Education and out of the box thinking – linearization of Graham’s scan algorithm complexity as fruit of education policy

Veljko B. Petrović, Dragan Ivetić
Faculty of Technical Sciences, University of Novi Sad, Serbia
pveljko@uns.ac.rs, ivetic@uns.ac.rs

ABSTRACT
The question of educational policy is considered with reference to its effect on student creativity. An example of this effect is presented – the linearization of Graham’s scan algorithm. The linearization is described in detail with reference to how varying aspects of it originate with the original educational environment. The Graham’s Scan approach to two-dimensional convex hull calculation is considered. The performance bottleneck is found in the sorting step that precedes the Graham’s Scan scanning operation. Methods are considered to eliminate this bottleneck. The method chosen is replacing the \(O(n \log n)\) sorting algorithm normally used with a radix sort. To operate within Graham’s scan, the radix sort algorithm must be modified. The main modification is getting it to operate on real numbers. This is achieved by using the fact that digital computers only operate on a countable and finite subset of real numbers, and using this fact to reduce the problem of sorting by real number to sorting by integer. The ramifications of this modification are taken into consideration in the light of previous theoretical work in this area. Performance as compared to other algorithms is considered. Further, the consequences of a proof that the lower bound for the temporal complexity of two-dimensional convex hull algorithms is \(\Omega(n \log n)\) are considered.

Keywords: education, convex hull, Graham’s scan, algorithms, complexity

1 INTRODUCTION
The convex hull problem in two-dimensional space can be summarized as the search for a convex polygon that contains all the points in a certain set \(Q\) while being minimal in size. This is, to an extent, a simplification. In the general case finding the convex hull of a set of points \(Q\) in \(\mathbb{R}^n\) is the search for a boundary of the least convex set which contains all those points. In the two-dimensional case this boundary is clearly a polygon which brings us to the initial definition [1].

The convex hull is an important concept in computer graphics and geometric modeling university courses. Convex combinations are the basis for a great number of crucial algorithms. The solution to the convex hull problem is also an excellent example of a computer graphics algorithm. It is simple enough that the problem can be understood by anyone, but complex enough for the solution to pose an actual challenge. Another positive aspect of convex hull algorithms is that they demonstrate quite clearly the nature of global approaches in algorithmic thinking, such as divide and conquer.

It is not a question of “if” the convex hull problem should be studied. It is just a matter of how it should be studied. The way a topic is presented to students can impact how they learn, but also how they view this topic in the future. Our teaching process focuses on the following aspects: approach the topic as a problem, independent work, out of the box thinking and synergy between disparate fields of study. Approaching the topic as a problem means that instead of focusing on one solution, the topic is presented as a problem with many possible solutions. Independent work is another key facet of the educational policy employed. The structured environment of the classroom is not always the best place to foster inquiry and independent thought. This is why a great deal of our coursework is accomplished through independently pursued projects. The result presented in this paper was the result of preparatory research for a master’s thesis performed during work on the GIMMobile project.

Out of the box thinking is not a skill that can be taught reliably. However, it can be encouraged. The educational policy that was found to be best for this is an openly questioning approach to all subject matter. Another facet of out of the box thinking is the exploration of synergies between disparate fields of study, in itself another cornerstone of computer science education at the Faculty of Technical Sciences. These synergies can be found between...
wide ranging fields, but in the case of the result reported here, were mostly focused on the obvious connection between complexity theory, algorithm design and computer graphics. However, less expectedly, knowledge of computer architecture was also crucial to formulating a solution.

The educational policy so described led to the discovery of a new modification to an existing algorithm for convex hull calculation by students. The rest of the paper describes the modification in detail and illustrates how certain aspects of this modification depend on the education policy.

An example of an algorithm capable of finding the convex hull of a set of points in two dimensions is Graham’s Scan. Graham’s scan is a simple rotational sweep algorithm with good performance characteristics. It is \(O(n \lg n)\) in the worst case. The algorithm exists in multiple variants, but the original works in three distinct phases: \([2]\) \([3]\)

1. Preparing the input point set.
2. Computing the initial hull.
3. Sweeping around the points removing ones which would not fit into the hull.

The input set is prepared by first picking a pivot point for the algorithm. This pivot point is usually the lowest, leftmost point in the set and is, as an extreme, always a part of the convex hull \([4]\). All other points are sorted by their polar angle around the pivot point. Those points which had identical polar angles are eliminated in such a manner that the only point with a given angle remaining is the one farthest from the pivot point \([2]\).

The initial hull is computed by simply starting with the pivot point and the first two points in the sorted input points. This initial hull becomes the current hull which is updated throughout the algorithm. The sweeping phase considers points in the sorted input set one by one. For each of those points it is determined if its addition to the current hull causes a non-left turn. If it does, the latest point in the current hull is removed and the direction of the turn is tested again. This continues until a left turn is obtained, at which point the considered point is added to the current hull \([2]\).

Listing 1 contains a pseudocode representation of the unmodified algorithm. Several external operations are used in this algorithm. They include:

- \(\text{init}(S)\), \(\text{push}(S, E)\), \(\text{pop}(S)\) and \(\text{length}(S)\) are standard stack and/or list operations. The \(\text{init}(S)\) operation initializes the stack to its correct initial state. The \(\text{push}(S, E)\) operation places the element \(E\) at the top of the stack \(S\), while moving all other elements of \(S\) one step down. Conversely, \(\text{pop}(S)\) removes the top element of \(S\) and returns it, while moving all other elements of \(S\) one step upwards. Finally, \(\text{length}(S)\) returns the number of elements in \(S\).
- \(\text{lowest}(Q)\) returns the point in \(Q\) whose \(y\) coordinate is minimal. In the case that there are many such points it returns the point from this subset which is such that its \(x\) coordinate is minimal. In short it returns the leftmost lowermost point.
- \(\text{eliminate}(Q, p_0)\) takes a list \(Q\), sorted by the polar angle around \(p_0\) and returns a list where there are no points with the same angle. In case of conflict, only the point farthest away from \(p_0\) is kept. This can be made an \(O(n)\) operation. The way to do this is to iterate trough \(Q\). In each iteration the point from \(Q\) is copied into a new set provided the polar angle of that point is different from its predecessor. If the angle is the same as its predecessor it is copied in the position of the predecessor if and only if its distance from the polar point is greater than the distance of the point already copied into the output set. It should be noted that this only works if the input set is sorted beforehand.
- \(\text{nonleft}(a, b, c)\) determines if the angle formed by the three indicated points turns to the left or not. Easily determined by calculating the dot product of the relevant vectors.

It has previously been stated that Graham’s scan has \(O(n \lg n)\) complexity. It is possible, given the pseudocode representation, to see why. Lines 1, 5, 6, 7 and 8 are \(O(1)\), and need not be further discussed. The calculation of the lowest, leftmost point in line 2 is a \(\Theta(n)\) operation. This can be safely claimed because it must, by necessity, iterate trough all of the points in the input set. Line 3 is a simple heap sort and has the complexity class of \(O(n \lg n)\). Line 4 has the complexity of \(O(n)\).

The two nested loops give the impression of
great complexity but, in fact, pose little complication. The outer loop can execute, at the most \( n - 3 \) times. Discounting the inner loop, each execution of the outer loop is an \( O(1) \) push operation. The inner loop is less predictable. However, given that it contains only a pop operation, and that it is impossible to pop something from the stack which is not there in the first place, it is clear that the while loop can only execute, in toto, \( m - 2 \) times.

Since the nonleft test is \( O(1) \), and so is the pop operation, and since \( m - 2 \leq n - 1 \) the total complexity of the while loop is \( O(n) \). Thus, the total complexity of the algorithm is \( O(1) + O(n) + O(1) + O(1) + O(1) = O(n) \), \( [2] \).

It is interesting to note that the complexity of the algorithm does not depend on the part of the algorithm that does the calculation of the hull itself, but instead on the sorting step which is clearly the dominant member, complexity-wise. This poses the question if it is possible to decrease the complexity by trying for a more efficient sort. It is commonly known that there exists a lower bound of \( \Omega(n \lg n) \) for any sorting algorithms based on arbitrary key comparisons \([5]\). However, there exist more specialized algorithms that exhibit better performance in certain cases.

This paper is divided into four separate sections. The first is the introduction, which introduces the concepts used in the paper, chiefly, Graham’s Scan algorithm and the educational policy that lead to its modification. The second section outlines radix sort and how it may be modified and incorporated into Graham’s Scan. The third section deals with the implication of this modification and the linearization of Graham’s Scan temporal complexity. It deals with the existing proof of a lower bound for two-dimensional convex hull calculation, and compares the modified Graham’s Scan to other algorithms. The fourth section contains the results of performance testing of the algorithm modification on real hardware. The fifth section concludes the paper, briefly describes what has been accomplished and outlines potential avenues for further research.

2 RADIX SORT MODIFICATION

The core concept of this paper is to adapt the radix sort algorithm to Graham’s scan in order to reduce the complexity class of the resulting algorithm to \( O(kn) \). Radix sort is a variant of non-comparison based sorting generally meant for integers. Briefly, it operates by sorting a given list of integers by sorting them sequentially by their digits in a certain radix. The specific version used here starts with the least significant digit (LSD) and uses iterated counting sort applied to bytes. This means that it will sort a, say, four-byte value in four passes. The total performance of a radix sort is \( O(kn) \) \([5]\).

**RADIX-SORT(L):**

```
01. len := length(L)
02. for i := 0 to len-1 do
03.     input[i] := raw(L[i])
04.     mIndices[i] := i
05.     mIndices2[i] := i
06.     h0 := 0; h1 := 256; h2 := 512; h3 := 768;
07.     h4 := 1024; h5 := 1280;
08.     h6 := 1536; h7 := 1792
09. for i := 0 to len-1 do
10.     counters[h0 + getbyte(input[i], 0)]++
11.     counters[h1 + getbyte(input[i], 1)]++
12.     counters[h2 + getbyte(input[i], 2)]++
13.     counters[h3 + getbyte(input[i], 3)]++
14.     counters[h4 + getbyte(input[i], 4)]++
15.     counters[h5 + getbyte(input[i], 5)]++
16.     counters[h6 + getbyte(input[i], 6)]++
17.     counters[h7 + getbyte(input[i], 7)]++
18. for pass := 0 to 7 do
19.     offsetTable[0] := 0
20.     for i := 1 to 255 do
21.         offsetTable[i] := offsetTable[i - 1]
22.     for i := 0 to len – 1 do
23.         id := mIndices[i]
24.         byt := getbyte(input[id], pass)
25.         mIndices2[offsetTable[byt]] := id
26.         offsetTable[byt]++
27.     tmp := mIndices
28.     mIndices := mIndices2
29.     mIndices2 := tmp
30. return mIndices
```

**Listing 2:** Modified and adapted radix sort

As described, radix sort will only work with integers. Further, it will only work on integers that
can be expressed in a certain, fixed, number of radices, though this number can be arbitrarily large. To be used in Graham’s scan, radix sort will need to sort by polar angle. This requires sorting points by real keys, which is quite different than sorting a single list of integers. The problems which need to be solved are:

- Sorting real numbers with radix sort.
- Sorting points by key, and not only the keys themselves.
- Maintaining the performance gain.
- Assuring unchanged precision.

It should be stated immediately that radix sort cannot sort true real numbers. However, since no digital computer works with true real numbers, but an approximation thereof, this should not pose any problem. The numbers that radix sort needs to sort in this instance are limited to the range $[0, 2\pi]$. Any implemented approximation used on an actual computer will have limited precision. Given the limited precision, and the limited range of possible values it is clear that it is possible to construct an $O(1)$ function such that equation 1 holds.

$$f : [0, 2\pi] \cap RF \rightarrow N_0$$

$$a \leq b \iff f(a) \leq f(b) \wedge a, b \in [0, 2\pi] \cap RF$$

(1)

In Eq. (1) RF is the set of real numbers which can be represented in the system of approximation in question. Given that a fixed-size system of representation can only represent a fixed number of distinct numbers, it is possible to construct $f$ by ordering all possible real representations using the index of a given input parameter in the resulting list as the result of $f$. With this in mind, it is clear that radix sort can be used to sort by polar angle, provided an appropriate $f$ is used.

Most modern computers represent real numbers using floating point, specifically, the IEEE 754 standard. The implementation used in this paper is based on the double precision IEEE 754 standard as seen on Figure 1. Double precision floating point values can be treated as integers for the purposes of sorting. Radix sort is normally an in-place sort stable sort [5], but for the purposes of this paper it was necessary to maintain association between the sorted polar angle values and the points they referred to. There is no immediately convenient way of doing so. It is possible to use a hashtable, but this would lead to performance issues and greater memory consumption. Thus, the radix sort used was modified to create a permutation which, when applied to the input list of points would create the sorted version, as can be seen in listing 2. The input to the RADIX-SORT algorithm is a list of floats which corresponds to the calculated polar angles of the input list of points.

**GRAHAM-SCAN-RADIX(Q):**

01. if(length(Q) <= 3) return Q
02. p0 := lowest(Q)
03. for i := 0 to len − 1 do
04.  slist[i] := getPolarAngle(p0, Q[i])
05.  perm := radix-sort(slist)
06.  (p1...pm) := eliminate(Q, perm, p0)
07.  init(S)
08.  push(p0, S)
09.  push(p1, S)
10.  push(p2, S)
11.  for i := 3 to m do
12.    while nonleft(peek(S),top(S),pi) do
13.      pop(S)
14.      push(pi, S)
15. return S

**Listing 3:** Graham’s Scan modified to include RADIX-SORT

The precision of this approach is not in question, as long as computers unable to handle true reals are used. The imprecision of the described approach is exactly equal to the imprecision of the method of depicting real numbers in fixed space. As a result of this, the precision of the modified algorithm is no better, and no worse, than any other comparable algorithm. RADIX-SORT uses several external operations. The raw(x) operation turns a double precision IEEE float into an integer based on their binary representation. This does not require any substantial operations, and does not influence performance. The getbyte(x, i) operation extracts the i-th byte from x. This can be accomplished with a couple of bitwise operations.

Graham’s scan is relatively easy to adapt to include RADIX-SORT. The modification is localised to lines 3-6 of listing 3. First, slist is created as a
separate list of polar angles around p0. To do this, the \texttt{getPolarAngle(p1, p2)} external operation is employed. Slist is then sorted creating a permutation perm. The external operation \texttt{eliminate(L, p, p0)} works as before, but generates a new list and, instead of expecting a sorted L, it uses the permutation p to sort L.

As far as performance is concerned, the only substantive difference between this implementation and the original is in the complexity of the sorting step which is $O(kn)$. The copying of the polar angle is an $O(n)$ operation. Thus the temporal complexity of the entire algorithm is $O(1) + O(n) + O(1) + O(n) + O(n) + O(1) + O(n) + O(n) = O(kn)$.

It should be clear that this modification could never have been invented by students had they not been encouraged to approach the problem unconventionally. Of particular importance is how the design of the modification is influenced by the synergies between computer graphics and basic algorithm theory and even more by realizations gleaned from a study of computer architecture.

### 3 IMPLICATIONS AND COMPARISONS

There exists a proof which states that the lower bound for convex hull algorithms in two dimensions is $\Omega(n \lg n)$. Given \( n \) real numbers \( x_1...n \) a set \( P \) is constructed as in Eq. (3).

\[
P = \{ p_i | 1 \leq i \leq n \} \\
p_i = (x_i, x_i^2) \tag{3}
\]

Then, the convex hull of \( P \) is computed. The order in which the points \( p_1...n \) appear on the lower half-hull of \( \text{conv}P \) is the order in which \( x_1...n \) should be sorted. Thus, if the convex hull can be computed in \( o(n \lg n) \) time, points can be sorted in \( o(n \lg n) \) time \cite{7}. This conflicts with the known lower bound for general sorts \cite{5}. Despite appearances, the described modification to Graham’s Scan does not conflict with this proof. The proof rests on the theoretical framework of algebraic trees and assumes the coordinates of the points are actual real values. The modified Graham’s scan does not work on actual real values, and as such the proof does not apply.

It is of some considerable interest to compare this modification against other algorithms for two-dimensional convex hulls. Table 1 shows the names of the more common algorithms and their expected performance characteristics in an average case and in the worst-possible case.

**Table 1: Performances of Various Algorithms for Calculating Two-Dimensional Convex Hulls**

<table>
<thead>
<tr>
<th>Name</th>
<th>Complexity, average case</th>
<th>Complexity, worst case</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gift Wrapping Algorithm</td>
<td>$O(nh)$</td>
<td>$O(nh)$</td>
<td>[2][4][8]</td>
</tr>
<tr>
<td>Graham’s Scan</td>
<td>$O(n \lg n)$</td>
<td>$O(n \lg n)$</td>
<td>[2][3]</td>
</tr>
<tr>
<td>Quickhull</td>
<td>$O(n \lg n)$</td>
<td>$O(n')$</td>
<td>[9]</td>
</tr>
<tr>
<td>Incremental with Edelsbrunner modification</td>
<td>$O(n \lg n)$</td>
<td>$O(n \lg n)$</td>
<td>[10]</td>
</tr>
<tr>
<td>Preparata – Hong</td>
<td>$O(n \lg n)$</td>
<td>$O(n \lg n)$</td>
<td>[11]</td>
</tr>
<tr>
<td>Chan’s Algorithm</td>
<td>$O(n \lg h)$</td>
<td>$O(n \lg h)$</td>
<td>[12]</td>
</tr>
<tr>
<td>Graham’s Scan, modified</td>
<td>$O(kn)$</td>
<td>$O(kn)$</td>
<td>/</td>
</tr>
</tbody>
</table>

Some convex hull algorithms belong to a class of algorithms known as \textit{output-sensitive}. That means that they express their complexity as a function of not only \( n \), but also the size of the output set – \( h \). To compare algorithms easily, it is necessary to estimate a value for \( h \). This is a non-trivial problem of stochastic geometry, but there exist certain solutions in the literature as seen in Table II.

For purposes of easy and intuitive comparison, for an average case the circular/square uniform distribution used which means that \( h = n^{1/3} \). In the worst case, the input set of points is on the border of a circle, which means that \( h = n \). The only remaining parameter is a value for \( k \). An illustrative estimate for \( k \) is 3. Of course this is only good for simple comparisons.

**Table 2: Stochastic Estimations for Hull Size**

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Estimate</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circular uniform</td>
<td>$n^{1/3}$</td>
<td>[3][4]</td>
</tr>
<tr>
<td>Square uniform</td>
<td>$n^{1/3}$</td>
<td>[4]</td>
</tr>
<tr>
<td>Normal planar distribution</td>
<td>$(\lg n)^{1/3}$</td>
<td>[3]</td>
</tr>
<tr>
<td>Uniform within convex polygon</td>
<td>$\lg n$</td>
<td>[3]</td>
</tr>
</tbody>
</table>

The estimate of \( k \) used is chosen to illustrate the behavior of the complexity as \( n \) increases. A true value for \( k \) is best determined via experiment.
Figure 2: Average case complexity graph

Figure 3: Worse case complexity graph
these algorithms can be seen on Figure 2 and Figure 3. Figure 2 is the comparison between algorithms in an average case, and figure 3 is the comparison between algorithms in the worst possible case. As can easily be seen, the modified Graham’s scan is the fastest algorithm in the long run. This is to be expected as a constant multiplier, no matter how large, can always be surpassed by a function of n, as n increases.

However, the weakness of modified Graham’s scan is that it takes a relatively large input set before its superiority sets in. How practical this is, can only be determined by experimenting.

Determining accurate performance characteristics is another example of how study in one area naturally leads to study in other, seemingly unrelated areas. Research in this direction encouraged students to acquire a deep understanding of performance on a theoretical as well as practical level.

4 RESULTS AND PERFORMANCE

The question raised in section 3 remains: how practical is the modified algorithm? The theoretical performance is demonstrably greater, but the overhead of the radix sort, not to mention the memory operations may serve to make the value of n at which the performance gain is noticeable unacceptably large.

The only way to determine the value of n is to experiment. To this end, a series of tests was conducted with the aim to compare the unmodified Graham’s Scan with the version employing modified radix sort. Both algorithms were implemented naively, to avoid differences caused by differing degrees of optimization. The only optimization performed was on the unmodified Graham’s Scan where the results of polar angle calculations were cached. This was considered acceptable because the
equivalent of the caching is part of the core modified algorithm.

Tests were performed with varying input set sizes, which range from 100000 points to 700000 points, increasing by 100000 each step. The test sets are randomly generated. All tests were performed ten times, and the results averaged together in an attempt to eliminate the influence of random events on the computer used for testing. Another element of tests is to see if there is variation of performance depending on the way the input sets were generated: in a circle, on a circle or in a square. As a matter of convention the circles used have a radius of 128 units and the square is 128 by 128.

The first test measures performance in the case of points uniformly distributed in a circle. The results can be seen on Figure 4. First, it is clear that the modified algorithm has linear performance, to within the accuracy of the measurements. Second, it is clear that the original algorithm does not yet deal with values of n large enough so that it is clear that it is not linear from visual data. However, it can be noticed that the general trend of the difference between the two is to increase – meaning that the modified algorithm has a lower complexity class than the unmodified one.

An unexpected element of the data is that the difference between the algorithms is visible much sooner than the theoretical estimate would suggest. This is largely due to the fact that the unmodified algorithm deals with floating point data, while the modified one does not to such a great extent. Further those elements of floating point data that the modified algorithm does deal with, it caches. Caching is of great importance, as its introduction to the unmodified algorithm served to increase performance six fold, starting, as specified, from a naïve implementation.

Figure 5 demonstrates the behavior of the algorithms in the event of the points being distributed on the boundary of a circle. The only major differences are that the difference between O(n lg n) is more pronounced than before.

Figure 6 confirms the expected. Baring minor variations caused by the measuring process, the behavior of the algorithm is consistent across all considered categories. It can then be concluded that the linearization of performance has been achieved and that the algorithm is practical for use.

5 CONCLUSION

This paper has outlined how the temporal complexity of Graham’s Scan can be linearized provided it operates on a finite, countable subset of reals that can be represented on some digital computer. It provides the framework to create such an algorithm independently of the system a given computer uses to represent real numbers.

This principle is illustrated on the example of the floating point representation of reals, specifically one described in IEEE’s 754 standard. A concrete implementation of the idea, in pseudocode, allows for discussion of implementation detail and a more nuanced analysis of expected performance.

The paper also demonstrated the potential of educational policy to spark innovation. Adherence to a policy resembling the one described cannot guarantee innovation, but it can certainly help it develop should it appear.

Further possible avenues of research include an analysis of potential applications and an experimental comparison between this algorithm and reference implementations of already well known algorithms.

In the context of education, the chief unexplored avenue of research is to determine if this approach helps with other fields of computer science education. Another interesting question is if an approach such as this could be devised for other areas of study, perhaps focusing strongly on experiment.
ACKNOWLEDGEMENT

This work is financially supported by the Ministry of Science and Technological Development, Republic of Serbia; under the project number III47003. “Infrastructure for e-learning in Serbia”, 2011-2014.

6 REFERENCES