



Department of Computer Science

CPCS223, Fall Term 2020

Empirical Analysis of Convex hull Brute Force & divide-and-Conquer algorithms of set of points in 2-dimensional

Analysis & design algorithm CPCS-223

Instructor: Bassma Alsulami

December 2020

By: Razan Muhammed Aljuhani

Table of contents

1.Introduction	3
2. Empirical Analysis of Algorithms Time efficiency	4
2.1 The Experiment purpose	4
2.2 The Efficiency Matrix	4
2.3 Input Characteristics.....	5
2.4 Algorithms.....	5
2.4.1. The brute force Algorithm.....	5
2.4.2. The QuickHull Algorithm.....	7
2.5 Generate a sample of inputs	8
2.6 Data Observed from Running the Algorithm.....	8
2.7 Analyse of the data obtained	10
2.7.1 Study Discussion of results.....	10
2.7.2 Comparison to Theory	11
3. Conculosion.....	12
4. Appendix	12
5. References	13
 Figure 1: Brute Force Algorithm and its graphical illustration.....	5
Figure 2: Quick Hull Algorithm and its graphical illustration	7
Figure 3: The Brute Force graph	9
Figure 4: The QuickHull graph.....	9
Figure 5: Average for 2 algorithms graph	10
 Table 1: Brute Force algorithm's data	8
Table 2: QuickHull divide-and-conquer algorithm's data	8
Table 3: Average for 2 algorithms data.....	9

1. Introduction

The one of the most important problems in computational geometry is finding the convex hull for a given set of points in the plane, that could be solved computationally. The problem is about constructing a given set of points in plane by a polygonal capsule called convex polygon. The property of the polygon that surrounds the given points in plane making a capsule and it forms the boundary of a convex set.

The determination of convex hull of a point set has successfully been applied in application domains such as data mining, geographical information systems, pattern recognition, artificial intelligence and image processing.

Computer scientists have different approaches of algorithms to solve that problem such as, Brute Force and Divide-and-Conquer but always one of them better than on others.

Brute force approach is simple & straightforward approach to solve this problem A line segment connecting two points P_i and P_j of a set of n points is a part of its convex hull's boundary if and only if all the other points of the set lie on the same side of the straight line through these two points. Repeating this test for every pair of points yields a list of line segments that make up the convex hull's boundary.

Divide-and-Conquer approach solve problems by dividing them into smaller instances, solving each instance recursively and merging the corresponding results to a complete solution. it focusses to find a fast way to merge the small hulls that were recursively generated. Quick Hull is a method of computing the convex hull of a set of points in the plane. It is based on divide-and-conquer strategy and very similar to quick sort , the algorithm works quite good under average circumstances, but processing usually becomes slow in cases of high symmetry or points lying on the circumference of a circle.

In this experiment we find performance for of convex hull algorithms in two approach (brute force and divide-and-conquer) by empirical analysis, then can decide which one is better .

2. Empirical Analysis of Algorithms

2.1 Experiment's purpose

We use empirical analysis in this experiment to compare the performance of solving the convex hull problem of 2 dimensional by using brute force and divide-and-conquer approaches. with different input size and random points. We will measure runtime for each algorithm. After doing this experiment can choose which algorithms are better.

2.2 Efficiency matrix

There are two ways to measure the efficiency of any algorithm :

1. Counter: counting the number of times that the basic operation of the algorithm is to be executed by insert counter variable into a program implementing the algorithm.

2. Physical Time: measure time the program takes to execute the algorithm. there are functions that use in languages to determine the physical time , such as `currentTimeMillis()` in java. The physical time measure has more disadvantage because it depends on many extraneous factors, but also better than counter measure.

physical time measurement is appropriate in this experiment, because it have the advantage of providing specific information about an algorithm's performance in a real computing environment.

Some problems when use physical time measurement:

- 1- Getting different results on repeated runs for the same input take several measurements and calculate the average of them.
- 2- The runtime represented as zero that reason of high speed of the computer, we don't need to consider this problem because this project applies the experiment on large data.
- 3- Another problem is the use of time-sharing system such as UNIX. we don't need to consider this problem because we work on WINDOWS.

2.3 Input Characteristics

The data structure used is array that representing the set of points that specified by their x and y coordinates. We will fill the array with points generated randomly within the range. We will repeat that for different input sizes to measure the efficiency class of algorithms. The purpose of taking a big range of sizes is to test the performance more accurately on a wide variety of inputs.

Input specification:

- The number of points ranges is $3 \leq n \leq 100000$
- The range x- and y-coordinates of each point will be no less than -400 and no greater than 801.
- No point will appear more than once in the same test case.

2.4 Algorithms

There is many algorithm that solve the convex hull problem , but in this section we using 2 algorithms, Brute Force and Quick Hull , we write the reference of each algorithm , and we show some graph to understand the algorithms.

2.4.1. The brute force Algorithm.

From: Textbook - Introduction to The Design and Analysis of Algorithms
Chapter3,Section 3.3, page 112 .

Pseudocode

```
for each point  $P_i$ 
  for each point  $P_j$ 
    where  $P_j \neq P_i$  Compute the line segment for  $P_i$  and  $P_j$ 
      for every other point  $P_k$  where  $P_k \neq P_i$  and  $P_k \neq P_j$ 
        If each  $P_k$  is on one side of the line segment, label  $P_i$  and  $P_j$ 
          in the convex hull
```

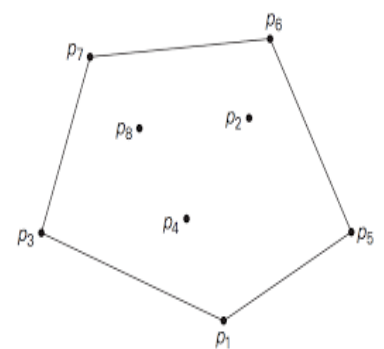


Figure1. Brute Force Algorithm and its graphical illustration

- **The algorithm Convex Hull by brute force.**

ALGORITHM *ConvexHullBF*(n)

// compute convex hull by extreme points

// Input : set of points where $n \geq 2$

//output : Find convex hull from the set of n points

For i = 1 **to** n-1 **do**

For j = i+1 **to** n **do**

$a = X_i - X_j$

$b = Y_i - Y_j$

$c = (1/a)X_j - X_j Y_i$

$K=0$; $d=0$

For k = 1 **to** n && k != i && k != j **do**

$d = a(1/k) + by_k - c$

if ($d > 0$)

$C_1 = C_1 + 1$

else if ($d < 0$)

$C_2 = C_2 + 1$

else $C_3 = C_3 + 1$

If ($C_1 + C_3 = n-2$) **or** ($C_2 + C_3 = n-2$) **then** Point(P_i, P_j)

return points

2.4.2. The Quick Hull divide-and-conquer Algorithm

From : <https://www.cs.jhu.edu/~misha/Spring16/06.pdf>

ALGORITHM *QuickHull*(S)

```
// Compute convex hull by QuickHull algorithm
// Input : set of points where  $S \geq 2$ 
//output : Find convex hull from the set S of n points

Convex Hull := { }

 $(a, b) \leftarrow \text{HorizontalExtrema}(S)$  // Segment (a,b) divides points into 2 groups A, B
 $A \leftarrow \text{RightOf}(S, ab)$  //A upper subset of S
 $B \leftarrow \text{RightOf}(S, ba)$  //B Lower subset of S

 $Q_A = \text{FindHull}(A, ab)$  //call FindHull algorithm to find convex for A subset
 $Q_B = \text{FindHull}(B, ba)$  //call FindHull algorithm to find convex for B subset

return  $\{a\} \cup Q_A \cup \{b\} \cup Q_B$ 
```

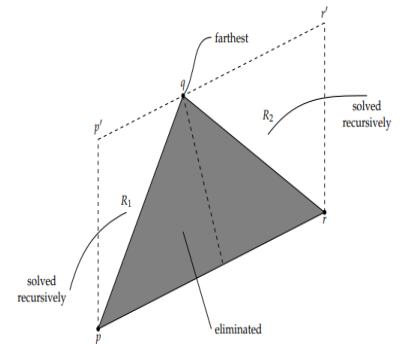


Figure 2. Quick Hull Algorithm and its graphical illustration

- **Helping algorithm to find Hull of subsets**

ALGORITHM *FindHull*(Sk, ab $\in S \times S$)

```
// Find points on convex hull from the set Sk of points
// that are on the right side of the oriented line from a to b

if(  $S_k == \emptyset$  ) return  $\emptyset$ 

 $c \leftarrow \text{Furthest}(S_k, ab)$  // c is farthest point of Sk from segment ab

 $A \leftarrow \text{RightOf}(S_k, ac)$  //A are points on the right side of the oriented line from a to c
 $B \leftarrow \text{RightOf}(S_k, cb)$  //B are points on the right side of the oriented line from c to b.

 $Q_A \leftarrow \text{FindHull}(A, ac)$ 
 $Q_B \leftarrow \text{FindHull}(B, cb)$ 

return  $Q_A \cup \{c\} \cup Q_B$ 
```

2.5 Generate a sample of input

The Random Sampling Points tool is provided for creating randomly distributed points within the specified extent. In general case we use random class to generate random points for a given input size using object of class random.

2.6 Data Observed from Running the Algorithm

We will use some tools for analyzing the algorithms and comparing between them.

- **JAVA language:** using to implement the code.
- **NetBeans application:** using to run the code and show the output.
- **Microsoft Excel:** using to represent result data.

Following tables contains records of 7 test for each input and calculate the total and average of each row on each algorithm of convex hull. Followed by graphs representing these data.

Brute Force algorithm's data									
input size	T(1)	T(2)	T(3)	T(4)	T(5)	T(6)	T(7)	Total	Average
100	39	19	21	25	30	17	26	177	25.2857
350	98	96	89	90	83	65	78	599	85.5714
600	129	103	102	95	133	198	123	883	126.143
1000	181	213	146	171	138	137	158	1144	163.429
2100	602	657	720	575	743	722	710	4729	675.571
3300	870	708	841	856	703	796	883	5657	808.143
5000	1088	1209	1249	1583	993	1347	1509	8978	1282.57
10000	4923	5730	5294	5431	7664	6549	6420	42011	6001.57
Total	7930	8735	8462	8826	10487	9831	9907	64178	9168.29

Table 1: Brute Force algorithm's data

QuickHull algorithm's data									
input size	T(1)	T(2)	T(3)	T(4)	T(5)	T(6)	T(7)	Total	Average
100	3	4	3	5	4	3	3	25	3.57143
350	6	10	5	7	8	14	9	59	8.42857
600	9	6	13	11	8	6	7	60	8.57143
1000	13	12	8	9	11	15	10	78	11.1429
2100	19	17	12	16	21	15	18	118	16.8571
3300	22	14	13	18	24	17	12	120	17.1429
5000	24	19	25	18	21	17	18	142	20.2857
10000	27	24	30	28	26	31	27	193	27.5714
Total	123	106	109	112	123	118	104	795	113.57

Table 2: Quick Hull divide-and-conquer algorithm's data

Average of 2 algorithm (BruteForce and QuickHull)		
input size	Average Brute Force	Average QuickHull
100	25.2857	3.57143
350	85.5714	8.42857
600	126.143	8.57143
1000	163.429	11.1429
2100	675.571	16.8571
3300	808.143	17.1429
5000	1282.57	20.2857
10000	6001.57	27.5714
Total	9168.2831	113.57143

Table 3: Average for 2 algorithms data

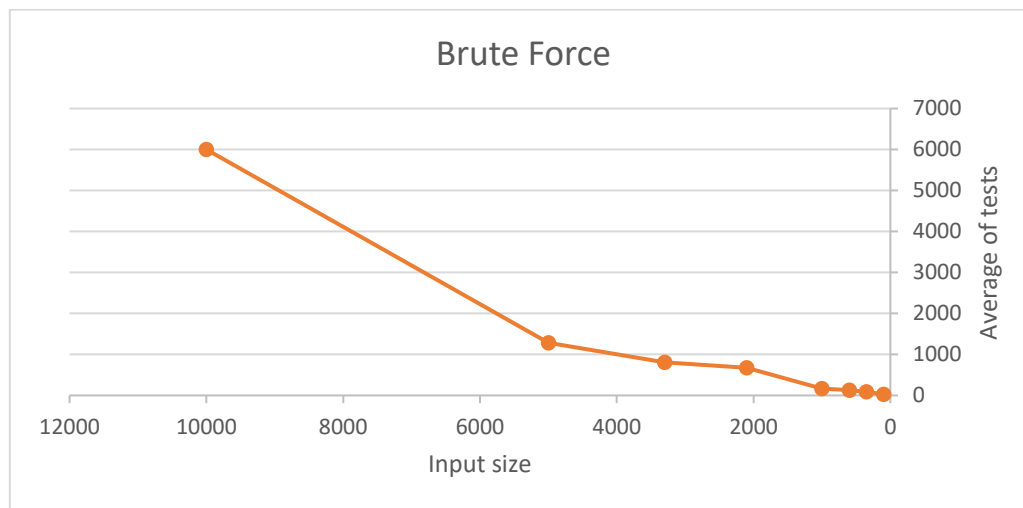


Figure 3: The Brute Force graph

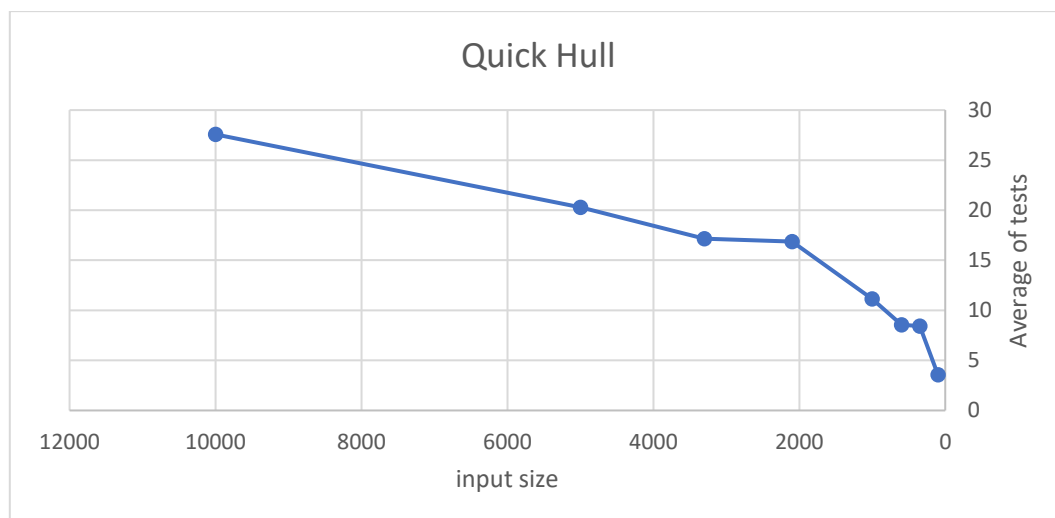


Figure 4: The Quick Hull graph

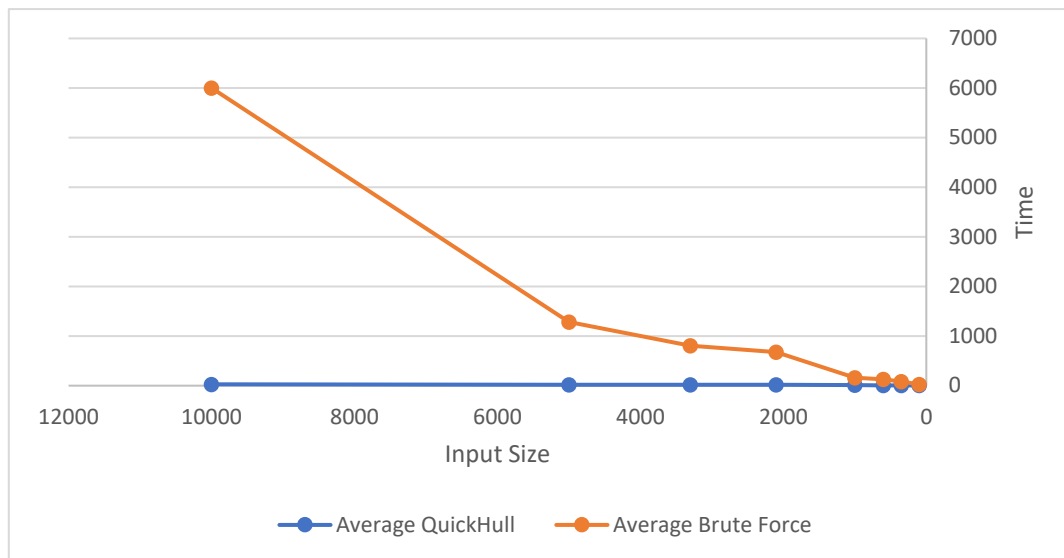


Figure 5: Average for 2 algorithms graph

2.7 Analyze the data obtained

2.7.1. Study Discussion of Results

According to the data in last part 2.6, We observed that the Quick Hull algorithm running time works much faster than the Brute Force algorithms. with less time to execute the task. The runtime of the algorithm depends on the implementation.

We observe that the small input size, Brute Force works with acceptable performance but always Quick Hull Faster than it. For example: when input size equal 100 the performance for Brute Force 25.2857 Quick Hull 3.57143.

We also noticed that the large input size makes big the difference in performance. which Quick Hull give better result. For example: when the input size equal 10,000 the performance for Brute Force 6001.57 and Quick Hull 27.5714 there are big difference between them.

The mathematical analysis of Convex Hull algorithms shows that the efficiency class of each algorithm is as follow:

- **Brute Force**

Best, average and worst case time complexity: $O(n^3)$.

- **Quick Hull**

It is a divide and conquer approach with these efficiency

- Best and Average case time complexity: $O(n \log n)$ when each partition is almost balanced .
- Worst case time complexity: $O(n^2)$ when each points in set is extreme points.

2.7.2 Comparison to theory

1-Iterative algorithm: Brute Force

The **input size**: number of points n

The **basic operation**: the comparison in inner most loop .

The **number of times it is executed** s given by the following sum:

$$c(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^n \sum_{k=1}^n 1 = (n-2) \cdot \frac{(n-1)n}{2}$$

>> So, Best, Worst and average cases is $O(n^3)$

2-Recursive algorithm: QuickHull

The **input size**: number of points n

The **basic operation**: division

The **number of times it is executed** s given by the following recurrence relations:

Note: The function that find the extreme require $O(n)$ time.

Best and Average case:

$$T(n) = 2T(n/2) + O(n)$$

After solving, $T(n) = O(n \log n)$

Worst case

The recurrence relation:

$$T(n) = T(n-1) + O(n)$$

After solving, $T(n) = O(n^2)$

>> So, Best and Average $O(n \log n)$ but the worst case is $O(n^2)$

3. Conclusion

Finally, the empirical analysis relies on real-world data, metrics and results rather than theories and concepts, it's applicable to all algorithms, it uses physical time to determine the performance of the algorithms.

The performance according to our experiment follows:

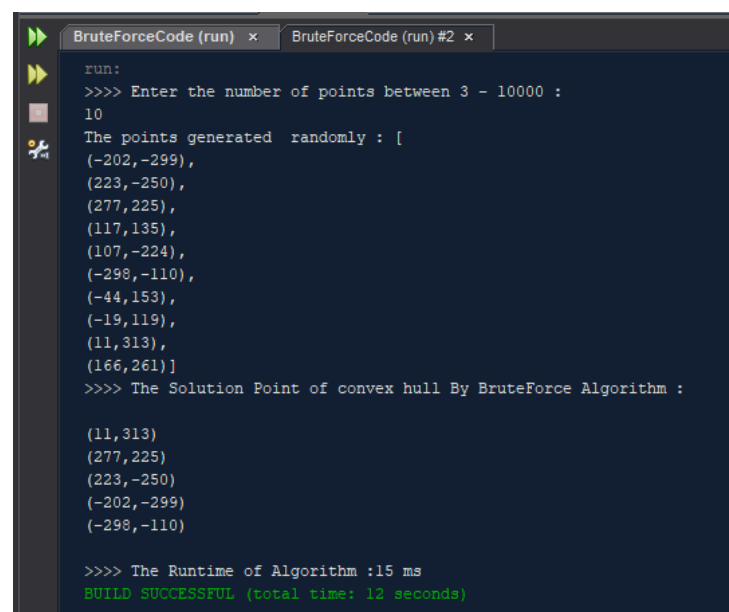
	Brute Force	QuickHull
C(n) best	$O(n^3)$	$O(n \log n)$
C(n) Average	$O(n^3)$	$O(n \log n)$
C(n) Worst	$O(n^3)$	$O(n^2)$

When using empirical analysis, we must take a big range of input sizes, and repeat the same experiment many times and take the average in order to get a more accurate result. In this experiment we compare between Brute Force and Quick Hull algorithms for convex Hull problem and find that Quick Hull is faster with accurate results. Therefore, based in experiment we recommend the use of Quick Hull algorithm for convex Hull problem.

4. Appendix

This section contains Screenshots for the outputs of the two algorithm.

*****Brute Force*****



```
BruteForceCode (run) x BruteForceCode (run) #2 x
Run:
>>>> Enter the number of points between 3 - 10000 :
10
The points generated randomly : [
(-202,-299),
(223,-250),
(277,225),
(117,135),
(107,-224),
(-298,-110),
(-44,153),
(-19,119),
(11,313),
(166,261)]
>>>> The Solution Point of convex hull By BruteForce Algorithm :

(11,313)
(277,225)
(223,-250)
(-202,-299)
(-298,-110)

>>>> The Runtime of Algorithm :15 ms
BUILD SUCCESSFUL (total time: 12 seconds)
```

*****Quick Hull Output*****

```
run:
>>>> Enter the number of points between 3 - 10000 :
10
The points generated randomly : [
(225,-243),
(123,151),
(271,-87),
(-318,-148),
(-396,-393),
(-21,-301),
(-62,118),
(-264,-242),
(-28,-27),
(115,-113)]
>>>> The Solution Point of convex hull By Quick Hull Algorithm :
[
(225,-243),
(-396,-393),
(-318,-148),
(-62,118),
(123,151),
(271,-87)]

>>>> The Runtime of Algorithm :3 ms
BUILD SUCCESSFUL (total time: 17 seconds)
```

5. Reference

- [1]Textbook Introduction to the design and analysis of algorithms 3rd Edition. Anany Levitin. Villanova University.
- [2] <https://www.diva-portal.org/smash/get/diva2:931027/FULLTEXT02.pdf>
- [3] <http://pages.cs.wisc.edu/~dieter/ICPC/13-14/problems/ps5.pdf>
- [4]https://help.xttools.pro/pro/18.0/en/XTools_Pro_Components/Analysis_Tools/Create_Random_Sampling_Points.htm
- [5]https://surface.syr.edu/cgi/viewcontent.cgi?article=1058&context=eecs_techreports
- [6]<https://www.cs.jhu.edu/~misha/Spring16/06.pdf>
- [7]file:///C:/Users/razan/Downloads/hkkr_algorithms_11_00195_v3.pdf