

# A comparison of two GPU triangular mesh simplification algorithms in Vulkan on generated, convex terrain

William Vickers-Hastings, Falmouth University, Games Academy

**Abstract—So basically, mesh smaller**

**Index Terms**—Mesh Simplification, GPU Compute, Vulkan Compute, Compute Shader, Procedural Mesh, Terrain Generation

## I. INTRODUCTION

**M**esh simplification is an intensively studied field since the 1990s. Over the years, robust CPU based algorithms and in recent years GPU based algorithms have been freely available. There has been a growing interest in 3d procedural terrain generation in games with titles like No Man's Sky [1]–[7] especially promising a procedurally generated 3d universe to explore. Optimisation of the meshes that compromise them is challenging as perceived quality has a direct trade off to the number size of the meshes. The two intuitive ways to optimise such terrain meshes;

- Generate a lower resolution mesh. This is optimal both for generation time and rendering performance but has a major quality impact compared to a high-resolution generation, as a lot of detail can be lost through a low sampling rate. This downside can be algorithm dependent and can even be a good artistic constraint.
- Generate multiple terrains at various resolutions then use a level of detail system, to draw the high-resolution mesh as needed. This is often a workable trade off but the disadvantages of having to run the terrain generation algorithm multiple times and potentially extra overhead of slicing the terrain into tiles or chunks, which can be a limitation with some terrain generation algorithms, like hydraulic erosion [?].

In 3D modelling, mesh simplification (also known as decimation or optimisation) is a tool for reducing the geometry in a models such as for characters, while maintaining a high level of detail.

Applying such an algorithm to a high-resolution generated terrain mesh produces a higher quality result at the same level of geometry compared to running the terrain generation algorithm at an equivalent, lower input resolution.

This paper aims to compare these two GPU based algorithms to low resolution generation and determine which one is better for runtime convex terrain generation:

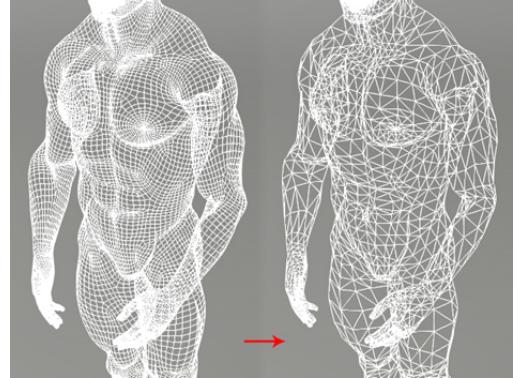


Fig. 1. Example mesh simplification of a character model from Character Creator 3 [?]

- Algorithm A, Embedded Tree Collapse, proposed by Hyunho Lee and Min-Ho Kyung in *Parallel mesh simplification using embedded tree collapsing* [8]
- Algorithm B, GPU Half Edge, proposed by Mohamed H. Mousa and Mohamed K. Hussein in *High-performance simplification of triangular surfaces using a GPU* [9]

## II. CONTEXTUAL ANALYSIS

### A. Mesh Simplification

To simplify a mesh is to maintain the shapes and details of the mesh with fewer vertices and triangles.

Since the early 1990s [10], [11] Many CPU based algorithms have been put forward for mesh simplification [10]–[20], with one of the most cited and widely used algorithms known as QSLIM [20] proposed in the 1997. Most algorithms developed since build upon QSLIM's edge contraction approach, widely known as half-edge contraction/collapse (see figure 2), but other novel approaches have been proposed since such as Multiple-choice Scheme [21].

Two reasons to do this, both revolving around size:

- **Data size** Smaller meshes take up less space on the disk and in the system memory and GPU frame buffer at runtime.
- **Rendering performance** The GPU doesn't need to do as much work to render a smaller meshes, allowing the application to render more meshes, or render more frames in a given unit of time.

The challenge with mesh simplification is, reducing geometry while maintaining a high level of quality, trying to do the

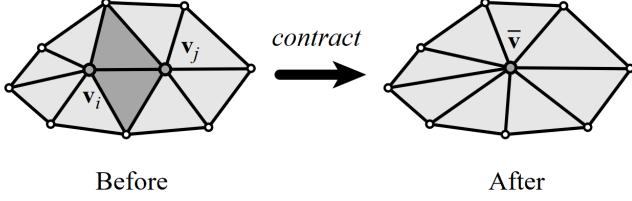


Fig. 2. Example of a single half edge contraction to simplify 10 triangles into 8, from figure 1 of QSlim [22]

same thing with less. A further challenge comes with doing it in real-time in a game like application. It can take seconds [22] to simplify a mesh depending on its variables, and while doing so incur a high memory cost.

After an extensive review, there is a lack of research on procedurally generated geometry or for real-time application in games. [13], [23]

Why there may be a lack of research on procedurally generated geometry could be because to reduce the geometry of a generated mesh is simple, so long as you can control the resolution of the generation. However, higher quality results should be achievable using simplification algorithms.

*1) Quantifying Results:* To quantitatively measure the quality of an algorithm, the geometric error of the simplified mesh compared to the original mesh is calculated using geometric deviation. Some papers [8] use other methods for measuring quality like Hausdorff distance [?] and/or Root Mean Squared [?].

Given two meshes, meshes A and mesh B, their faces, and a point that belongs to a face of mesh A, the geometric deviation for a point and face in mesh B is defined as the minimum euclidean distance between the point in mesh A and a point in mesh B, where the point in mesh B belongs to one of its faces. [?].

*2) GPU Algorithms:* As GPUs have gotten faster and more programmable since the late 2000s [?], many GPU implementations of mesh simplification algorithms have been proposed [8], [9], [14], [21], [24], [25], a paper from AMD in 2007 [24] was the first to take advantage of the high parallelised performance offered by GPUs on large data sets like multimillion polygon meshes and was also the first paper to put forward a real-time implementation of mesh simplification.

Since 2007 many other GPU implementations have been proposed [8], [9], [14], [21], [25], but not all have compared themselves to each other, from an exhaustive review, most GPU papers build upon the previous paper and compare to that previous implementation.

The current bleeding edge GPU algorithm was proposed by Mousa & Hussein in 2021 [9], they compared only compared their algorithm to QSlim [22] and Multiple-choice scheme [21]. However, they referenced a paper from 2016 by Lee & Kyung which proposed an algorithm using embedded tree collapsing [8], which they did not compare against.

Both Mousa & Hussein and Lee & Kyung papers, used

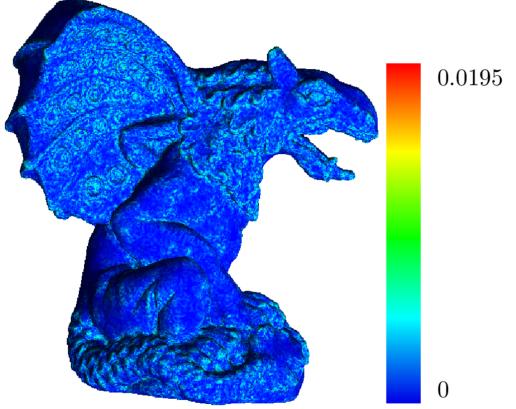


Fig. 3. The gargoyle model

a gargoyle (see figure 3), which appear the same, but have slightly different vertex and index counts. They did use directly comparable reduction rates. The results are visually similar, but as Lee and Kyung used MeshLab to measure Hausdorff distance [?] and Root Mean Square [?] to use as their quality metric, it is therefore not comparable to the Mousa & Hussein algorithm, as they used geometric deviation [?].

The execution time for algorithm proposed by Lee and Kyung, are orders of magnitude lower than the Mousa & Hussein algorithm. Lee and Kyung used a more powerful GPU, which likely explain this. Both algorithms were implemented using Nvidia's CUDA platform [26].

Given that Mousa & Hussein algorithm is more recent, it is expected that it will perform better in execution time, given the same hardware, and in quality.

This paper proposes to compare these two algorithms.

- Algorithm A, **Embedded Tree Collapse**, proposed by Hyunho Lee and Min-Ho Kyung *Parallel mesh simplification using embedded tree collapsing* [8]
- Algorithm B, **GPU Half Edge**, proposed by Mohamed H. Mousa and Mohamed K. Hussein *High-performance simplification of triangular surfaces using a GPU* [9]

Both CPU and GPU simplification algorithms are designed to work on arbitrary meshes usually pre-modelled in software like Blender [27], or Maya [28].

This paper will compare these two algorithms on simplex-based generated convex terrain geometry, as there is a lack of research on this. This paper will also compare the simplification algorithms to a low-resolution terrain geometry. This will be done using Vulkan [29] instead of CUDA [26] as the terrain generation artefact is written within a custom Vulkan application, and Vulkan native supports GPU Compute Shaders

### B. Embedded Tree Collapse

This algorithm's objective is to maximise GPU utilisation by breaking data dependency in the updates of the mesh. [9] does not do this, instead opting to use wait commands before moving on. The way this works, is by storing edge update information in a table and then updating the mesh data with it in the next step of the algorithm. This allows

a higher number of edges to be chosen from for contraction. All of this allows each step of the algorithm to reduce more of the mesh compared to [9]. Unlike [9] Embedded tree collapse has a quality parameter, which can be tailored to the mesh its simplifying to minimise quality loss at the cost of performance. [8] [8] suggest a high value of 0.1 for real-time applications.

The operations of Embedded Tree Collapse is broken down into 6 steps. [8]

- 1) Calculate quadric error matrix at every vertex. Quadric error is used to measure the detail cost to contract a vertex [22]
- 2) Construct a set of trees with collapsible edges. From the half-edge leaving each vertex, a tree edge must be selected without causing a cycle. Prevented with a weighting system and an error checking pass.
- 3) Split the trees to meet the quality threshold. In the previous step the whole tree may accumulate quadric error over the quality threshold, this step splits up trees that exceed it. [8]
- 4) Cancel tree edges causing illegal mesh changes. [8] An illegal change would be the flipping of a normal vector. Fixing this is hard, so triangles are found before hand and discarded from the tree
- 5) Collapse trees to their root vertices [8] this is down to prevent topological changes to the mesh
- 6) Refine root vertex positions, the root vertex positions are refined after collapse by minimising their accumulated quadric errors using a method proposed by [20]. Repeat 1-6 until it reaches the specified vertex count. [8]

### C. GPU Half Edge

GPU half edge works by splitting a mesh up into a set of regions, then simplifying each region separately in parallel and then combining their results [9].

The operations of GPU Half Edge algorithm is broken down into 5 distinct steps. [9]

- 1) Build the data structure, a customised half Edge data structure [?] to work around memory allocation limits in CUDA [9], [26]
- 2) Calculate the edge contraction costs, the quadric error metric, for each half edge. [9].
- 3) Build a priority queue using a skip list (a special list that has an average  $O(\log n)$  complexity for search and insertion [?]) of these costs to choose a pair of vertices to be contracted. [9]
- 4) Construct a set of disjoint parts of the mesh using k-d tree space partitioning. [9] The k-d tree is used to break the mesh down into chunks for parallelisation.
- 5) Apply the decimation process [9] During this process, one contraction per region of the k-d tree which guarantees the consistency of the neighbourhood properties among the vertices of the mesh [9] otherwise a region may rely on geometry another region has already optimised out. The data structures are then updated and the mesh is decimated again until it reaches the specified vertex count.

### D. Convex Terrain Generation

The type of terrain generation being used to assess the algorithms is similar to height-map generation but directly displaces vertices in their local up direction around a spherical like body resembling a planet. The resultant mesh is convex.

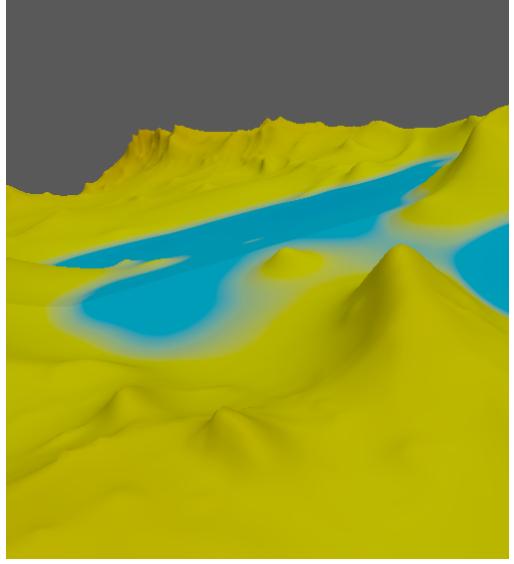


Fig. 4. Example terrain output

The terrain generator uses simplex-noise [30] in multiple layers with various settings based off of work by Sebastian Lague [?], combined with the gradient-trick for erosion proposed by inigo quilez [31]. This generator was created by the author in a custom Vulkan [29] framework in C# with SDL3 [?]

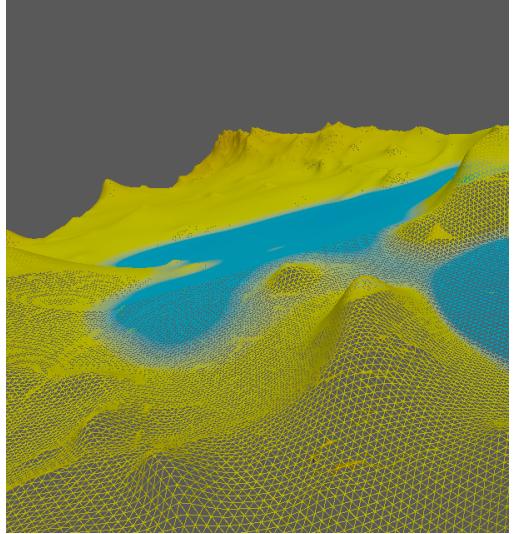


Fig. 5. The gargoyle model

## III. SCIENTIFIC QUESTIONS & HYPOTHESES

On a generated convex terrain mesh, do both mesh simplification algorithms produce a low geometric error result than a lower resolution terrain generation pass, and if so, which

simplification algorithm A or B produces the lowest geometric error for the least compute time.

- Algorithms A and B have a lower geometric error than a reduced resolution generated mesh
- Algorithm B run faster than algorithm A
- Algorithm B has a lower geometric error than algorithm A
- Algorithm B has a better execution time per geometric error ratio than algorithm A

As the purpose of this paper is to find which of the two algorithms is better, should hypotheses 2-4 be proven false, this information is still useful.

#### IV. METHODOLOGY

##### A. Computing Artefact

1) *Requirements:* To test the hypotheses a computing artefact that implements and can measure the performance of algorithm A and B must be created as well as this it must be able to generate a convex hull terrain mesh and multiple resolutions and with repeatable but randomizable results.

An existing terrain generating artefact will be used, created by the author. The data recording and algorithm implementation will be added into framework.<sup>1</sup> The variables being measured about each algorithm are the execution time of the algorithm and the quality of the result, expressed as geometric error.

As this experiment is to determine which algorithm is better with generated terrain meshes, multiple terrain meshes will be required to verify the null hypotheses is false. The existing framework can do this by randomly generating a seed which terrain generation is based off, the seed used to generate the mesh should be recorded in-case of a validation needing to be performed.

To simplify the mesh using the algorithms, they need an additional input, how much to simplify the high-resolution mesh by. This is given as a % of the original Triangle count, to maximise the resolution of the test. Each algorithm will be tested using a set of reduction levels:

Reduction of original geometry
75%
50%
25%

To compare the algorithms to a lower resolution terrain, each terrain will also be generated at a lower resolution with roughly equal to the reduction levels used by the algorithms. Data will be recorded as a csv file.

Finally, algorithm A takes an additional input variable, a quality threshold. The lower this value is, the high quality the output mesh will be at the cost of performance. Some testing will be needed to determine a good value for this variable on a range of meshes. Alternately, the variable could be used to normalize the quality between the two algorithms and compare only the execution time.

2) *Breakdown of Operations:* The artefact will test the algorithms through by first generate a high-resolution terrain mesh with N triangles.

The initial resolution will be a broadly configurable variable. For each algorithm and for each reduction level, the artefact will run the algorithm using the high-resolution mesh as its input to create a simplified mesh, measuring how long it takes the algorithm to create the simplified mesh.

After creating the simplified mesh, the artefact will then compare the simplified result to the high-resolution mesh, by calculating a geometric error score, with 100% meaning identical in shape to the high-resolution mesh.

Finally, the artefact will create a csv entry, recording the terrain generation seed, the starting triangle count, the reduction level, the execution time and the geometric error.

3) *Development:* In order to deliver this artefact, agile [?] development practices will be used to rapidly iterate on the application. Tasks will be kept track off using GitHub's task-board [?]. As its iterated, the artefact will be tested using unit tests and refactored based on static code analysis [?] To ensure Algorithm A and B are providing outputs consistent with their author's expectations, they will be tested using the same models each paper used as they are widely available.

##### B. Ethical Considerations

The ethical concerns for this project end at the usage of the algorithms being tested within the paper from an IP perspective and the IP of the libraries, frameworks and tools used to carry out the experiment and analysis.

It is acknowledged that this is a computing experiment and so consideration for sustainability (energy usage) has been made, and deemed low as the proposed experiment can be run on a desktop computer, a high-power data centre is not necessary.

The overall ethical risk is low as this project does not involve human participants, or any significant legal issues. This research cannot be weaponised.

##### C. Data Management and Analysis Plan

###### 1) Sampling: Control tests

Each algorithm will be test on a control generated mesh to determine its run-to-run variance in geometric error and execution. If run to run error is high for one or both algorithms, then during the experiment, each step can be repeated and averaged together as many times as needed to be confident in the results. Additionally, Algorithm A, Embedded Tree Collapse, will be tested against itself to find an optimal value for its quality threshold. This optimal value when then be used in its tests against algorithm B. Algorithm B does not need a test like this as it has no quality parameters.

###### Sample Size

From analysis of [8], [9] execution time, cohen's d effect size of 9 was calculated, this puts sample size total for both algorithms at 4 for a t-test. This is far too low and speaks to the major hardware differences between the two papers. To compensate, initially cohen's d will be capped at 2 and beta

<sup>1</sup>Link to repository

error will be raised from 0.95 to 0.99. This makes total sample size 22, 11 for each algorithm. After data collection, it may be necessary to raise further as the hardware will be the same, or the large effect will support the hypotheses.

Due to the difference in geometric error measurement, the data sets between the papers are not comparable, making calculation of sample size less accurate. Comparing within the papers by looking at the algorithms they compared against, a large Cohen's d is produced. This suggests that to each other, Cohen's d will be lower, for now a value of 1 will be used, for a sample size of 76(38 per algorithm).

More data can be easily collected in the case Cohen's d is lower than suggested here based on initial analysis due to the artefacts random generation system.

## Experiment

To acquire data to answer each hypothesis, the artefact will start by generating a high-resolution terrain mesh at a specific polygon count (100k triangles). The seed to produce this mesh will be recorded. Each algorithm will then simplify the high-resolution mesh to a range of steps 75%, 50% and 25% of the original polygon count.

The same will be done by the generation algorithm, making it regenerated the mesh at 75%, 50% and 25% of the original polygon count.

Time milliseconds it takes to simplify from the original to the reduction step will be recorded.

After a simplified mesh has been created, the result will be compared to the original to determine the geometric error % of the simplified relative to the original mesh, using Geometric Deviation.

The geometric error of each algorithm will also be compared to running the generation algorithm at a lower resolution.

2) *Data Analysis:* Each hypothesis will be tested using a T test.

```
library(readr)
dat <- read_csv("H: Stats\tests_data.csv")
view(dat)
summary(dat)
t.test(x~y)
```

x and y will refer to the geometric deviation or to the execution time.

To prove the null hypotheses false for each hypotheses, both algorithms will have to show a quality change to a lower resolution terrain generation - either positive or negative, one of the algorithms should prove to be faster to execute than the other and one of the algorithms should have a low geometric deviation than the other.

## REFERENCES

- [1] Hello Games, "About - no man's sky."
- [2] Ludeon Studios, "RimWorld - sci-fi colony sim."
- [3] Chucklefish, "About | starbound."
- [4] Chucklefish, "Starbound."
- [5] Iron Gate Studio, "Valheim."
- [6] S. Parkin, "The video game that maps the galaxy," Section: annals of technology.
- [7] Mojang, "What is minecraft? build, discover realms & more."
- [8] H. Lee and M.-H. Kyung, "Parallel mesh simplification using embedded tree collapsing," vol. 32, no. 6, pp. 967–976. Number: 6-8 Num Pages: 967-976 Place: Heidelberg, Netherlands Publisher: Springer Nature B.V.
- [9] M. H. Mousa and M. K. Hussein, "High-performance simplification of triangular surfaces using a GPU," vol. 16, no. 8, p. e0255832. Number: 8.
- [10] W. J. Schroeder, J. A. Zarge, and W. E. Lorensen, "Decimation of triangle meshes," vol. 26, no. 2, pp. 65–70.
- [11] M. J. DeHaemer and M. J. Zyda, "Simplification of objects rendered by polygonal approximations," vol. 15, no. 2, pp. 175–184.
- [12] J.-L. Peyrot, F. Payan, and M. Antonini, "Aliasing-free simplification of surface meshes," in *2014 IEEE International Conference on Image Processing (ICIP)*, pp. 4677–4681. ISSN: 2381-8549.
- [13] P. K. Agarwal and P. K. Desikan, "An efficient algorithm for terrain simplification," in *Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, SODA '97, pp. 139–147, Society for Industrial and Applied Mathematics.
- [14] S. M. Shontz and D. M. Nistor, "CPU-GPU algorithms for triangular surface mesh simplification," in *Proceedings of the 21st International Meshing Roundtable* (X. Jiao and J.-C. Weill, eds.), pp. 475–492, Springer.
- [15] Y. Zhang, H. Wang, H. Zhou, and J. Li, "Finite triangular surface mesh simplification with geometrical feature recognition," vol. 223, pp. 2627–2636. Number: C11 Num Pages: 10 Place: London, United States Publisher: SAGE PUBLICATIONS, INC.
- [16] W. Jakob, M. Tarini, D. Panozzo, and O. Sorkine-Hornung, "Instant field-aligned meshes," vol. 34, no. 6, pp. 1–15. Number: 6.
- [17] Martin Franc, "Methods for polygonal mesh simplification."
- [18] Y. P. Wang, J. W. Dang, J. Y. Yang, and S. Wang, "Research on triangular mesh simplification algorithm of virtual object model," vol. 271–272, p. 1410. ISBN: 9783037855782 Num Pages: 1410 Place: Zurich, Switzerland Publisher: Trans Tech Publications Ltd.
- [19] D. Salinas, F. Lafarge, and P. Alliez, "Structure-aware mesh decimation," vol. 34, no. 6, pp. 211–227. \_eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12531>.
- [20] M. Garland and P. S. Heckbert, "Surface simplification using quadric error metrics," in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '97, pp. 209–216, ACM Press/Addison-Wesley Publishing Co.
- [21] N. Koh, W. Zhang, J. Zheng, and Y. Cai, "GPU-based multiple-choice scheme for mesh simplification," in *Proceedings of Computer Graphics International 2018*, pp. 195–200, ACM.
- [22] P. S. Heckbert and M. Garland, "Optimal triangulation and quadric-based surface simplification," vol. 14, no. 1, pp. 49–65. Number: 1.
- [23] Stiaan Uyttersprot, "Mesh compression and procedural content generation: Finding common ground for an improved 3d representation."
- [24] C. DeCoro and N. Tatarchuk, "Real-time mesh simplification using the GPU," in *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pp. 161–166, ACM.
- [25] A. Papageorgiou and N. Platis, "Triangular mesh simplification on the GPU," vol. 31, no. 2, pp. 235–244. Number: 2 Num Pages: 235-244 Place: Heidelberg, Netherlands Publisher: Springer Nature B.V.
- [26] Nvidia, "CUDA toolkit - free tools and training."
- [27] Blender Foundation, "blender.org - home of the blender project - free and open 3d creation software."
- [28] Autodesk, "Autodesk maya | get prices & buy official maya 2025 | autodesk."
- [29] Vulkan, "Home | vulkan | cross platform 3d graphics."
- [30] K. Perlin, "Chapter 2: Noise hardware,"
- [31] I. Quilez, "Inigo quilez."

## APPENDIX A

Example data for a single test

Seed	Algorithm	Triangle Count	Reduction Level	Runtime	Geometric Error
1349123	Terrain Gen	15m	75%	300ms	50%
1349123	Terrain Gen	15m	50%	30ms	25%
1349123	Terrain Gen	15m	25%	3ms	10%
1349123	Algorithm A	15m	75%	600ms	75%
1349123	Algorithm A	15m	50%	60ms	50%
1349123	Algorithm A	15m	25%	6ms	25%
1349123	Algorithm B	15m	75%	1000ms	95%
1349123	Algorithm B	15m	50%	100ms	75%
1349123	Algorithm B	15m	25%	10ms	50%