# TIME AND SPACE TRADEOFFS IN POINT LOCATION

A Thesis
in
Computer Science

Presented to the Faculty of the University
of Missouri–Kansas City in partial fulfillment of
the requirements for the degree

MASTER OF SCIENCE

by

MOUNIKA GONUGUNTLA

Bachelor of Technology,
Koneru Lakshmaiah College of Engineering , Guntur, India, 2018

Kansas City, Missouri,Unitied States of America
2024

TIME AND SPACE TRADEOFFS IN POINT LOCATION

Mounika Gonuguntla, Candidate for the Master of Science Degree

University of Missouri–Kansas City, 2024

## ABSTRACT

We preprocess the input subdivision with $n$ points on the plane in $O(n\sqrt{\log n})$ time and store them in $O(n/t)$ space to facilitate point location in $O(\log t)$ time, where $t$ is an adjustable parameter. When $t$ is a constant we get $O(n)$ space and constant time. When $t = O(n)$ we get constant space and $O(\log n)$ time.

APPROVAL PAGE

The faculty listed below, appointed by the Dean of the School of Science and Engineering, have examined a thesis titled “Time and Space Tradeoffs in Point Location,” presented by Mounika Gonuguntla, candidate for the Master of Science degree, and certify that in their opinion it is worthy of acceptance.

Supervisory Committee

Yijie Han, Ph.D., Chair
School of Science & Engineering

Wajeb Gharibi, Ph.D.
School of Science & Engineering

Jesse Lowe, Ph.D.
School of Science & Engineering

CONTENTS

ILLUSTRATIONS

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my heartfelt gratitude to all those who have contributed to the completion of this thesis.

First and foremost, I would like to express my sincere gratitude to my supervisor, Dr. Yijie Han, for his assistance, invaluable mentorship, and expert guidance throughout this journey. His guidance and advice enabled me to accompish this thesis and fostered my academic growth in immeasurable ways. The results in this thesis have been published as a conference paper [8].

I extend my sincere appreciation to the members of my thesis committee, Dr. Wajeb Gharibi and Dr. Jesse Lowe, for their invaluable insights, constructive feedback, and rigorous evaluation of my work. Their expertise and scholarly contributions have significantly enriched the quality of this thesis.

I am grateful to my family and friends for their constant encouragement, understanding, and unwavering belief in my abilities. Their support and encouragement have been a constant source of strength and motivation throughout this process.

I would also like to acknowledge the University of Missouri - Kansas Cityfor providing the necessary resources, facilities, and academic environment conducive to research and learning.

Lastly, I extend my gratitude to all the individuals, colleagues, and peers who have provided assistance, encouragement, and support in various ways, directly or indirectly, during the course of this thesis.

# CHAPTER 1

# INTRODUCTION

Point location is a problem in computational geometry that has been studied by many researchers [5,6,13–15] . Earlier results for this problem [5,13,14] take $O(n \log n)$ preprocessing time, $O(n)$ storage and $O(\log n)$ time for locating the point. Recently Han and Saxena achieved constant space and $O(\log n)$ time for locating the point [9] and Chaganti and Han achieved $O(n)$ space and constant time for locating the point [2] . In this paper we show that we can preprocess the input subdivision in $O(n\sqrt{\log n})$ time to facilitate the point location in $O(n/t)$ space and $O(\log t)$ time, where $t$ is an adjustable parameter. When $t$ is a constant, we get $O(n)$ space and constant time. When $t = O(n)$ we get constant space and $O(\log n)$ time. The approach we take is to convert the real numbers of point coordinates to integers and then multiple integers can be packed into one word. This speeds up the algorithm. The tool we used to do this real number to integer conversion is the real number sorting algorithm shown in [10] which has time $O(n\sqrt{\log n})$ and runs on the computation model used in computational geometry.

# CHAPTER 2

## THE $O(N\sqrt{\log N})$ TIME SORTING ALGORITHM FOR SORTING REAL NUMBERS

It used to be that real numbers can only be sorted with comparison sorting algorithms. Comparison sorting has a lower bound of $\Omega(n \log n)$ time complexity for sorting $n$ numbers [4]. This is because for the input $n$ numbers there are $n!$ permutations of these $n$ number and only one permutation (the right permutation) permutes the input $n$ numbers to the sorted order assuming that these $n$ input numbers are all different. A comparison of two input numbers $a$ and $b$ will have the result of $a > b$ or $a < b$. Among the $n!$ permutations of $n$ input numbers some permutations $A$ comply with the result of $a > b$ and other permutations $B$ comply with the result of $a < b$. At least one of $A$ and $B$ has no less than $n!/2$ permutation. If $a > b$ then the permutations in $B$ need not be further considered as they do not satisfy $a > b$. But we need to further figure out which permutation in $A$ is the right permutation. If $a < b$ then permutations in $A$ can be precluded as they do not satisfy $a < b$. But we need to figure out which permutation in $B$ is the right permutation. In the worst case the comparison of $a$ and $b$ will preclude the smaller set of $A$ and $B$ and thus we have at least $n!/2$ permutations remaining. Each subsequent comparison will, in the worst case, cut the number of remaining permutations to half. Thus we need at least $\log(n!)$ comparisons to cut the number of permutations to 1. By Sterling's approximation $n! \approx (n/e)^n$, where $e$ is the natural number. Thus the number of comparisons needed is $\Omega(n \log n)$. This reasoning is from [16].

It has been shown in [10] that real numbers can be sorted in $O(n\sqrt{\log n})$ time. In [10] real numbers are first converted to integers while preserving their relative order. After converting to integers these integers can be sorted in $O(n \log \log n)$ time [11, 12].

CHAPTER 3

POINT LOCATION

## 3.1 Preprocess

We first convert the real numbers of the point coordinates to integers. This is accomplished by sorting the real numbers of $x$ and $y$ coordinates of $n$ points. This is done in $O(n\sqrt{\log n})$ time [10]. We will do this separately for nonnegative numbers and negative numbers. Let $a_0, a_1, ..., a_{2n-1}$ be sorted sequence of these coordinates. We separate negative numbers from nonnegative numbers and do $a_{i+1} - a_i$ when $a_{i+1}$ and $a_i$ are both positive or both negative for $i = 0, 1, ..., 2n-1$, and find the most significant bit $b_i$ of $a_{i+1} - a_i$ (method given in the following). Let $b_i$ be the $m$-th bit (counting from the least significant bit starting from 0). If we cut $a_{i+1}$ and $a_i$ at bit $b_i$, that is: remove all bits less significant than the $m$-th bit, then the resulting numbers for $a_{i+1}$ and $a_i$ are integers (after shifting $m$ bits or multiplying by $2^{-m}$) and their order is preserved. We will find the smallest $m$ value $m'$ from the $2n-1$ $b_i$ values and cut all $a_i$'s, $0 \leq i < 2n$, at bit $m'$. This converts all real numbers of point coordinates to integers and the original order of these real numbers is preserved in the converted integers.

Real number $a$ can be cut at bit $m'$ and converted to an integer by the operation $int(a * 2^{-m'})$.

The most significant bit can be found by the method shown in [7]. It can also be found by taking the logarithm base 2 then take the integer part of the result. Another

approach is to first scale all real numbers so that their absolute value is less than 1. Then take the largest value $b$ among $1/|a_{i+1} - a_i|$ , $i = 0, 1, 2n - 1$. Value $b$ can then be used to convert $a_i$ to an integer as $int(ba_i)$. This converts all $a_i$'s to integers while preserving their order.

After converting real numbers for point coordinate values to integers we then triangulate the subdivision. This can be done in $O(n)$ time using [3].

We then pack the 2's complement values of the integer coordinate for $x$ ($y$) coordinate into one word. Say that each integer has $B$ bits, we will add an extra two bits of $00$ (positive) or $01$ (negative) in front of every integer value. The first bit is needed to prevent overflow to the next integer when doing pairwise addition. The second bit is the sign bit. Thus, we will use $B + 2$ bits for each integer. Let the word for the packed $x$ values be $X$ and let the word for the packed $y$ values be $Y$. For an edge with the equation $a_i x + b_i y + c_i = 0$ we will also pack the converted integer values (of $B+2$ bits) of $a_i, b_i, c_i$ into one word for all edges $i = 0, 1, ...$ of the triangles of the subdivision.

We will also prepare some constants. The constants we need are $C_1 = (0^{B+1}1)^n$, $C_2 = (010^B)^n$. Each of these constant can be precomputed in $O(\log n)$ time. $C_1$ is used to duplicate a $B$ bit integer $a$ for $n$ copies by the $aC_1$ operation. $C_2$ is used to extract sign bits.

Example:

Line 1: $2x + 3y = 0$,

Line 2: $3x - 6y - 4 = 0$,

Line 3: $-4x + 5y - 6 = 0$.

We will evaluate point $(10, 12)$ against these line equations.

Assuming that we use $B = 5$ bits for each integer. In order for performing the multiplication for packed integers we use $2 * (B + 2) = 14$ bits for each integer. Because for multiplication the signs are multiplied separately than the numbers and thus, we pack absolute values into a word and multiply the numbers as:

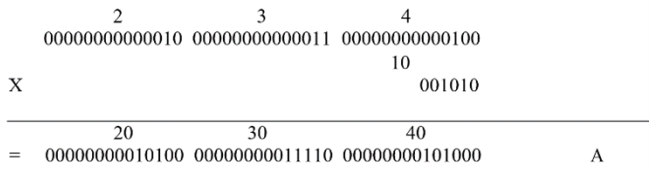


Figure 1: Multiplication for Packed Integer

Here 2, 3, 4 are the absolute values of the coefficients of $x$ in these 3 equations and 10 is the $x$ coordinate of the query point.

The sign bits are XORed together as

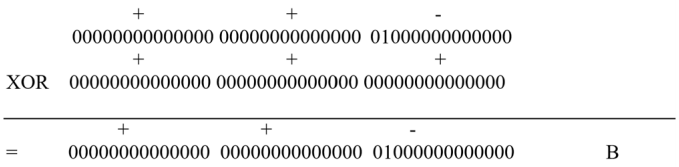


Figure 2: XOR for Signed Bits

Here the first line is the sign for 2, 3, -4 and the second line is the 3 copies of the sign for 10. Note that we use the second bit from left as the bit indicating the sign as if we use 2's complement representation we need the leftmost bit to take the possible carry. If we use the sign plus magnitude result, we need OR $A$ and $B$ together as:

| | | | | |
|---|---|---|---|---|
| | 00000000010100 | 00000000011110 | 00000000101000 | A |
| OR | 00000000000000 | 00000000000000 | 01000000000000 | B |
| = | 00000000001010 | 00000000010100 | 01000000110010 | Sign plus magnitude result |

Figure 3: Sign Representation for Packed Integers with 2's Complement

If we need to convert the result to 2's complement (in order for do addition) we need to shift $B$ 12 bits (number of bits for each integer -2) to the right to get

| | | | |
|---|---|---|---|
| 00000000000000 | 00000000000000 | 00000000000001 | C |

Figure 4: Shifting for Conversion to 2's Complement

We then subtract $C$ from $B$ to get:

| | | | | |
|---|---|---|---|---|
| | 00000000000000 | 00000000000000 | 01000000000000 | B |
| - | 00000000000000 | 00000000000000 | 00000000000001 | C |
| | 00000000000000 | 00000000000000 | 00111111111111 | D |

Figure 5: Subtraction of C from B

We then AND $A$ with $D$ to extract negative numbers:

| | | | | |
|---|---|---|---|---|
| | 00000000010100 | 00000000011110 | 00000000101000 | A |
| AND | 00000000000000 | 00000000000000 | 00111111111111 | D |
| | 00000000000000 | 00000000000000 | 00000000010100 | E |

Figure 6: Extraction of Negative Numbers

We then shift $B$ to the left by 1 bit and subtract $E$ to get 2's complement of $E$:

Now get nonnegative numbers by subtracting $E$ from $A$:

| | | | |
|---|---|---|---|
| | 00000000000000 | 00000000000000 10000000000000 | B<<1 |
| - | 00000000000000 | 00000000000000 00000000101000 | E |
| | 00000000000000 | 00000000000000 01111111011000 | F, 2's complement of E |

Figure 7: Shifting and Subtraction to Obtain 2's Complement of E

| | | | |
|---|---|---|---|
| | 00000000010100 | 00000000011110 00000000101000 | A |
| - | 00000000000000 | 00000000000000 00000000101000 | E |
| | 00000000010100 | 00000000011110 00000000000000 | G |

Figure 8: Subtraction of E from A to Obtain Nonnegative Numbers

Now OR $F$ and $G$:

| | | | |
|---|---|---|---|
| | 00000000000000 | 00000000000000 01111111011000 | F |
| OR | 00000000010100 | 00000000011110 00000000000000 | G |
| | 00000000010100 | 00000000011110 01111111011000 | 2's complement result |

Figure 9: OR Operation between F and G

Using the same technique we can get (3, -6, 5)x12, add (0, -4, -6) and then see the result of testing (10, 12) against all lines.

### 3.2 Locating a Point

For a query point $q = (x_0, y_0)$, we convert its coordinate real values to integers by cutting them at bit $m'$ (the same bit used to cut the points in the preprocessing stage) to convert them to a $B + 2$ bit integers $x_1$ and $y_1$. We then test $(x_1, y_1)$ against all edges of all triangles. Thus, we can decide for each triangle, whether $(x_1, y_1)$ is within the triangle, outside the triangle or on the edge of a triangle. Note that these triangle's edges' functions are $ax + by + c = 0$ where $a$, $b$, $c$ are converted integers. These multiple integer

addition, subtraction, multiplication is computed in constant time by packing them to a word as shown in Section 3. These operations are used in previous research papers of other researchers. It is not clear who was the first to use these operations. These operations are sufficient for us to determine the triangle that contains the integer coordinate of the query point in constant time.

### 3.3 Error Correction and Tradeoff between Time and Space

If our algorithm tells that the query point $p = (p_x, p_y)$ is located within triangle $A = \triangle abc$ then we can test the real value coordinates of $p$ to the 3 lines of A constructed from the real value coordinates of the 3 points $a, b, c$ of $A$. Thus we need to fetch 6 real values $a_x, a_y, b_x, b_y.c_x.c_y$ from the input $O(n)$ real values which are the coordinates of the input points. If our algorithm tells that $p$ is on an edge of a triangle then we need to fetch the two end points $a, b$ of this edge and then test out whether $p$ (using the real value of its coordinates) is to the left, to the right or on the edge. If our algorithm tells that the point $p$ is at a vertex $a'$ then the real value coordinates $a$ of $a'$ need to be fetched and compared with the real value of $p$ to tell whether $p = a$. If $p = a$ then we are done. Otherwise $p \neq a$ and thus we need to cut $p$ and $a$ at a precision such that the converted points $p'$ and $a''$ with rational number coordinates are not equal. Note that the set $S$ of the coordinates of other points have been converted to rational numbers (integers) before and we need not to re-convert them. The converted point $a''$ cannot be equal to any converted points in $S$ because $a''$ has higher precision than $a'$ and $a'$ is not equal to any converted points in $S$. Thus, we can retest $p$ against the converted points of $S$ union with $\{a''\}$. Again, this takes

constant time. Thus, we need to fetch a constant number of real values from the $O(n)$ input real values. Using [9] we can store $t$ real numbers into a constant number of real numbers and each of these $t$ real numbers can be extracted in $O(\log t)$ time. This will require us to use $O(n/t)$ space.

# CHAPTER 4

## MAIN THEOREM

An $n$ vertex planar subdivision can be preprocessed in $O(n\sqrt{\log n})$ time and stored in $O(n/t)$ space to support $O(\log t)$ time point location, where $c \leq t \leq n$ is an adjustable parameter and $c$ is a constant.

# CHAPTER 5

# CONCLUSIONS

We studied point location for planar subdivision. We showed that $n$ points subdivision on the plane can be preprocessed in $O(n\sqrt{\log n})$ time and stored in $O(n/t)$ space to support $O(\log t)$ time point location, where $t$ is a parameter. When $t$ is a constant we get $O(n)$ space to support constant time point location. When $t$ is $n$ we get constant space to support $O(\log n)$ time point location.

REFERENCE LIST


[1] A. Andersson, T. Hagerup, S. Nilsson, R. Raman. Sorting in linear time? *Proc. 1995 Symposium on Theory of Computing STOC'1995, 427-436(1995). Also in Journal of Computer and System Science 57*, 74-93(1998).

[2] S. Chaganti, Y. Han. Point location in constant time. Paper 2401.02440 in arXiv.org, 2024.

[3] B. Chazelle. Triangulating a simple polygon in linear time, *Discrete and Computational Geometry*, 6 (3): 485-524(1991).

[4] T. H. Corman, C. E. Leiserson, R. L. Rivest, C. Stein. Introduction to Algorithms. Fourth Edition, The MIT Press. 2022.

[5] D. Dobkin, R. J. Lipton. Multidimensional searching problems. *SIAM Journal on Computing*. 5 (2): 181-186(1976). doi:10.1137/0205015.

[6] H. Edelsbrunner, L. J. Guibas, J. Stolfi. Optimal point location in a monotone subdivision. *SIAM Journal on Computing*. 15 (2): 317-340(1986).

[7] M. L. Fredman, D. E. Willard. BLASTING through the information theoretic barrier with FUSION TREES. *Proc.1990 ACM Symposium on Theorem of Computing (STOC'1990)*, 1-7(1990).

[8] M. Gonuguntla, Y. Han. Time and space tradeoffs in point location. *in Proc. 2023 Int. Conf. on Computational Science and Computational Intelligence (CSCI'2023)*, December 2023.

[9] Y. Han, S. Saxena. Storage in computational geometry. Paper 2302.11821 in arXiv.org, 2023.

[10] Y. Han. Sorting real numbers in $O(n\sqrt{\log n})$ time and linear space. *Algorithmica* 82, 966- 978(2020).

[11] Y. Han. Deterministic sorting in $O(n \log\log n)$ time and linear space. *Proc. 2002 ACM Symp. on Theory of Computing STOC'02, 702-707(2002). Also in Journal of Algorithms, 50*, 96-105(2004).

[12] Y. Han. A linear time algorithm for ordered partition. *Proc. 2015 International Frontiers in Algorithmics Workshop (FAW'15), LNCS 9130*, 89-103(2015).

[13] D. G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM J. Computing*. 12 (1): 28-35(1983).

[14] R. J. Lipton, R. E. Tarjan. Applications of a planar separator theorem. *Proc. 18th Annual IEEE Symposium on Foundations of Computer Science*, 162-170(1977).

[15] F. P. Preparata, M. Ian Shamos. Computational Geometry, An Introduction. Springer-Verlag. 1985.

[16] M. Snir. On parallel searching. *SIAM J. Computing*, Vol. 14, 3, 688-708(1985).

# VITA

Mounika Gonuguntla was born on December 24, 1996 in Giddalur, Andra Pradesh, India. She completed her elementry schooling at Bhagawan Sri SathyaSai Vidhyalayam, Chandapuram, India. She pursued her bachelor's degree in Electronics and Computer Engineering from Koneru Lakshmaiah Education Foundation in Andhra Pradesh, India and graduated in 2018 with top score. Following her undergraduate studies, she worked as a Software Engineer at Uniphore for two and half years in India. Later, she began her master's in Computer Science at the University of Missouri-Kansas City in August 2022, majoring in Computer Science.