboos b_1, b_2 with daily growth rates $h_1 = 1 - \epsilon$ and $h_2 = \epsilon$, respectively. Clearly, when bamboo b_2 must be cut, the height of b_1 becomes at least $2 - 2\epsilon$. This implies that the best makespan one can hope for is 2.

Two natural strategies are known for the BGT problem, namely `Reduce-Max` and `Reduce-Fastest`(x). The former consists of trimming the tallest bamboo at the end of every day, while the latter cuts the bamboo with fastest growth rate among those having a height of at least x. Experimental results show that `Reduce-Max` performs very well in practice as it seems to guarantee a makespan of 2 [2, 10]. However, the best known upper bound to the makespan is $1 + \mathcal{H}_{n-1} = \Theta(\log n)$, where $\mathcal{H}_{n-1}$ is the $(n-1)$-th harmonic number [5]. Interestingly, this $\Theta(\log n)$ bound also holds for the adversarial setting in which at every day an adversary decides how to distribute the unit of growth among all the bamboos. In this adversarial case such upper bound can be shown to be tight, while understanding whether `Reduce-Max` achieves a constant makespan in the non-adversarial setting is a major open problem [11, 10]. On the other hand, in [11] it is shown that `Reduce-Fastest`(x) guarantees a makespan of 4 for $x = 2$. Furthermore, it is also conjectured that `Reduce-Fastest`(1) guarantees a makespan of 2 [10].

In [11], the authors also provide a different algorithm guaranteeing a makespan of 2. This is obtained by transforming the BGT problem instance into an instance of a related scheduling problem called *Pinwheel Scheduling*, by suitably rounding the growth rates of the bamboos. Then, a perpetual schedule for the Pinwheel Scheduling instance is computed using existing algorithms [8, 13]. It turns out that this approach has a problematic aspect since it is known that *any* perpetual schedule for the Pinwheel Scheduling instance can have length $\Omega\left(\prod_{i=1}^{n} \frac{1}{h_i}\right)$ in the worst case.

The above observation gives rise to the following complexity issue: Can a perpetual schedule be efficiently implemented in general? Essentially, a solution consists of designing a

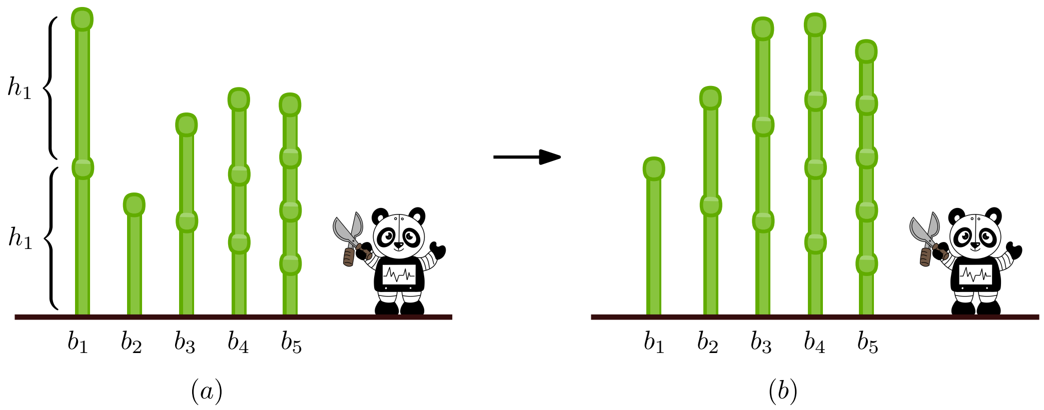

Figure 1 (a) The bamboo garden at the end of a day, just before the robotic gardener trims bamboo b_1. (b) The bamboo garden at the end of the next day, before cutting a bamboo.

trimming oracle, namely a *compact* data structure that is able to *quickly* answer to the query "What is the next bamboo to trim?".

It is worth noticing that similar problems are discussed in [11], where the authors ask for the design of trimming oracles that implement known BGT algorithms. For example, they explicitly leave open the problem of designing an oracle implementing `Reduce-Max` with query time of $o(n)$.

Our results. Our contribution is twofold. In Section 2, we provide the following improved analyses of `Reduce-Max` and `Reduce-Fastest`(x):

- We show that `Reduce-Max` achieves a makespan of at most 9. This is the first constant upper bound for this strategy and shows a separation between the static and the adversarial setting for which the makespan is known to be $\Theta(\log n)$.
- We show that, for any $x > 1$, `Reduce-Fastest`(x) guarantees a makespan of at most $\max\left\{x + \frac{x^2}{4(x-1)}, \frac{1}{2} + x + \frac{x^2}{4(x-\frac{1}{2})}\right\}$. For the best choice of $x = 1 + \frac{1}{\sqrt{5}}$, this results in a makespan of $1 + \phi = \frac{3+\sqrt{5}}{2} < 2.62$, where ϕ is the golden ratio. Notice also that for $x = 2$ (the best choice of x according to the analysis of [11]) we obtain an upper bound of 19/6 which improves over the previously known upper bound of 4.

Then, in Section 3, we provide the following trimming oracles:

- A trimming oracle implementing `Reduce-Max` whose query time is $O(\log^2 n)$ in the worst-case or $O(\log n)$ amortized. The size of the oracle is $O(n)$ while the time needed to build it is $O(n \log n)$. This answers the open problem given in [11].
- A trimming oracle implementing `Reduce-Fastest`(x) with $O(\log n)$ worst-case query time. This oracle has linear size and can be built in $O(n \log n)$ time.
- A trimming oracle guaranteeing a makespan of 2. This oracle uses the rounding strategy from [11] but it uses a different approach to compute a perpetual schedule. Our oracle answers queries in $O(\log n)$ amortized time, requires $O(n)$ space, and can be built in $O(n \log n)$ time.
  This result favorably compares with the existing oracles achieving makespan 2 implicitly obtained when the reduction of [11] is combined with the results in [13, 8] for the Pinwheel Scheduling problem. Indeed, once the instance G of BGT has been transformed into an instance P of Pinwheel Scheduling, any oracle implementing a feasible schedule for P is an oracle for G with makespan 2. In [13], the authors show how to compute a schedule

for P of length $L = \Omega(\prod_{i=1}^{n} \frac{1}{h_i})$, which results in an oracle with exponential building time and constant query time. In [8], an oracle having query time of $O(1)$ is claimed, but attaining such a complexity requires the use of $\Theta(n)$ parallel processors and the ability to perform arithmetic operations modulo L (whose binary representation may need $\Omega(n)$ bits) in constant time.

An interactive implementation of our Trimming Oracles described above is available at `https://www.isnphard.com/g/bamboo-garden-trimming/`.

Other related work. The BGT problem has been introduced in [11]. Besides the aforementioned results, this paper also provides an algorithm achieving a makespan better than 2 for a subclass of instances with *balanced* growth rates; informally, an instance is said to be balanced if at least a constant fraction of the overall daily growth is due to bamboos $b_2, \ldots, b_n$. The authors also introduce a generalization of the problem, named *Continuous BGT*, where each bamboo b_i grows continuously at a rate of h_i per unit of time and is located in a point of a metric space. The gardener can instantaneously cut a bamboo that lies in its same location, but needs to move from one bamboo to the next at a constant speed. Notice that this is a generalization of BGT problem since one can consider the trivial metric in which all distances are 1 (and it is never convenient for the gardener to remain in the same location).

Another generalization of the BGT problem called *cup game* can be equivalently formulated as follows: each day the gardener can reduce the height of a bamboo by up to $1 + \epsilon$ units, for some constant parameter $\epsilon \geq 0$. If the growth rates can change each day and an adversary distributes the daily unit of growth among the bamboos, then a (tight) makespan of $O(\log n)$ can still be achieved. If the gardener's algorithm is randomized and the adversary is *oblivious*, i.e., it is aware of gardener's algorithm but does not know the random bits or the previously trimmed bamboos, then the makespan is $O(m)$ with probability at least $1 - O(2^{-2^m})$, i.e., it is $O(\log \log n)$ with high probability [4]. The generalization of the cup game with multiple gardeners has been also addressed in [4, 15].

As we already mentioned, a problem closely related to BGT is the Pinwheel Scheduling problem that received a lot of attention in the literature [7, 8, 13, 14, 16, 19].

The BGT problem and its generalizations also appeared in a variety of other applications, ranging from deamortization, to buffer management in network switches, to quality of service in real-time scheduling (see, e.g., [3, 12, 1] and the references therein).

2 New bounds on the makespan of known BGT algorithms

In this section we provide an improved analysis on the makespan guaranteed by the `Reduce-Fastest`(x) strategy and the first analysis that upper bounds the makespan of `Reduce-Max` by a constant. In the rest of this section, we say that a bamboo b_i is trimmed at day d to specify that the schedule computed using the heuristic chooses b_i as the bamboo that has to be trimmed at the end of day d.

2.1 The analysis for `Reduce-Max`

Here we analyze the heuristic `Reduce-Max`, that consists in trimming the tallest bamboo at the end of each day (ties are broken arbitrarily).

▶ **Theorem 1.** *`Reduce-Max` guarantees a makespan of* 9*.*

Proof. We partition the bamboos into groups, that we call levels, according to their daily growth rates. More precisely, we say that bamboo b_i is of *level* $j \geq 1$ if $\frac{1}{2^j} \leq h_i < \frac{1}{2^{j-1}}$. Let K be the level of bamboo b_n and, for every j, with $1 \leq j \leq K$, let L_j denote the set of all the bamboos of level j.

For every $1 \leq j \leq K$, let $\sigma(j)$ be the maximum height ever reached by any bamboo of level $k \geq j$, with $\sigma(K+1) = 0$ by definition. In order to bound the makespan, it suffices to bound $\sigma(1)$. Rather than doing this directly, we will instead show that for $1 \leq j \leq K$, we have

$$\sigma(j) \leq \max\Big\{3, \sigma(j+1)\Big\} + 3\sum_{k=1}^{j} \frac{|L_k|}{2^j}. \tag{1}$$

Let $q \leq K$ be the level with lowest index such that $\sigma(q) \leq 3$ (if there is no such index, $q = K$). For any $j < q$ it holds $\max\Big\{3, \sigma(j+1)\Big\} = \sigma(j+1)$. As a consequence, the makespan is at most

$$\sigma(1) \leq 3 + \sum_{j=1}^{q} 3\sum_{k=1}^{j} \frac{|L_k|}{2^j} \leq 3 + 3\sum_{j=1}^{K}\sum_{k=1}^{j} \frac{|L_k|}{2^j}. \tag{2}$$

If bamboo b_i is of level s, then the bamboo stalk contributes $\sum_{j=s}^{K} \frac{1}{2^j} < \frac{2}{2^s} \leq 2h_i$ to the sum in (2). As $\sum_{i=1}^{n} h_i = 1$ by definition, it follows that the makespan is bounded by

$$\sigma(1) \leq 3 + 3\sum_{j=1}^{K}\sum_{k=1}^{j} \frac{|L_k|}{2^j} \leq 3 + 6\sum_{i=1}^{n} h_i = 9.$$

We now complete the proof by proving (1), which compares $\sigma(j)$ and $\sigma(j+1)$ for all j. Suppose that bamboo b_i has level j, and that at the end of day d_1 bamboo b_i achieves the maximum height ever reached by any bamboo of level j. Let $d_0 < d_1$ be the largest-numbered day prior to d_1 at the end of which either (a) a bamboo b_ℓ with level greater than j was trimmed, or (b) a bamboo b_ℓ with height less than 3 was trimmed. Because the `Reduce-Max` algorithm always trims the tallest bamboo, the height of b_i at the end of day d_0 is at most the height of b_ℓ at the end of day d_0, right before b_ℓ is trimmed. It follows that the height of b_i at the end of day d_1, right before b_i is trimmed, is at most $h_i(d_1 - d_0)$ greater than the height of b_ℓ at the end of day d_0, right before b_ℓ is trimmed. Since the height of b_ℓ at the end of day d_0 is at most $\max\{3, \sigma(j+1)\}$, it follows that

$$\sigma(j) \leq \max\{3, \sigma(j+1)\} + h_i(d_1 - d_0) < \max\{3, \sigma(j+1)\} + \frac{2}{2^j}(d_1 - d_0), \tag{3}$$

where in the last inequality we use the fact that $h_i < \frac{1}{2^{j-1}}$. Now, in order to prove (1), it suffices to show that $d_1 - d_0 \leq \frac{3}{2}\sum_{k=1}^{j} |L_k|$. By the definition of d_0, at any day $t \in [d_0+1, d_1]$ a bamboo of height at least 3 and with level equal or smaller than j is trimmed. We call a cut at day $t \in [d_0+1, d_1]$ a *repeated cut* if, at day t, a bamboo that was already trimmed at any day in $[d_0+1, t-1]$ is trimmed again, and a *first cut* otherwise. Note that each repeated cut trims a bamboo whose growth occurred entirely during days $[d_0+1, t-1]$ and that the total growth of the forest in the interval interval $[d_0+1, d_1]$ is $d_1 - d_0$. It means that at most $\frac{1}{3}$ of the cuts at day $t \in [d_0+1, d_1]$ can be repeated cuts, since at the end of each of these days a bamboo of height at least 3 is trimmed. On the other hand, the number of first cuts is bounded by the number of distinct bamboos with levels less or equal to j, i.e., by $\sum_{k=1}^{j} |L_k|$. It follows that the number of days in the window $[d_0+1, d_1]$ satisfies $d_1 - d_0 \leq \frac{1}{3}(d_1 - d_0) + \sum_{k=1}^{j} |L_k|$, and thus $d_1 - d_0 \leq \frac{3}{2}\sum_{k=1}^{j} |L_k|$ as desired. ◀

2.2 The analysis for `Reduce-Fastest`(x)

Here we provide an improved analysis of the makespan achieved by the `Reduce-Fastest`(x) strategy. The heuristic `Reduce-Fastest`(x) consists in trimming, at the end of each day, the bamboo with the fastest daily growth rate among those that have reached a height of at least x (ties are broken in favour of the bamboo with the smallest index).

▶ **Theorem 2.** *The makespan of* `Reduce-Fastest`(x)*, for a constant x such that $x > 1$, is upper bounded by* $\max\left\{x + \frac{x^2}{4(x-1)}, \frac{1}{2} + x + \frac{x^2}{4(x-\frac{1}{2})}\right\}$.

Proof. Let M be the makespan of `Reduce-Fastest`(x) and let b_i be one of the bamboos such that the maximum height reached by b_i is exactly M. Let $[d_0, d_1]$ be an interval of days such that b_i reaches the makespan in d_1 and d_0 is the last day in which b_i was trimmed before d_1 (d_0 may also be equal to 0). Let δ the first day in $[d_0, d_1]$ such that the height of b_i is at least x. For sake of simplicity we rename the interval $[\delta, d_1]$ as $[0, T]$, with $T = d_1 - \delta$. Let N be the number of distinct bamboos that are trimmed in $[0, T-1]$.

We now give some definitions. Let the *volume* V of the garden be the overall growth of the bamboo in the days of the interval $[0, T-1]$. Since the garden grows by $\sum_{i=1}^{n} h_i = 1$ per day, we have $V = T$. Consider the cut of a bamboo b_j on day $d \in [0, T-1]$. If b_j was cut at least once in $[0, d-1]$ we say that the cut is a *repeated cut* otherwise we will say that the cut is a *first cut*. The act of cutting bamboo b_j on a day $d \in [0, T-1]$ with a repeated cut *removes* an amount of volume that is equal to $(d - d')h_j$, where d' is the last day of $[0, d-1]$ in which b_j has been cut, if this is a repeated cut, and $d' = 0$ if this is a first cut. Finally, the *leftover volume* of a bamboo b_j is the overall growth of b_j that happened during interval $[0, T-1]$ and has not been cut by the end of day $T-1$.

We will now bound the amount V' of volume V that is removed by repeated cuts in the interval $[0, T-1]$. Notice that, for each bamboo b_j that is cut in the interval $[0, T-1]$, it holds that $h_j \geq h_i$. If b_j is cut for its first time at day d (among the days in $[0, T-1]$), then the removed volume will be at least $(d+1)h_j \geq (d+1)h_i$. Therefore, after all the N bamboos of the interval $[0, T-1]$ have been cut at least once, the amount of volume removed by first cuts will be at least $\sum_{j=i}^{N} jh_i = \frac{N(N+1)}{2} \cdot h_i$, since at most one bamboo is cut per day. Moreover, if b_j is cut for its last time at day $T-1-d$ (among the days in $[0, T-1]$), b_j will have a height of dh_i at the end of day $T-1$. Finally, bamboo h_i is never cut in the interval $[0, T-1]$ and hence during the interval $[0, T-1]$ it grows by exactly Th_i. This means that the overall leftover volume will be at least $\sum_{j=1}^{N}(j-1)h_i + Th_i = \frac{N(N-1)}{2} \cdot h_i + Th_i$.

We can then write

$$V' \leq V - \left(\frac{N(N+1)}{2} + \frac{N(N-1)}{2}\right) \cdot h_i - Th_i = V - N^2 h_i - Th_i = T(1-h_i) - N^2 h_i,$$

where the last equality follows from $V = T$.

Since in $[0, T-1]$ the bamboo b_i has height at least x, each repeated cut removes at least x units of volume from V'. Therefore, the number N' of repeated cuts is at most $\frac{V'}{x} \leq \left(T(1-h_i) - N^2 h_i\right)/x$. We now use this upper bound on N' to derive an upper bound to the time T:

$$T = N + N' \leq N + \frac{T(1-h_i) - N^2 h_i}{x}.$$

For $T'(N) = (Nx - N^2 h_i)/(h_i + x - 1)$, the above formula implies $T \leq T'(N)$. If we fix h_i and x, $T'(N)$ is a concave downward parabola that attains its maximum in its vertex at

$N = x/2h_i$. Thus:

$$T \le T'(x/2h_i) \le \frac{\frac{x^2}{2h_i} - \frac{x^2}{4h_i}}{h_i + x - 1} = \frac{x^2}{4h_i(h_i + x - 1)}.$$

Using this upper bound to T we now bound the overall growth of the bamboo b_i, i.e., the makespan M. At day $d = 0$, b_i has height at most $x + h_i$ by our choice of δ, and in the next T days it grows by Th_i. Hence:

$$M \le x + h_i + Th_i < x + h_i + \frac{x^2}{4(h_i + x - 1)}. \tag{4}$$

Let $M'(h_i) = x + h_i + \frac{x^2}{4(h_i+x-1)}$. The derivative w.r.t. h_i of the above formula is

$$\frac{\partial M'}{\partial h_i} = 1 - x^2/4(h_i + x - 1)^2 = \frac{4(h_i + x - 1)^2 - x^2}{4(h_i + x - 1)^2} = \frac{(x + 2h_i - 2)(3x + 2h_i - 2)}{4(h_i + x - 1)^2}.$$

The denominator is always positive, and the numerator is a concave upward parabola having its two roots at $h_i = 1 - 3x/2$ and at $h_i = 1 - x/2$. Let us briefly restrict ourselves to the case $h_i \le \frac{1}{2}$ and notice that, since $x > 1$, the first root is always negative, while the second root is always smaller than $\frac{1}{2}$. It follows that the maximum of $M'(h_i)$ is attained either at $h_i = 0$ or at $h_i = \frac{1}{2}$. Substituting in Equation 4 we get:

$$M \le \max \left\{ x + \frac{x^2}{4(x-1)}, x + \frac{1}{2} + \frac{x^2}{4(x - \frac{1}{2})} \right\}$$

As far as the case $h_i > \frac{1}{2}$ is concerned, notice that it implies $i = 1$ (since if $i \ge 2$ we would have the contradiction $\sum_{i=1}^{n} h_i > \frac{1}{2} \cdot i = 1$) and hence bamboo b_1 is trimmed as soon as its height reaches at least x. The makespan M must then be less than $x + h_1 < x + 1$, which is always smaller than $x + \frac{1}{2} + \frac{x^2}{4(x-\frac{1}{2})} > x + \frac{1}{2} + \frac{1}{2}$. ◀

▶ **Corollary 3.** *The makespan of* `Reduce-Fastest`(2) *is at most* $19/6$ *and the makespan of* `Reduce-Fastest`$(1 + \frac{1}{\sqrt{5}})$ *is at most* $1 + \phi < 2.62$*, where* ϕ *is the golden ratio.*

3 Trimming oracles

This section is devoted to the design of trimming oracles. More precisely, we first design two trimming oracles that implement `Reduce-Fastest`(x) and `Reduce-Max`, respectively. The trimming oracle that implements `Reduce-Fastest`(x) has a $O(\log n)$ worst-case query time, uses linear size and can be built in $O(n \log n)$ time. The trimming oracle that implements `Reduce-Max` has a $O(\log^2 n)$ worst-case query time or a $O(\log n)$ amortized query time, uses linear space, and can be built in $O(n \log n)$ time. We conclude this section by designing a novel trimming oracle that guarantees a makespan of 2 and has a $O(\log n)$ amortized query time. The oracle uses linear size and can be built in $O(n \log n)$ time. For technical convenience, in this section we index days starting from 0, so that at the end of day 0 the gardener can already trim the first bamboo.

An interactive implementation of the Trimming Oracles described in this section is available at `https://www.isnphard.com/g/bamboo-garden-trimming/`.

3.1 A Trimming Oracle implementing `Reduce-Fastest`(x)

We now describe our trimming oracle implementing `Reduce-Fastest`(x). The idea is to keep track, for each bamboo b_i, of the next day d_i at which b_i will be at least as tall as x. When a query at a generic day D is performed, we will then return the bamboo b_i with minimum index i among the ones for which $d_i \geq D$.

To this aim we will make use of a *priority search tree* [17] data structure T to dynamically maintain a collection $P = \{(x_1, y_1), (x_2, y_2), \dots\}$ of 2D points with distinct y coordinates in $\{1, \dots, n\}$ under insertions and deletions while supporting the following queries:

MinYInXRange(T, x_0): report the minimum y-coordinate among those of the points (x_i, y_i) for which $x_i \leq x_0$, if any.

GetX(T, y): report the x-coordinate x_i of the (at most one) point (x_i, y_i) for which $y_i = y$, if any.

All of the above operations on T require time $O(\log |P|)$, as long as all coordinates and query parameters fit in $O(1)$ words of memory.[1]

In our case, the points (x_i, y_i) will be the pairs (d_i, i) for $i = 1, \dots, n$. In such a way, a MinYInXRange query with $x_0 = D$ will return exactly the index i of the bamboo b_i to be cut at the end of day D, if any. After cutting b_i, we *update* T to account for the new day at which the height b_i will be at least x, i.e., we replace the old point (d_i, i) with $(D + \lceil x/h_i \rceil, i)$. Unfortunately, since the trimming oracle is ought to be used perpetually, (the representations of) both d_i and D will eventually require more than a constant number of memory words.

We solve this problem by dividing the days into contiguous intervals $I_0, I_1, \dots$ of n days each, where $I_j = [nj, nj+1, \dots, n(j+1) - 1]$, and by using two priority search trees T_1 and T_2 that are associated with the current and the next interval, respectively. This allows us to measure days from the start of the current interval I_j, i.e., if $D = nj + \delta \in I_j$, then we only need to keep track of $\delta \in [0, \dots, n-1]$. In place of (d_i, i), we store the point (δ_i, i) in T_1, where $\delta_i = d_i - nj$. In this way, the previous query with $x_0 = D$ will now correspond to a query with $x_0 = \delta$.

Finally, we also ensure that at the end of the generic day $D = nj + \delta$, T_2 contains the point (δ'_i, i) for each $d_i = n(j+1) + \delta'$ and $i = 1, \dots, \delta + 1$. This allows us to swap T_2 for T_1 when interval I_j ends.

Since bamboo b_i reaches height x exactly $\lceil x/h_i \rceil$ days after being cut, it follows that the largest x-coordinate ever stored in T_1 or T_2 is at most $n + x/h_n$ and we can then support MinYInXRange queries in $O(\log(n))$ time (where we are assuming that x, h_n and thus x/h_n fit in a constant number of memory words).

The pseudocode of our trimming oracle is as given in Algorithm 1. The procedure Query() is intended to be run every day. Consider a generic day δ of the current interval I_j. At this time, T_1 correctly encodes all the days at which the bamboos reached, or will reach, height at least x when measured from the starting day of the current interval (i.e., from day nj), and after all the cuts of the previous days have already been performed.[2] The same information concerning bamboos $b_1, \dots, b_\delta$ is replicated in T_2 with respect to the starting time of the next interval (i.e., $(n+1)j$). The procedure Query() accomplishes two tasks: (1) it computes the bamboo b_i to cut at the end of day δ of the current interval (if any) and it updates the

---

[1] While this query is not described in [17], it can be easily implemented in $O(\log |P|)$ time using a dictionary and the fact that y-coordinates are distinct.

[2] Actually, if a bamboo b_i reached height x before the beginning of the considered interval, we will store the point $(0, i)$ in place of (δ_i, i) with $\delta_i < 0$. This still encodes the fact that it is possible to trim b_i from the very fist day of the interval and prevents δ_i from becoming arbitrarily small.

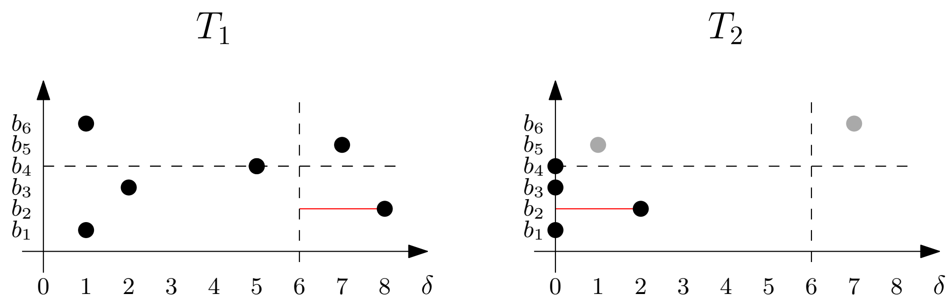

Figure 2 An example of the points contained in the priority search trees T_1 and T_2 for an instance with 6 bamboos at the end of day $\delta = 4$ of a generic interval I_j. We labeled the y-coordinate i with b_i since the unique point (d_i, i) having y-coordinate i represents the day at which b_i reached/will reach a height of at least x. Notice that the points corresponding to bamboos b_1, b_2, b_3, and b_4 are already updated in T_2, while b_5 and b_6 (shown in gray) will be updated by the days $\delta = 5$ and $\delta = 6$, respectively. At the end of day $\delta = 6$, all the points in T_2 are updated and T_1 can be safely swapped with T_2.

data structures T_1 and T_2 to account for the new height of b_i; (2) it updates the information concerning $b_{\delta+1}$ in T_2. See Figure 2 for an example.

The following theorem summarizes the performances of our trimming oracle.

▶ **Theorem 4.** *There is a Trimming Oracle implementing* `Reduce-Fastest`(x) *that uses* $O(n)$ *space, can be built in* $O(n \log n)$ *time, and can report the next bamboo to trim in* $O(\log n)$ *worst-case time.*

3.2 A Trimming Oracle implementing `Reduce-Max`

The idea is to maintain collection L of n lines $\ell_1, \ldots, \ell_n$ in which $\ell_i(d) = h_i d + c_i$ is associated with bamboo b_i and represents its height at the end of day d. Initially $c_i = h_i$.

Determining the bamboo b_i to trim at a generic day d then corresponds to finding the index i that maximizes $\ell_i(d)$. After bamboo b_i, previously of height H, has been cut, ℓ_i needs to be updated to reflect the fact that b_i has height 0 at time d, which corresponds to decreasing c_i by H.

The *upper envelope* $\mathcal{U}_L$ of L is a function defined as $\mathcal{U}_L(d) = \max_{\ell \in L} \ell(d)$. We make use of an *upper envelope data structure* U that is able to maintain L under insertions, deletions and lookups of named lines and supports the following query operation:

Upper(U, d) return a line $\ell \in L$ for which $\ell(d) = \mathcal{U}_L(d)$.

Unfortunately, the trivial implementation of the trimming oracle suggested by the above description encounters similar problems as the ones discussed in Section 3.1 for `Reduce-Fastest`(x): the current day d and the coefficients c_i will grow indefinitely, thus affecting the computational complexity.

Once again, we solve this problem by using two copies U_1, U_2 of the previous *upper envelope data structure* and by dividing the days into intervals $I_1, I_2, \ldots$ with $I_j = [nj, nj + 1, \ldots, n(j+1) - 1]$. At the beginning of the current day $D = nj + \delta \in I_j$, U_1 will contain all lines $\ell_1, \ldots, \ell_n$ and the value of each $\ell_i(\delta)$ will be exactly the height of b_i. Moreover, at the end of day D (i.e., after the highest bamboo of day D has been trimmed), U_2 will contain a line ℓ'_i for each $i \leq \delta + 1$ such that $\ell'_i(\delta')$ with $\delta' \in [0, n-1]$ is exactly the height reached by b_i on day $n(j+1) + \delta'$ if it is not trimmed on days $nj + \delta + 1, \ldots, n(j+1) + \delta' - 1$. This means that at the end of day $nj + (n-1)$, U_2 correctly describes the heights of all bamboos

Algorithm 1 Trimming Oracle for `Reduce-Fastest(x)`

```
1  Function Build():
2      δ ← 0;
3      T_1, T_2 ← Pointers to two empty priority search trees;
4      h_1, ..., h_n ← Sort the growth rates of the n bamboo in nonincreasing order;
5      for i = 1 ..., n do
6          Insert(T_1, (⌈x/h_i⌉ − 1, i))
7  Function Update(T, δ_i, i):
8      δ'_i ← GetX(T, i);
9      if δ'_i exists then Delete((δ'_i, i));
10     Insert(T, (max{0, δ_i}, i));
11 Function Query():
       // Cut fastest bamboo b_i that reached height x by day δ
12     i ← MinYInXRange(T_1, δ);
13     if i exists then
14         Update(T_1, δ + ⌈x/h_i⌉, i) ;
15         Update(T_2, δ + ⌈x/h_i⌉ − n, i) ;
       // Make sure that bamboo b_{δ+1} is updated in T_2
16     δ_{δ+1} ← GetX(T_1, δ + 1);
17     Update(T_2, δ_{δ+1} − n, δ + 1)
       // Move to the next day and possibly to the next interval
18     δ ← (δ + 1) mod n;
19     if δ = 0 then Swap T_1 and T_2;
20     if i exists then return "Trim bamboo b_i" else return "Do nothing";
```

in the next interval I_{j+1} as a function of δ', and we can safely swap U_1 with U_2. See Figure 3 for an example.

The pseudocode of our trimming oracle is given in Algorithm 2. A technicality concerns the initial construction of the set of lines in U_1. Notice that this is not handled by the Build() function, but we iteratively add $\ell_1, \ldots, \ell_n$ during the first n calls to Query() (i.e., during the days of interval I_0). We can safely do this since the `Reduce-Max` strategy ensures that at time $D \in I_0$ only bamboos in $\{b_1, \ldots, b_{D+1}\}$ can conceivably be trimmed. This is handled by the test of line 9, which is only true for $D \in I_0$ and will impact our amortized bounds, as noted below.

The performances of our trimming oracle depend on the specific implementation of the upper envelope data structure use. In [18], such a data structure guaranteeing a worst-case time of $O(\log^2 n)$ per operation is given, while a better amortized bound of $O(\log n)$ per operation was obtained in [6].[3] Moreover, from Theorem 1 we know that the makespan of `Reduce-Max` is at most constant, implying that the maximum absolute value of a generic coefficient c_i is at most $O(nh_i) = O(n)$.

The following theorem summarizes the time complexity of our trimming oracle.[4]

[3] Actually, the authors of [18] and [6] design a dynamic data structure to maintain the convex hull of a set of points in the plane. As explained in [6], point-line duality can be used to convert such a structure into one maintaining the upper envelope of a set of linear functions.

[4] Due to lines 9 and 10, the complexity of a query operation is only amortized over the running time of

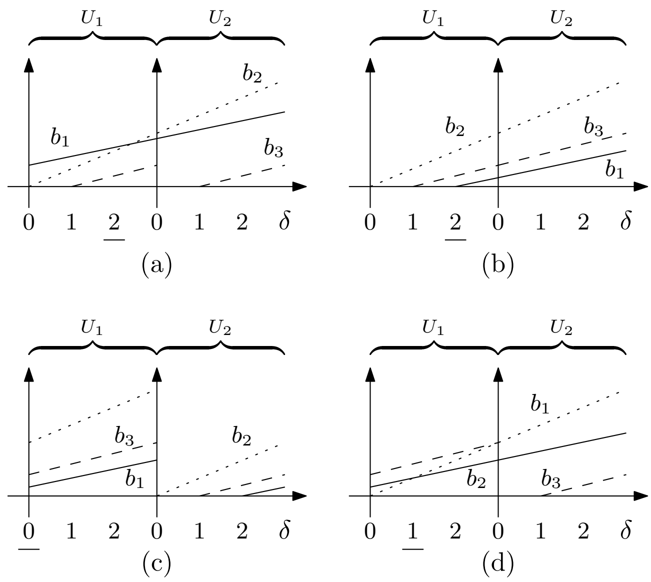

Figure 3 An example of the points contained in U_1 and U_2 for an instance with 3 bamboos, at the beginning of day 2 of a generic interval I_j (a), at the end of day 2 of I_j but before moving to I_{j+1} (b), at beginning of day 0 of I_{j+1} (c), and at the beginning of day 1 of I_{j+1} (d).

▶ **Theorem 5.** *There is a Trimming Oracle implementing* `Reduce-Max` *that uses $O(n)$ space, can be built in $O(n \log n)$ time, and can report the next bamboo to trim in $O(\log^2 n)$ worst-case time, or $O(\log n)$ amortized time.*

3.3 A Trimming Oracle achieving makespan 2

We now design a Trimming Oracle implementing a perpetual schedule that achieves a makespan of at most 2.

We start by rounding the rates $h_1, \ldots, h_n$ down to the previous power of $\frac{1}{2}$ as in [11], i.e., we set $h'_i = 2^{\lfloor \log_2 h_i \rfloor}$. We will then provide a perpetual schedule for the rounded instance achieving makespan at most 1 w.r.t. the new rates $h'_1, \ldots, h'_2$. Since $h_i \leq 2h'_i$, this immediately results in a schedule having makespan at most 2 in the original instance.

Henceforth we assume the input instance is already such that each h_i is a power of $\frac{1}{2}$. Moreover, we will also assume that $\sum_{i=1}^{n} h_i = 1$. Indeed, if $\sum_{i=1}^{n} h_i < 1$ then we can artificially increase some of the growth rates to meet this condition. Clearly, any schedule achieving makespan of most 1 for the transformed instance, also achieves makespan at most 1 in the non-transformed instance.

We transform the instance as follows: we iteratively consider the bamboos in nonincreasing order of rates; when b_i is considered we update h_i to $2^{\left\lfloor \log_2 \left(1 - \sum_{j \neq i} h_j\right) \right\rfloor}$, i.e., to the highest rate that is a power of $\frac{1}{2}$ and still ensures that the sum of the growth rates is at most 1. One can easily check that the above procedure yields an instance for which $\sum_{i=1}^{n} h_i = 1$, as otherwise $\sum_{i=1}^{n} h_i < 1$ and $1 - \sum_{i=1}^{n} h_i \geq h_n$, which is a contradiction since h_n would have been increased to $2h_n$. This requires $O(n \log n)$ time.

previous queries.

Algorithm 2 Trimming Oracle for `Reduce-Max`

Function `Build()`:
- $\delta \leftarrow 0$;
- $T_1, T_2 \leftarrow$ Pointers to two empty upper envelope data structures;
- $h_1, \dots, h_n \leftarrow$ Sort the growth rates of the n bamboo in nonincreasing order;

Function `Update(`U, i, c`)`:
- `Delete(`U, ℓ_i`)`;
- `Insert(`$U, \ell_i(d) = h_i d + c$`)`;

Function `Query()`:
- `// Ensure that the line` $\ell_{\delta+1}$ `corresponding to bamboo` $b_{\delta+1}$ `is in` U_1
- **if** *there is no line named* $\ell_{\delta+1}$ *in* U_1 **then**
  - `Insert(`$U_1, \ell_{\delta+1}(d) = h_{\delta+1} d + h_{\delta+1}$`)`;
- `// Cut highest bamboo` b_i `at day` δ
- $\ell_i(d) = h_i d + c_i \leftarrow$ `Upper(`δ`)`;
- `Update(`$U_1, i, -\delta h_i$`)`;
- `Update(`$U_2, i, (n-\delta) h_i$`)`;
- `// Ensure that the line` $\ell_{\delta+1}$ `corresponding to bamboo` $b_{\delta+1}$ `is updated in` U_2
- Let $\ell_{\delta+1}(d) = h_{\delta+1} d + c_{\delta+1}$ be the line named $\ell_{\delta+1}$ in U_1;
- `Update(`$U_2, \delta+1, n h_{\delta+1} + c_{\delta+1}$`)`;
- `// Move to the next day and possibly to the next interval`
- $\delta \leftarrow (\delta + 1) \bmod n$;
- **if** $\delta = 0$ **then** Swap U_1 and U_2;
- **return** "Trim bamboo b_i";

In the rest of this section, we will design Trimming Oracles achieving a makespan of at most 1 for instances where all h_is are powers of $\frac{1}{2}$ and $\sum_{i=1}^{n} h_i = 1$.

A Trimming Oracle for regular instances

Let us start by considering an even smaller subset of the former instances, namely the ones in which b_i has a growth rate of $h_i = 2^{-i}$, for $i = 1, \dots, n-1$, and $h_n = h_{n-1} = 2^{-n+1}$. For the sake of brevity we say that these instances are *regular*.[5]

It turns out that a schedule for regular instances can be easily obtained by exploiting a connection between the index i of bamboo b_i to be cut at a generic day D and the position of the least significant 0 in the last $n-1$ bits in the binary representation of D.

The schedule is as follows: if the last 0 in the binary representation of D appears in the i-th least significant bit, with $i < n$, then b_i is to be cut at the end of day D. Otherwise, if the $n-1$ least significant bits of D are all 1, bamboo b_n is cut at day D.

In this way, the maximum number of days that elapses between any two consecutive cuts of bamboo b_i with $i < n$ is $M_i = 2^i$, while b_n is cut every $M_n = 2^{n-1}$ days. It is then easy to see that, for each bamboo b_i, $h_i \cdot M_i = 1$, thus showing that the resulting makespan is 1 as desired (and this is tight since, in any schedule with bounded makespan, b_1 grows for at least 2 consecutive days). See Figure 4 for an example with $n = 5$.

[5] Notice that, in any regular instance, the grow rates of the bamboos are completely specified by the number n.

D	$(D)_2$	Trim
0	0 0 0 0	b_1
1	0 0 0 1	b_2
2	0 0 1 0	b_1
3	0 0 1 1	b_3

D	$(D)_2$	Trim
4	0 1 0 0	b_1
5	0 1 0 1	b_2
6	0 1 1 0	b_1
7	0 1 1 1	b_4

D	$(D)_2$	Trim
8	1 0 0 0	b_1
9	1 0 0 1	b_2
10	1 0 1 0	b_1
11	1 0 1 1	b_3

D	$(D)_2$	Trim
12	1 1 0 0	b_1
13	1 1 0 1	b_2
14	1 1 1 0	b_1
15	1 1 1 1	b_5

Perpetual schedule: $b_1, b_2, b_1, b_3, b_1, b_2, b_1, b_4, b_1, b_2, b_1, b_3, b_1, b_2, b_1, b_5 \ldots$

Figure 4 A perpetual schedule of a regular instance with 5 bamboos.

This above relation immediately suggests the implementation of a Trimming Oracle that maintains the binary representation of $D \bmod 2^{n-1}$. Since it is well known that a binary counter with n bits can be incremented in $O(1)$ amortized time [9, pp. 454–455], we can state the following:

▶ **Lemma 6.** *For the special case regular instances, there is a Trimming Oracle that uses $O(n)$ space, can be built in $O(n)$ time, can be queried to report the next bamboo to cut in $O(1)$ amortized time, and achieves makespan* 1*.*

A Trimming Oracle for non-regular instances

Here we show how to design a Trimming Oracle for non-regular instances by iteratively transforming them into suitable regular instances. We will refer to the bamboos $b_1, \ldots, b_n$ as *real bamboos* and will introduce the notion of *virtual bamboos*.

A virtual bamboo v represents a collection of (either real or virtual) bamboos whose growth rates yield a regular instance when suitably scaled by a common factor. The growth rate of v will be equal to the sum of the growth rates of the bamboos in its collection.

To see why this concept is useful, consider an example instance I with 6 bamboos $b_1, \ldots, b_6$ with rates $h_1 = \frac{1}{2}$, $h_2 = \frac{1}{8}$, $h_3 = \frac{1}{8}$, $h_4 = \frac{1}{8}$, $h_5 = \frac{1}{16}$, $h_6 = \frac{1}{16}$. If we replace h_4, h_5, and h_6 with a virtual bamboo v with growth rate $\mathfrak{h} = \frac{1}{8} + \frac{1}{16} + \frac{1}{16} = \frac{1}{4}$ we obtain the related regular instance I' in which the bamboos b_1, v, b_2, b_3 have growth rates $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$, and $\frac{1}{8}$, respectively. Notice also that the collection of bamboos associated with v is a regular instance I_v once all the rates are multiplied by $\frac{1}{\mathfrak{h}} = 4$. We can now build two Trimming Oracles $\mathcal{O}'$ and $\mathcal{O}_v$ for I' and I_v, respectively, by using Lemma 6. It turns out that $\mathcal{O}'$ and $\mathcal{O}_v$ together allow us to build an oracle $\mathcal{O}_r$ for I as well, which can be represented as a tree (See Figure 5). In general, our oracles $\mathcal{O}$ will consist of a tree $T_{\mathcal{O}}$ whose leaves are the real bamboos $b_1, \ldots, b_n$ of the input instance and in which each internal vertex u serves two purposes: (i) it represents a virtual bamboo whose associated collection C contains the bamboos associated to the children of u; and (ii) it serves as a Trimming Oracle $\mathcal{O}_u$ over the bamboos in C.[6] In order to query $\mathcal{O}$ we proceed as follows: initially we start with a pointer p to the root r of $T_{\mathcal{O}}$; then, we iteratively check whether p points to a leaf ℓ or to an internal vertex u. In the former case, we trim the real bamboo associated with ℓ, otherwise we query the Trimming Oracle $\mathcal{O}_u$ associated with u and we move p to the child of u corresponding to the (virtual or real) bamboo returned by the query on $\mathcal{O}_u$. Since all queries on internal vertices can be performed in $O(1)$ amortized time (see Lemma 6), the amortized time required to query $\mathcal{O}$ is proportional to the height of $T_{\mathcal{O}}$. See Figure 5 for the schedule associated to our example instance I.

[6] The root of $T_{\mathcal{O}}$ can be seen as a virtual bamboo with a growth rate of 1.

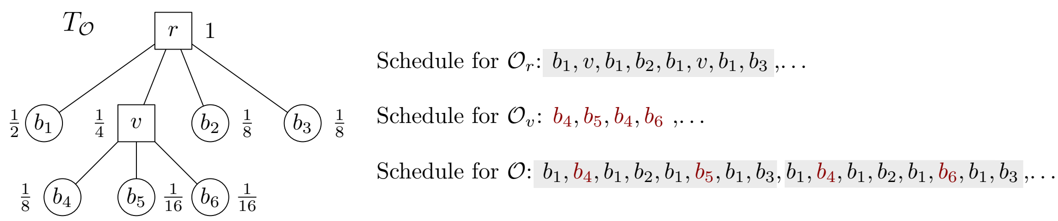

Figure 5 The tree $T_{\mathcal{O}}$ of the Trimming Oracle $\mathcal{O}$ for the instance with 6 bamboos $b_1, \dots, b_6$ with rates $h_1 = \frac{1}{2}$, $h_2 = \frac{1}{8}$, $h_3 = \frac{1}{8}$, $h_4 = \frac{1}{8}$, $h_5 = \frac{1}{16}$, and $h_6 = \frac{1}{16}$. Bamboos b_4, b_5, and b_6 have been replaced by a virtual bamboo v (and a corresponding oracle $\mathcal{O}_v$) with a virtual growth rate of $\frac{1}{4}$. The root r represents both a virtual bamboo with growing rate 1 and the corresponding Trimming Oracle $\mathcal{O}_r$ for the regular instance consisting of b_1, v, b_2, and b_3.

Before showing how to build the tree $T_{\mathcal{O}}$ of our Trimming Oracle $\mathcal{O}$, we prove that the perpetual schedule obtained by querying $\mathcal{O}$ achieves a makespan of at most 1. At any point in time, we say that the *virtual height* of a virtual bamboo v representing a collection C of (real or virtual) bamboos is the maximum over the (real or virtual) heights of the bamboos in C. The bound on the makespan follows by instantiating the following Lemma with $b = r$ and $h = 1$, and by noticing that: (i) the root r of $T_{\mathcal{O}}$ is scheduled every day, and (ii) that the maximum virtual height of r is exactly the makespan.

▶ **Lemma 7.** *Let b be a (real or virtual) bamboo with growth rate h. If b is scheduled at least once every $\frac{1}{h}$ days, then the maximum (real or virtual) height ever reached by b will be at most* 1.

Proof. The proof is by induction on the number η of nodes in the subtree rooted at the vertex representing b in $T_{\mathcal{O}}$.

If $\eta = 1$ then b is a real bamboo and the claim is trivially true since the maximum height reached by b can be at most $h \cdot \frac{1}{h} = 1$.

Suppose then that $\eta \geq 2$ and that the claim holds up to $\eta - 1$. Bamboo b must be a virtual bamboo representing some set $C = \{b'_1, b'_2, \dots, b'_k\}$ of (real or virtual) bamboos which appear as children of b in $T_{\mathcal{O}}$ and are such that: (i) for $i = 1, \dots, k-1$, b'_i has a growth rate of $h'_i = h/2^i$, and (ii) b'_k has a growth rate of $h'_k = h/2^{k-1}$.

Virtual bamboo b schedules the bamboos in C by using the oracle $\mathcal{O}_v$ of Lemma 6, on the regular instance obtained by changing the rate of bamboo b'_i from h'_i to $h''_i = h'_i/h$.

Let d_i (resp. d'_i) be the maximum number of days between any two consecutive cuts of bamboo b'_i according to to the schedule produced by $\mathcal{O}$ (resp. $\mathcal{O}_v$). We know that $d'_i \cdot h''_i \leq 1$ (as otherwise the schedule of $\mathcal{O}_v$ would result in makespan larger than 1 on a regular instance, contradicting Lemma 6), i.e., $d'_i \leq \frac{1}{h''_i}$. Since, b is scheduled at least every $1/h$ days by hypothesis, we have that $d_i \leq \frac{1}{h \cdot h''_i} = \frac{h}{h \cdot h'_i} = \frac{1}{h'_i}$ and hence, by inductive hypothesis, the maximum height reached by b'_i will be at most 1. ◀

We now describe an algorithm that constructs a tree $T_{\mathcal{O}}$ of logarithmic height.

The algorithm employs a collection of sets $S_0, S_1, \dots,$ where initially $S_0 = \{b_1, \dots, b_n\}$ contains all the real bamboos of our input instance, and S_i with $i > 0$ is obtained from S_{i-1} by performing suitable merge operations over the bamboos in S_{i-1}. A merge operation on a collection $C \subseteq S_{i-1}$ of bamboos, whose growth rates yield a regular instance when multiplied by some common factor, consists of: updating S_{i-1} to $S_{i-1} \setminus C$, *creating* a new virtual bamboo v representing C, and adding v to S_i.

The algorithm works in phases. At the generic phase $i = 1, 2, \ldots$, it iteratively: (1) looks for a bamboo b with the largest growth rate that can be involved in a merge operation and (2) perform a merge operation on a maximal set $C \subseteq S_{i-1}$ among the ones that contain b (and on which a merge operation can be performed). The procedure is then repeated from step (1) until no suitable bamboo b exists anymore. At this point we name R_{i-1} the current set S_{i-1}, we add to S_i all the bamboos in R_{i-1}, and we proceed to the next phase. The algorithm terminates whenever the set S_i constructed at the end of a phase contains a single virtual bamboo r (of rate 1).

The sequence of merge operations implicitly defines a bottom-up construction of the tree $T_{\mathcal{O}}$, where every merge operation creates a new internal vertex associated with its corresponding virtual bamboo. The root of $T_{\mathcal{O}}$ is r and the height of $T_{\mathcal{O}}$ coincides with the number of phases of the algorithm.

▶ **Lemma 8.** *The algorithm terminates after at most $O(\log n)$ phases.*

Proof. We first prove that the algorithm must eventually terminate. This is a direct consequence of the fact that, at the beginning of any phase i, every set S_{i-1} containing 2 or more bamboos, admits at least one merge operation. Indeed, since merge operations preserve the sum of the growth rates, the overall sum of the rates of the bamboos in S_{i-1} must be 1. Consider now a bamboo $b \in S_{i-1}$ having the lowest growth rate h. Since all rates are powers of $\frac{1}{2}$ and must sum to 1, there must be at least one other bamboo $b' \in S_{i-1} \setminus \{b\}$ having rate h, implying that merge operation can be performed on $C = \{b, b'\}$.

It remains to bound the number of phases. We prove by induction on i that any internal vertex/virtual bamboo v of $T_{\mathcal{O}}$ created at phase i has at least 2^i leaves as descendants. The base case $i = 1$ is trivial since the merge operation that created v must have involved at least 2 real bamboos.

Consider now the case $i \geq 2$. We will show that v was created by a merge operation on a collection C containing at least 2 bamboos v', v'' that were, in turn, created during phase $i - 1$. Hence, by inductive hypothesis, the number of leaves that are descendants of v is the sum of the number of leaves that are descendants of v' and v'', respectively, i.e., it is at least $2^{i-1} + 2^{i-1} = 2^i$.

Let $C \subseteq S_{i-1}$ be the set of bamboos used in the merge operation that created v, and let h be the smallest growth rate among the ones of the bamboos in C. Notice that, by definition of merge operation, there must be 2 distinct bamboos v', v'' with rate h in C. We will now show that v' and v'' were created during phase $i-1$. We proceed by contradiction. If neither of v' and v'' were created in phase of $i - 1$, then $\{v', v''\} \subseteq R_{i-2}$ which is impossible since $\{v', v''\}$ would have been a feasible merge operation in phase $i-2$. Assume then that v' was not created in phase $i - 1$, while v'' was created in phase $i - 1$, w.l.o.g. Then, $v' \in R_{i-2}$, while v'' was obtained from a merge operation on a set $C' \subseteq S_{i-2}$ performed in phase $i - 1$. Since the growth rate of v'' is h, the fastest growth rate among the ones of the bamboos in C' must be $h/2$. Hence, the set $C'' = \{v'\} \cup C'$ was a feasible merge operation in phase $i - 1$ when v'' was created. This is a contradiction since $C' \subset C''$ was not a maximal set, as required by the algorithm. ◀

Next Lemma bounds the computational complexity of constructing our oracle.

▶ **Lemma 9.** *The Trimming Oracle $\mathcal{O}$ can be built in $O(n \log n)$ time.*

Proof. It suffices to prove that every phase i of our algorithm can be implemented in $O(n)$ time, since from Lemma 8 the number of phases is $O(\log n)$.

We maintain the set S_{i-1} as a doubly linked list L_{i-1} in which each node ℓ is associated with a distinct growth rate $\mathfrak{h}_\ell$ attained by at least one bamboo in S_{i-1} and stores the set $H(\ell)$ of bamboos of S_{i-1} with grow rate $\mathfrak{h}_\ell$. Nodes appear in decreasing order of $\mathfrak{h}_\ell$. The very first list L_0 can be constructed in $O(n \log n)$ time by sorting the growth rates of the bamboos in S_0. We now show how to build L_i in $O(n)$ time.

The idea is to iteratively find two nodes ℓ_1, ℓ_2 of L_{i-1} such that: (i) ℓ_2 is not the head of L_{i-1} and appears not earlier than ℓ_1; (ii) if ℓ_1 is not the head of L_{i-1}, then selecting one bamboo from the set $H(\ell)$ of each node ℓ that appears before the predecessor ℓ'_1 of ℓ_1, and two bamboos from the set $H(\ell'_1)$ yields the (maximal) set C corresponding the merge operation that algorithm performs; and (iii) all the bamboos in the sets $H(\ell)$ of the nodes ℓ that appear not earlier than ℓ_1 and before ℓ_2 in L_{i-1} will not participate in any merge operation of phase i. We call the set of these bamboos D (notice that it is possible for ℓ_2 to be equal to ℓ_1, in which case no such node ℓ exists and $D = \emptyset$).

To find ℓ_1 and ℓ_2 notice that ℓ_2 is the the last node of L_{i-1} for which any two consecutive nodes preceding ℓ_2 correspond to consecutive rates[7], while the predecessor ℓ'_1 of ℓ_1 is the last node that appears before ℓ_2 and such that $|H(\ell'_1)| \geq 2$.

We now delete the bamboos in $C \cup D$ from their respective sets $H(\ell)$ of L_{i-1}, create a new virtual bamboo v by a merge operation on C. Finally, delete from L_{i-1} all nodes ℓ whose set $H(\ell)$ is now empty. We then repeat this procedure from the beginning until L_{i-1} is empty.

Concerning the time complexity, notice that finding ℓ_1 and ℓ_2 requires $O(k)$ time, where k is the number of nodes that precede ℓ_2 in L_{i-1}. Moreover, all the other steps can be implemented in $O(k)$ time. Therefore, we are able delete k bamboos from L_{i-1} in $O(k)$ time, and hence the overall time complexity to delete all bamboos in L_{i-1} is $O(n)$.

Finally, by keeping track of the sets D, of all the virtual bamboos v generated during the iterations, and by using the fact that the rates of the virtual bamboos are monotonically decreasing, it is also possible to build L_i in $O(n)$ time. ◀

▶ **Lemma 10.** *The Trimming Oracle $\mathcal{O}$ uses $O(n)$ space.*

Proof. By Lemma 6 each internal vertex of $T_\mathcal{O}$ maintains a Trimming Oracle with size proportional to the number of its children, implying that the overall space required by $\mathcal{O}$ is proportional to the number η of vertices of $T_\mathcal{O}$. Since every internal vertex in $T_\mathcal{O}$ has at least 2 children, we have that $\eta = O(n)$. ◀

By combing Lemma 7, Lemma 8, Lemma 9, and Lemma 10, we can state the following theorem that summarizes the result of this section:

▶ **Theorem 11.** *There is a Trimming Oracle that achieves makespan 2, uses $O(n)$ space, can be built in $O(n \log n)$ time, and can report the next bamboo to trim in $O(\log n)$ amortized time.*

Acknowledgements

The authors would like to thank Francesca Marmigi for the picture of the robotic panda gardener in Figure 1. We are also grateful to an anonymous reviewer whose comments allowed us to significantly simplify the analysis of `Reduce-Max`.

[7] For technical simplicity, when all consecutive nodes of L_{i-1} correspond to consecutive rates we allow ℓ_1 and/or ℓ_2 to point one position past the end of L_{i-1}.

References

1. Micah Adler, Petra Berenbrink, Tom Friedetzky, Leslie Ann Goldberg, Paul W. Goldberg, and Mike Paterson. A proportionate fair scheduling rule with good worst-case performance. In Arnold L. Rosenberg and Friedhelm Meyer auf der Heide, editors, *SPAA 2003: Proceedings of the Fifteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, June 7-9, 2003, San Diego, California, USA (part of FCRC 2003)*, pages 101–108. ACM, 2003. URL: `https://doi.org/10.1145/777412.777430`, `doi:10.1145/777412.777430`.
2. Sultan S. Alshamrani, Dariusz R. Kowalski, and Leszek Antoni Gasieniec. Efficient discovery of malicious symptoms in clouds via monitoring virtual machines. In Yulei Wu, Geyong Min, Nektarios Georgalas, Jia Hu, Luigi Atzori, Xiaolong Jin, Stephen A. Jarvis, Lei (Chris) Liu, and Ramón Agüero Calvo, editors, *15th IEEE International Conference on Computer and Information Technology, CIT 2015; 14th IEEE International Conference on Ubiquitous Computing and Communications, IUCC 2015; 13th IEEE International Conference on Dependable, Autonomic and Secure Computing, DASC 2015; 13th IEEE International Conference on Pervasive Intelligence and Computing, PICom 2015, Liverpool, United Kingdom, October 26-28, 2015*, pages 1703–1710. IEEE, 2015. URL: `https://doi.org/10.1109/CIT/IUCC/DASC/PICOM.2015.257`, `doi:10.1109/CIT/IUCC/DASC/PICOM.2015.257`.
3. Michael A. Bender, Rathish Das, Martin Farach-Colton, Rob Johnson, and William Kuszmaul. Flushing without cascades. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 650–669. SIAM, 2020. URL: `https://doi.org/10.1137/1.9781611975994.40`, `doi:10.1137/1.9781611975994.40`.
4. Michael A. Bender, Martin Farach-Colton, and William Kuszmaul. Achieving optimal backlog in multi-processor cup games. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 1148–1157. ACM, 2019. URL: `https://doi.org/10.1145/3313276.3316342`, `doi:10.1145/3313276.3316342`.
5. Marijke H. L. Bodlaender, Cor A. J. Hurkens, Vincent J. J. Kusters, Frank Staals, Gerhard J. Woeginger, and Hans Zantema. Cinderella versus the wicked stepmother. In Jos C. M. Baeten, Thomas Ball, and Frank S. de Boer, editors, *Theoretical Computer Science - 7th IFIP TC 1/WG 2.2 International Conference, TCS 2012, Amsterdam, The Netherlands, September 26-28, 2012. Proceedings*, volume 7604 of *Lecture Notes in Computer Science*, pages 57–71. Springer, 2012. URL: `https://doi.org/10.1007/978-3-642-33475-7_5`, `doi:10.1007/978-3-642-33475-7\_5`.
6. Gerth Stølting Brodal and Riko Jacob. Dynamic planar convex hull. In *43rd Symposium on Foundations of Computer Science (FOCS 2002), 16-19 November 2002, Vancouver, BC, Canada, Proceedings*, pages 617–626. IEEE Computer Society, 2002. URL: `https://doi.org/10.1109/SFCS.2002.1181985`, `doi:10.1109/SFCS.2002.1181985`.
7. Mee Yee Chan and Francis Y. L. Chin. General schedulers for the pinwheel problem based on double-integer reduction. *IEEE Trans. Computers*, 41(6):755–768, 1992. URL: `https://doi.org/10.1109/12.144627`, `doi:10.1109/12.144627`.
8. Mee Yee Chan and Francis Y. L. Chin. Schedulers for larger classes of pinwheel instances. *Algorithmica*, 9(5):425–462, 1993. URL: `https://doi.org/10.1007/BF01187034`, `doi:10.1007/BF01187034`.
9. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition.* MIT Press, 2009. URL: `http://mitpress.mit.edu/books/introduction-algorithms`.
10. Mattia D'Emidio, Gabriele Di Stefano, and Alfredo Navarra. Bamboo garden trimming problem: Priority schedulings. *Algorithms*, 12(4):74, 2019. URL: `https://doi.org/10.3390/a12040074`, `doi:10.3390/a12040074`.
11. Leszek Gasieniec, Ralf Klasing, Christos Levcopoulos, Andrzej Lingas, Jie Min, and Tomasz Radzik. Bamboo garden trimming problem (perpetual maintenance of machines with different

attendance urgency factors). In Bernhard Steffen, Christel Baier, Mark van den Brand, Johann Eder, Mike Hinchey, and Tiziana Margaria, editors, *SOFSEM 2017: Theory and Practice of Computer Science - 43rd International Conference on Current Trends in Theory and Practice of Computer Science, Limerick, Ireland, January 16-20, 2017, Proceedings*, volume 10139 of *Lecture Notes in Computer Science*, pages 229–240. Springer, 2017. URL: https://doi.org/10.1007/978-3-319-51963-0_18, doi:10.1007/978-3-319-51963-0_18.

12 Michael H. Goldwasser. A survey of buffer management policies for packet switches. *SIGACT News*, 41(1):100–128, 2010. URL: https://doi.org/10.1145/1753171.1753195, doi:10.1145/1753171.1753195.

13 R. Holte, A. Mok, L. Rosier, I. Tulchinsky, and D. Varvel. The pinwheel: a real-time scheduling problem. In *[1989] Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences. Volume II: Software Track*, volume 2, pages 693–702 vol.2, Jan 1989. doi:10.1109/HICSS.1989.48075.

14 Robert Holte, Louis E. Rosier, Igor Tulchinsky, and Donald A. Varvel. Pinwheel scheduling with two distinct numbers. *Theor. Comput. Sci.*, 100(1):105–135, 1992. URL: https://doi.org/10.1016/0304-3975(92)90365-M, doi:10.1016/0304-3975(92)90365-M.

15 William Kuszmaul. Achieving optimal backlog in the vanilla multi-processor cup game. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1558–1577. SIAM, 2020. URL: https://doi.org/10.1137/1.9781611975994.96, doi:10.1137/1.9781611975994.96.

16 Shun-Shii Lin and Kwei-Jay Lin. A pinwheel scheduler for three distinct numbers with a tight schedulability bound. *Algorithmica*, 19(4):411–426, 1997. URL: https://doi.org/10.1007/PL00009181, doi:10.1007/PL00009181.

17 Edward M. McCreight. Priority search trees. *SIAM J. Comput.*, 14(2):257–276, 1985. URL: https://doi.org/10.1137/0214021, doi:10.1137/0214021.

18 Mark H. Overmars and Jan van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Syst. Sci.*, 23(2):166–204, 1981. URL: https://doi.org/10.1016/0022-0000(81)90012-X, doi:10.1016/0022-0000(81)90012-X.

19 Theodore H. Romer and Louis E. Rosier. An algorithm reminiscent of euclidean-gcd computing a function related to pinwheel scheduling. *Algorithmica*, 17(1):1–10, 1997. URL: https://doi.org/10.1007/BF02523234, doi:10.1007/BF02523234.