THE RABIN-KARP ALGORITHM

LOUAY ELBICHE | SIDDHESH JETHE | ZUBIN KADVA
GROUP 2 | ALGORITHMICS 2017

ANALYSIS OF ALGORITHMS | SPRING 2017

STRING MATCHING

- Returns the position where a given pattern is found in a text

Pattern **P** of length **m**

String **S** of length **n**

$P \subseteq S$

S[i] = P[0]

S[i + 1] = P[1]

...

S[i + m + 1]= P[m - 1]

$\sum$ = alphabet

$\sum = \{A, \dots, Z\}$

$\sum = \{0, 1\}$

$\sum = \{A, C, G, T\}$

EXAMPLE

- $\sum = \{X, Y, Z\}$

- S = X Y X Y X Y Z Y

- P = X Y X Y Z

Index	0	1	2	3	4	5	6	7
Text	X	Y	X	Y	X	Y	Z	Y

THE PROBLEM

Given a string $S = s_1, s_2, \ldots, s_n$ of characters in some alphabet $\Sigma = \{a_1, a_2 \ldots, a_K\}$ and a pattern or query string $P = p_1, p_2 \ldots, p_m;\ m < n$ in the same alphabet, find all occurrences of $\mathbf{P}$ in $\mathbf{S}$.

ALGORITHMS

- Naïve string searching algorithm

- Rabin-Karp string searching algorithm

- Knuth-Morris-Pratt algorithm

- Boyer-Moore string search algorithm

BRUTE FORCE

- Naïve search algorithm

```
function NaiveSearch(string, pattern)

    n := length(string)

    m := length(pattern)

    for i = 0 to n – m + 1

        for j = 0 to m

                if string[i + j] != pattern[j]

                        break

        if j == m

                return i

    return not found
```

ALGORITHM TRACE

Text	F	I	N	D		M	E
Pattern	M	E					

ALGORITHM TRACE

Index	0	1	2	3	4	5	6
Text	F	I	N	D		M	E
Pattern	M	E					

ALGORITHM TRACE

Index	0	1	2	3	4	5	6
Text	F	I	N	D		M	E
Pattern		M	E				

ALGORITHM TRACE

Index	0	1	2	3	4	5	6
Text	F	I	N	D		M	E
Pattern			M	E			

ALGORITHM TRACE

Index	0	1	2	3	4	5	6
Text	F	I	N	D		M	E
Pattern				M	E		

ALGORITHM TRACE

Index	0	1	2	3	4	5	6
Text	F	I	N	D		M	E
Pattern					M	E	

ALGORITHM TRACE

Index	0	1	2	3	4	5	6
Text	F	I	N	D		M	E
Pattern						M	E

ALGORITHM TRACE

Index	0	1	2	3	4	5	6
Text	F	I	N	D		M	E
Pattern						M	E

ANALYSIS

```
function NaiveSearch(string, pattern)
    for i = 0 to n − m + 1
        for j = 0 to m
            if string[i + j] != pattern[j]
                break
        if j == m
            return i
    return not found
```
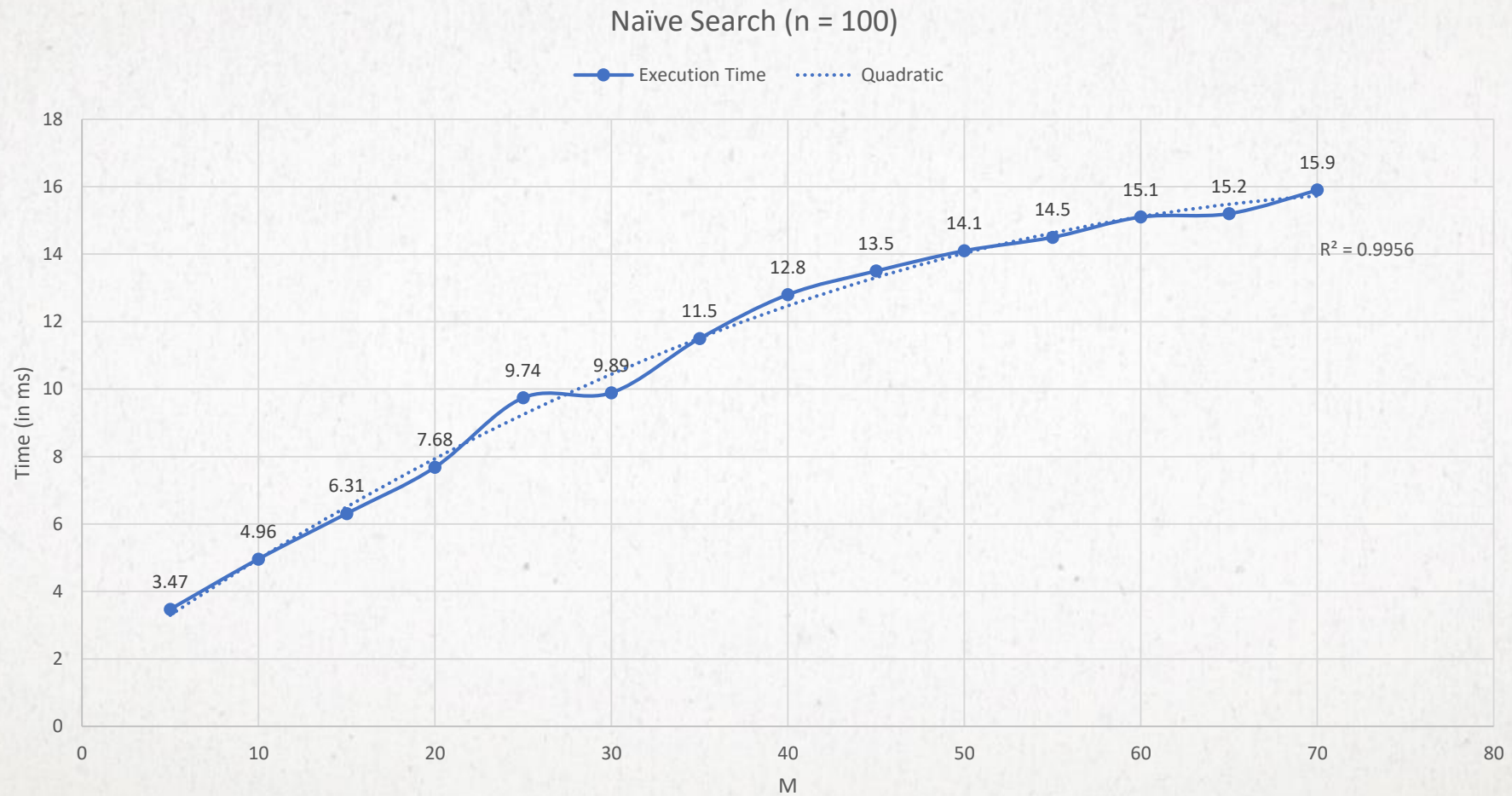
Complexities		Naïve Search
Performance	**Best**	$\Omega(m)$
	Average	$\theta(m+n)$
	Worst	$O(mn)$
Space		$O(1)$

VISUAL PERFORMANCE

Naïve Search (m = 10)

Execution Time ——— Quadratic ········

R² = 0.9937

Data points: 0.081, 0.625, 0.922, 1.37, 2.5, 3.22, 3.45, 4.42, 4.8, 5.5, 5.8, 6.37, 7.7, 8.14, 8.95

Y-axis: Time (in ms)
X-axis: N

VISUAL PERFORMANCE

Naïve Search (n = 100)

──●── Execution Time ⋯⋯⋯ Quadratic

Time (in ms) vs M

3.47, 4.96, 6.31, 7.68, 9.74, 9.89, 11.5, 12.8, 13.5, 14.1, 14.5, 15.1, 15.2, 15.9

R² = 0.9956
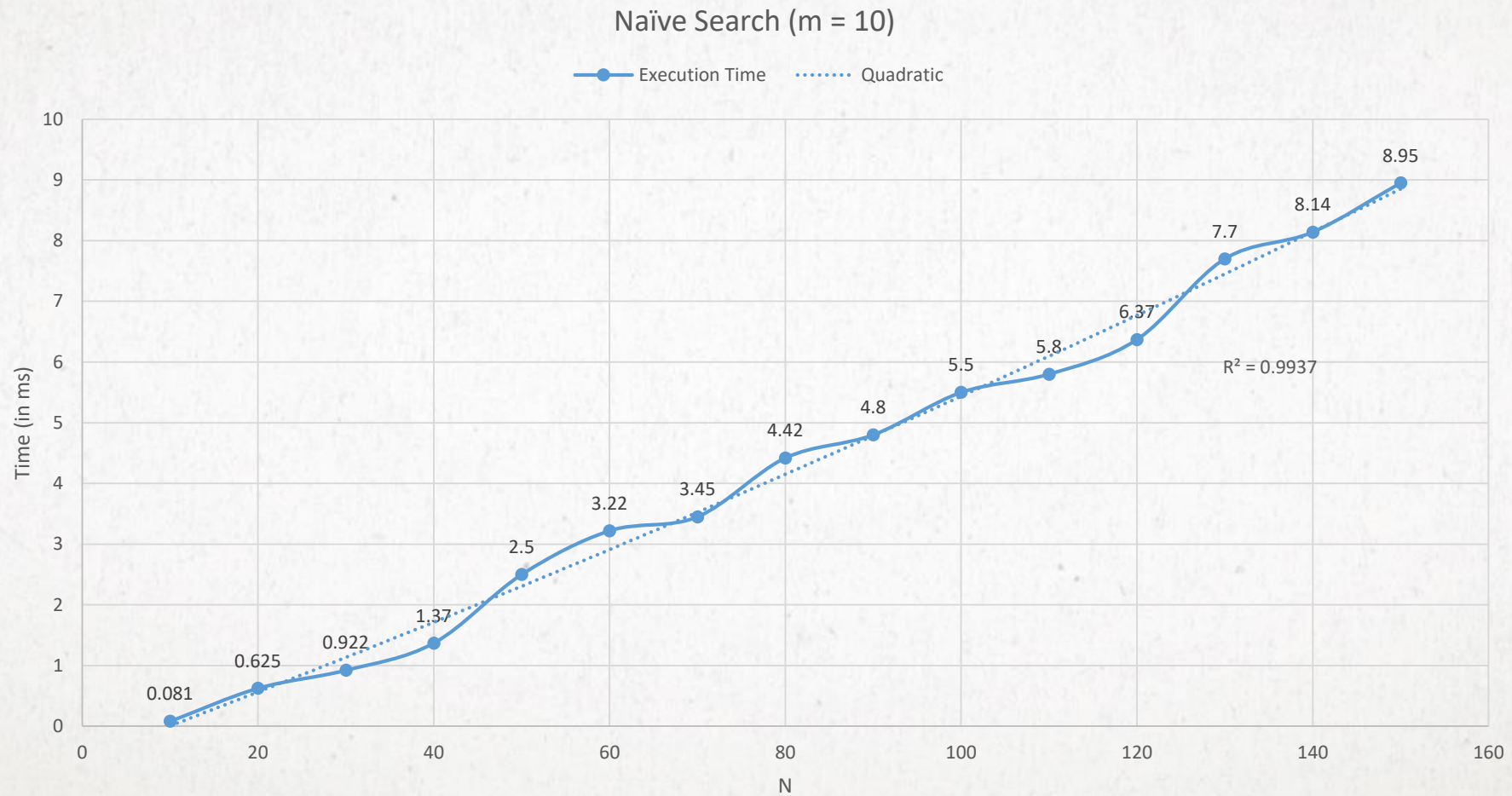
RABIN-KARP

- Based on computing hash of a pattern and then comparing it

```
function RabinKarp(string, pattern)

    hpattern := hash(pattern);

    hstring := hash(string)

    for i = 1 to n – m + 1

        if hstring = hpattern

            if string = pattern

                return i

        hstring:= reHash(i, i + m - 1)

    return not found
```

RABIN-KARP

- The hash is calculated as:

$$\sum_{i=0}^{length} val(string[i]) * prime^i$$

```
function hash(string, length)

    for i = 0 to length

        hash := hash + value(string[i]) * prime^i

    return hash
```

RABIN-KARP

- Updating the hash is done as:

$$new\_hash = old\_hash - val(string[old\_index])$$

$$new\_hash = new\_hash / prime$$

$$new\_hash = new\_hash + val(string[new]) * prime^{length-1}$$

```
function reHash(string, length, old, new, hash)
    newHash := hash - value(string[old])
    newHash := newHash / prime
    newHash := newHash + value(string[new]) * prime^(length - 1)
    return newHash
```

MODULAR RABIN-KARP

- To reduce:
  - Size of the hash result and
  - \# of hash collisions

- Elements:
  - Radix **d**
  - Prime **q**

BASIS FOR RABIN-KARP

- q = 23

i	0	1	2	
	6	4	7	% 23 = 3

i	0	1	2	3	
	9	6	4	7	
0	9	6	4		% 23 = 21
1		6	4	7	% 23 = 3

KEY COMPUTATION

- d = 10

i	0	1	2	3	4	...
current value	6	3	2	1	7	
new value		3	2	1	7	
		3	2	1		current value
-		3	0	0		
			2	1		subtract leading digit
*			1	0		multiply by radix
		2	1	0		
+				7		add trailing digit
		2	1	7		new value

HORNER'S METHOD

i	0	1	2	
	3	1	2	
0	3	% 23 = 3		
1	3	1	% 23 = (3 * 10 + 1) % 23 = 8	
2	3	1	2	% 23 = (8 * 10 + 2) % 23 = 13

MODULAR RABIN-KARP

```
function rabin-karp-matcher(T, P, d, q)

    n := T.length

    m := P.length

    h := d^{m-1} mod q

    p := 0

    t_0 := 0

    for i = 1 to m

        p := (d * p + P[i]) mod q

        t_0 := (d * t_0 + T[i]) mod q

    for s = 0 to n - m

        if p == t_s

            if P[1 .. m] == T[s + 1 .. s + m]

                return s

        if s < n - m

            t_{s+1} = (d * (t_s − T[s + 1] * h) + T[s + m + 1]) mod q

    return not found
```
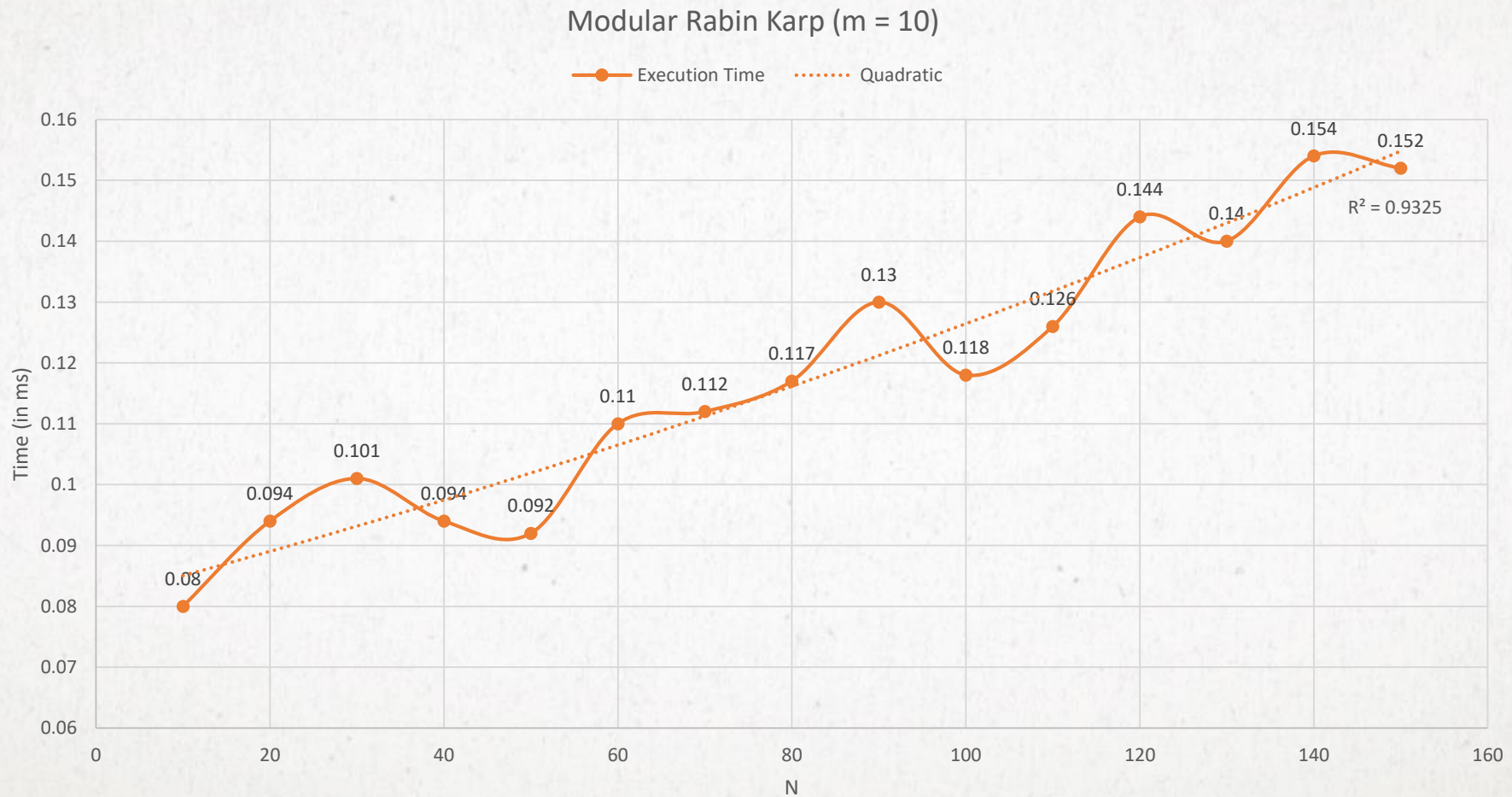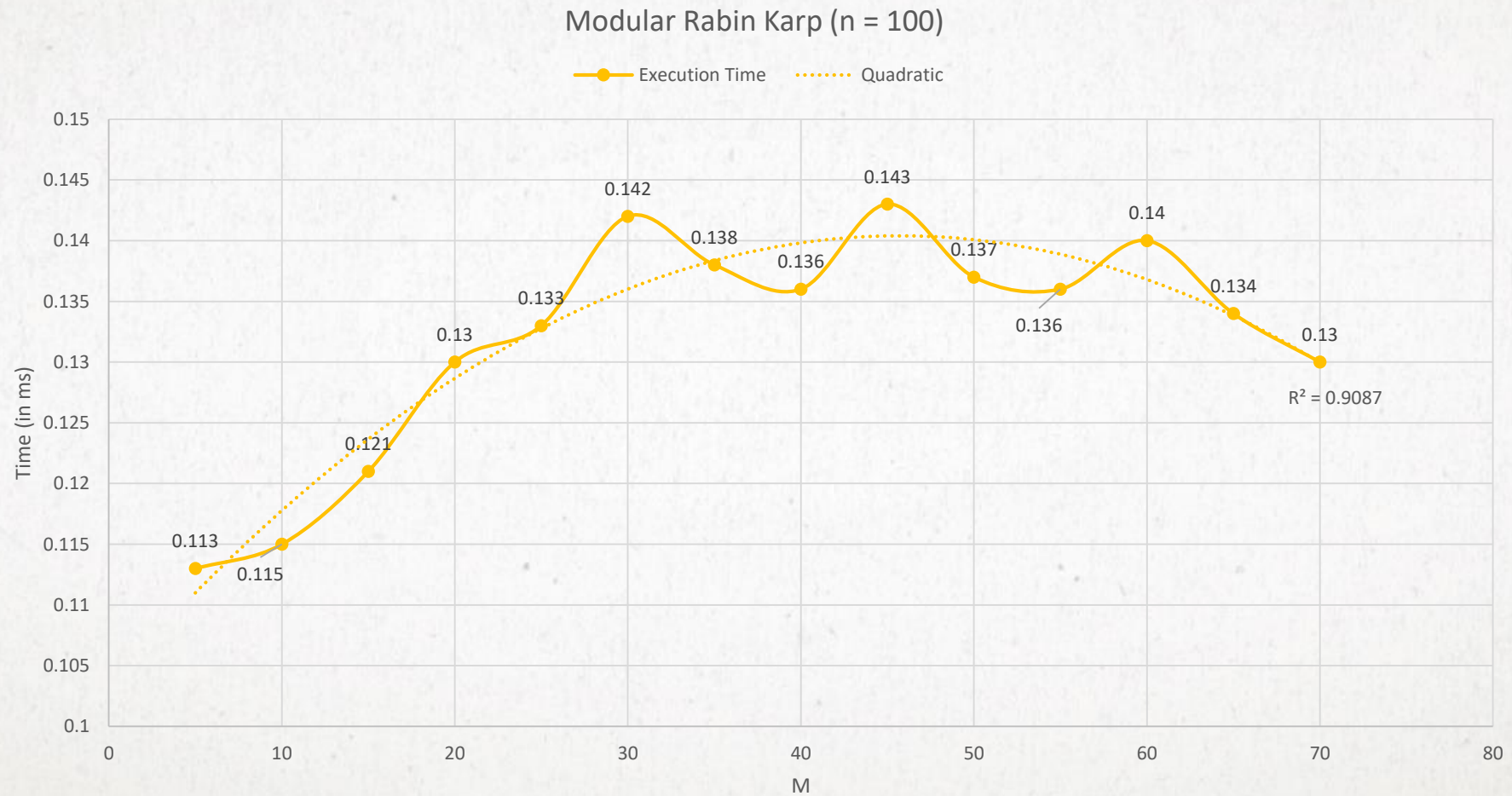
ANALYSIS

```
function rabin-karp-matcher(T, P, d, q)

    n := T.length

    m := P.length

    h := d^(m-1) mod q

    p := 0

    t₀ := 0

    for i = 1 to m

        p := (d * p + P[i]) mod q

        t₀ := (d * t₀ + T[i]) mod q

    for s = 0 to n - m

        if p == t_s

            if P[1 .. m] == T[s + 1 .. s + m]

                return s

        if s < n - m
```

$$t_{s+1} = (d * (t_s - T[s + 1] * h) + T[s + m + 1]) \bmod q$$

```
    return not found
```

Complexities		Rabin Karp
Performance	**Best**	$\Omega\,(m + n)$
	Average	$\theta\,(m + n)$
	Worst	$O\,(mn)$
Space		$O\,(m)$

VISUAL PERFORMANCE
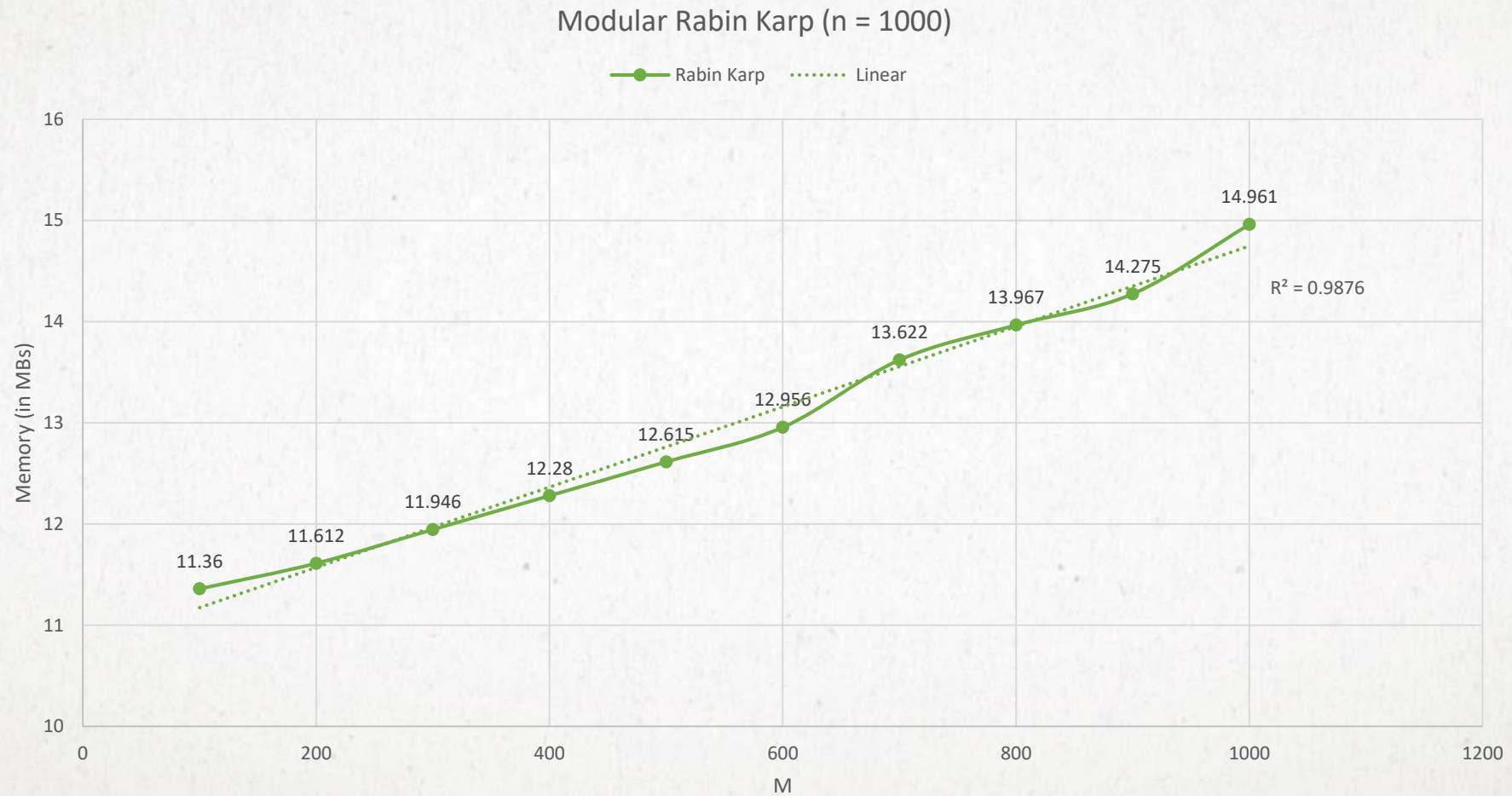
Modular Rabin Karp (m = 10)

VISUAL PERFORMANCE

Modular Rabin Karp (n = 100)

VISUAL SPACE

Modular Rabin Karp (n = 1000)

— Rabin Karp · · · · Linear

COMPARATIVE PERFORMANCE

Execution time (m = 10)

COMPARATIVE PERFORMANCE

Execution Time (n = 100)
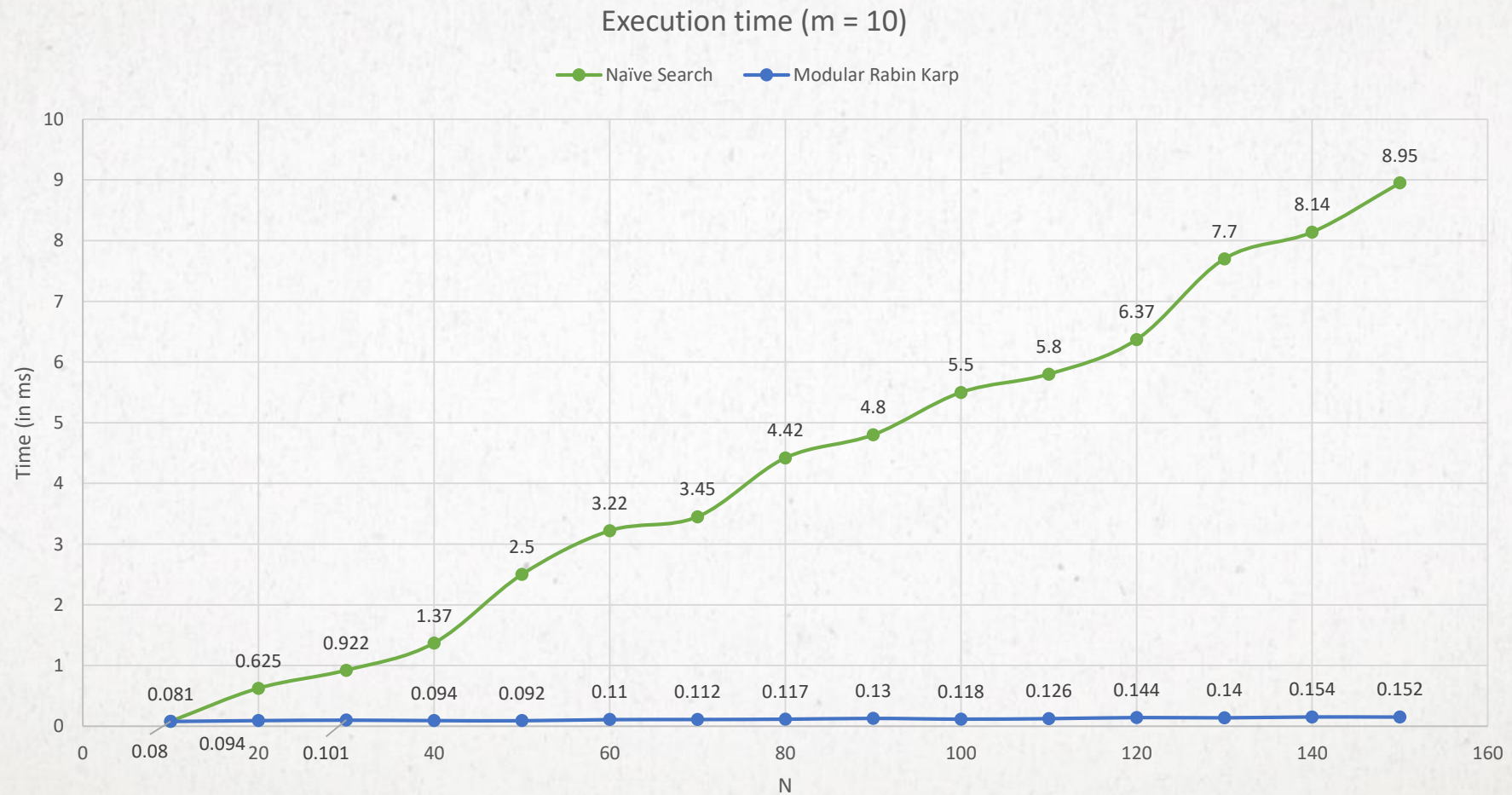
APPLICATION EXAMPLE: BIOINFORMATICS

- Certain known nucleotide and/or amino acid sequences have properties known to biologists

  - **ATG** is a string which must be present at the beginning of every protein (gene) a DNA sequence.

- A primer is a conserved DNA sequence used in the Polymerase Chain Reaction (PCR) to identify the location of the DNA sequence that will be amplified

- Genetic code is redundant. Consider the following sequence of nucleotides: **GCTACTATTTTTCAT**

  - When read in forward frame 0, this sequence encodes the following sequence of amino acids: **ATIFH**

APPLICATION EXAMPLE: BIOINFORMATICS

- Because of redundancy of the genetic code, the following sequences of nucleotides will produce the same sequence of amino acids

```
GCT ACT ATT TTT CAT
GCc ACc ATT TTa CAc
GCc ACc ATc TTg CAT
GCT ACT ATc TTa CAc
GCg ACc ATa TTc CAc

 A   T   I   F   H
```

Amino Acid	SLC	DNA codons
Isoleucine	I	ATT, ATC, ATA
Leucine	L	CTT, CTC, CTA, CTG, TTA, TTG
Valine	V	GTT, GTC, GTA, GTG
Phenylalanine	F	TTT, TTC
Methionine	M	ATG
Cysteine	C	TGT, TGC
Alanine	A	GCT, GCC, GCA, GCG
Glycine	G	GGT, GGC, GGA, GGG
Proline	P	CCT, CCC, CCA, CCG
Threonine	T	ACT, ACC, ACA, ACG
Serine	S	TCT, TCC, TCA, TCG, AGT, AGC
Tyrosine	Y	TAT, TAC
Tryptophan	W	TGG
Glutamine	Q	CAA, CAG
Asparagine	N	AAT, AAC
Histidine	H	CAT, CAC
Glutamic acid	E	GAA, GAG
Aspartic acid	D	GAT, GAC
Lysine	K	AAA, AAG
Arginine	R	CGT, CGC, CGA, CGG, AGA, AGG
Stop codons	Stop	TAA, TAG, TGA

http://www.cbs.dtu.dk/courses/27619/codon.html
http://users.csc.calpoly.edu/~dekhtyar/448-Spring2013/lectures/lec03.448.pdf

APPLICATION EXAMPLE: BIOINFORMATICS

- *Example*: Consider for following string S = "**ATTCCGT**".
  - Suppose we are looking for four-letter substrings.

```
function end-presentation()

    THANK YOU

    if questions?

        solve()
```