DECEMBER 2017

- Agenda
  - Introductions and thanks to Microsoft
  - Quantum News
  - Food/Pizza
  - Shor
- Presentations can be found at github.com/NYCQuantumComputing
- Twitter @NYCQuantum
- Looking for hosts, presenters, topics, suggestions

RECAP 2017

- March 2017 – Kickoff
- April 2017 – Grover search, IBM's Quantum Experience, math behind Grover
- May 2017 – Thanks to DWAVE for their technical presentation
- June 2017 – Thanks to Chris Monroe from IONQJ
- July 2017 – Quantum Entanglement
- September 2017- Bell's Inequality
- October 2017 – IBM presented QISKIT
- November 2017 – Nathan Weibe from Microsoft

INTRODUCTIONS

NEWS / INTERESTING

- IBM ThinkQ Conference December 6-8, 2017 -> videos
- Interesting papers
    - "Quantum Artificial Life in an IBM Quantum Computer"
    - Quantum Machine Learning
    - Others?

IDEAS FOR 2018

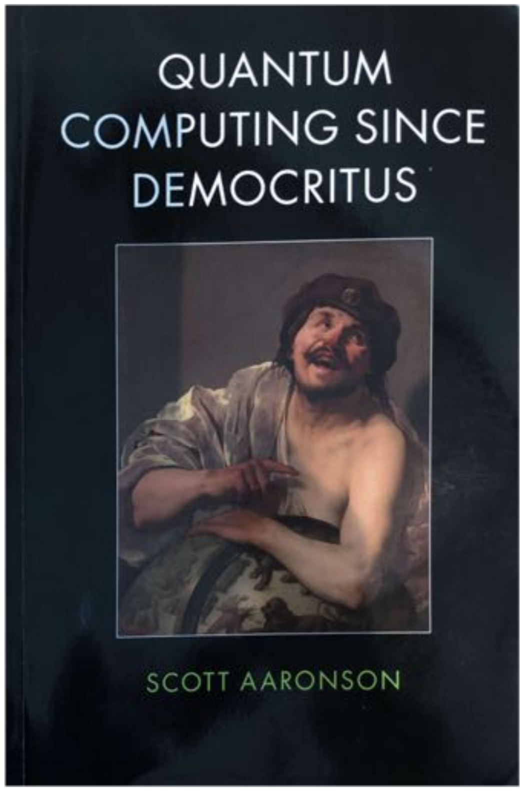
- Intro class – feedback for agenda, speakers
- Quantum Games
- Microsoft Topological Computing
- Complexity Theory
- Quantum Machine Learning meets Classical Modeling
- Physics of Quantum Computing

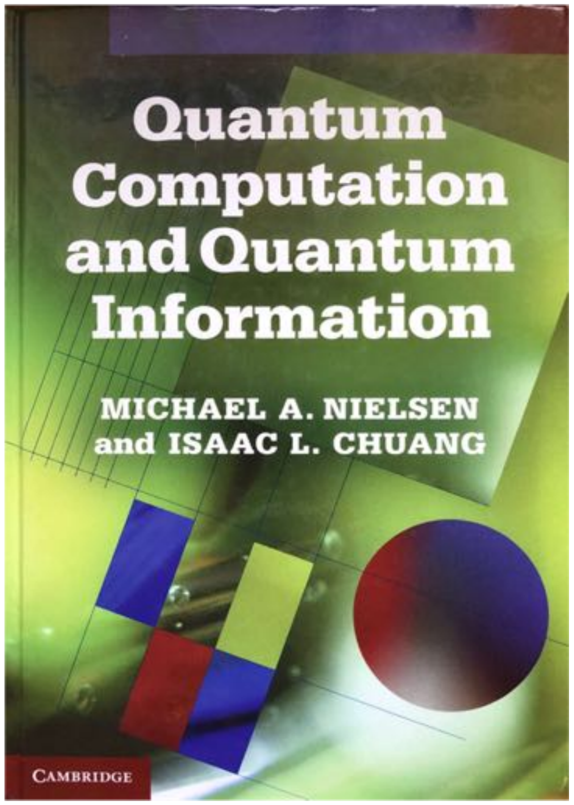
SHOR/FACTORING

NYC QUANTUM COMPUTING MEETUP

DECEMBER 20, 2017

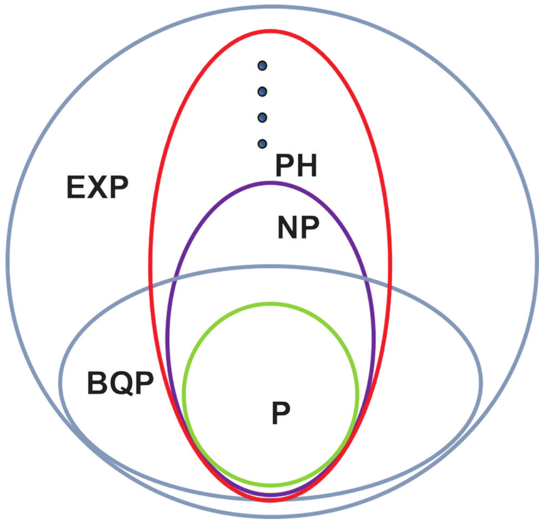
BACKGROUND


QUANTUM COMPUTING SINCE DEMOCRITUS
SCOTT AARONSON

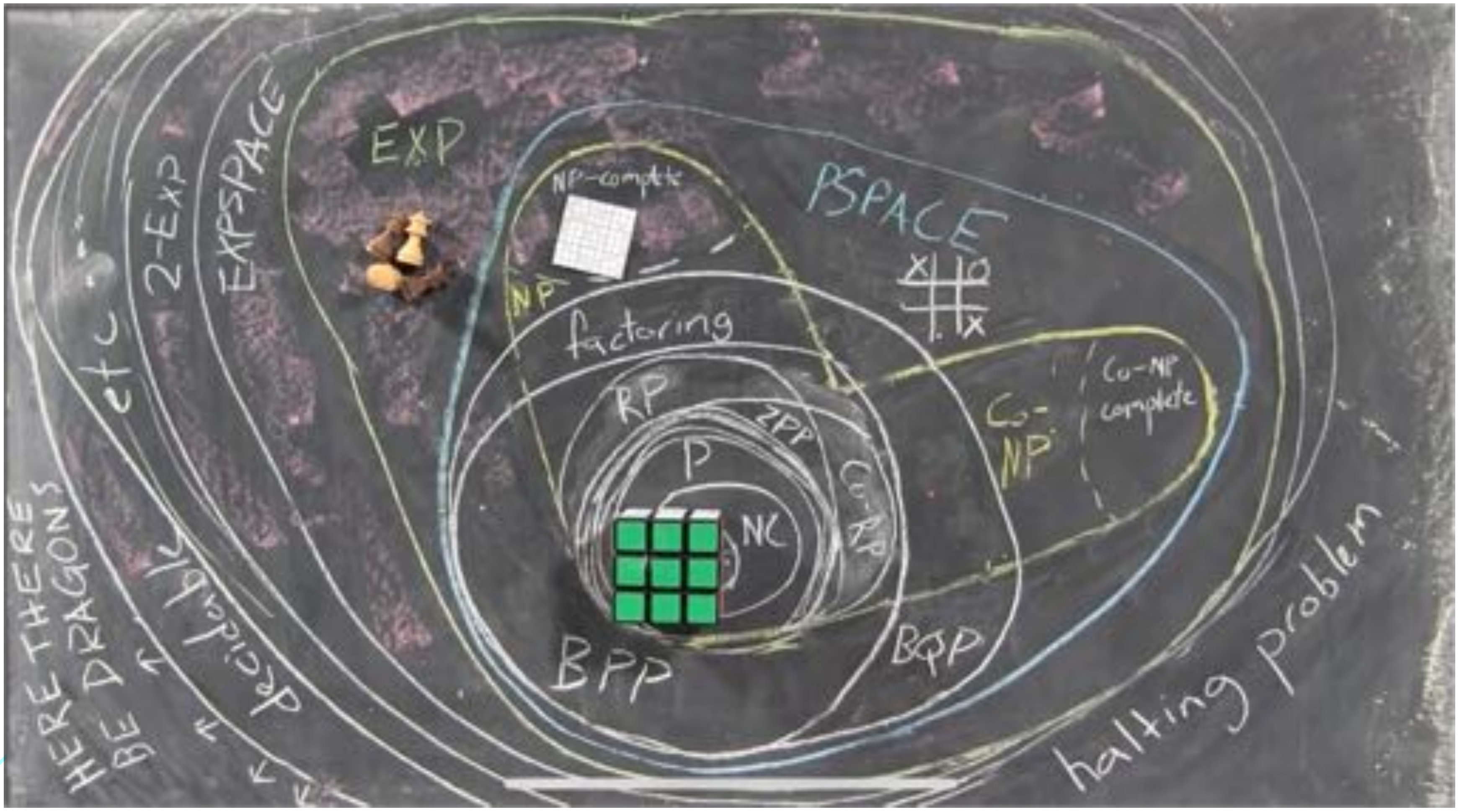

Quantum Computation and Quantum Information
MICHAEL A. NIELSEN and ISAAC L. CHUANG
CAMBRIDGE

AARONSON – “THE COMPLEXITY PETTING ZOO"

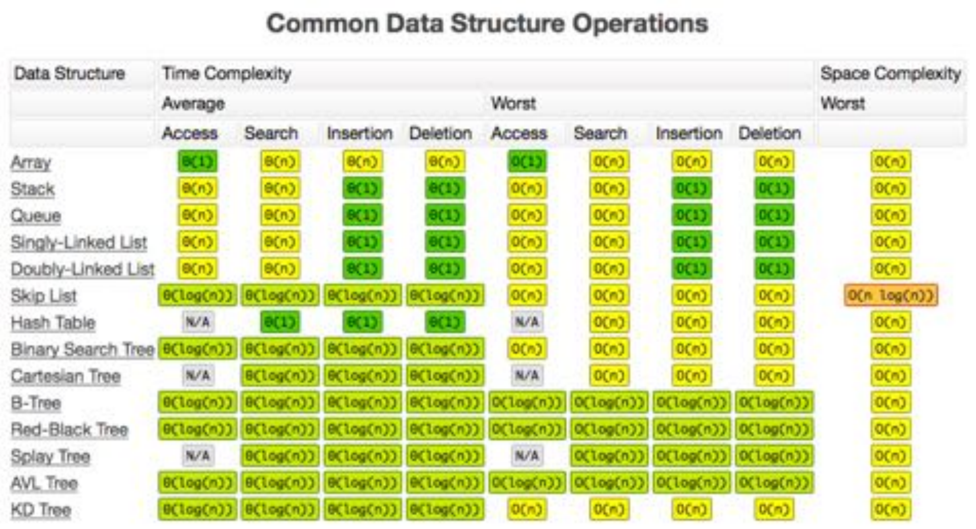
- Complexity theory analyzes the amount of resources time and/or space to solve a problem
  - Theoretical CS usesTuring machines as the basis
- P is the class of problems solved in polynomial time
  - ”problem is a decision problem”
    - f(k bits) -> {True, False) that gets solved in TIME(n^k)
- PSPACE is P space where Time is unlimited
- EXP is the class of problems solved in TIME($2^{n^\wedge k}$)
  - Examples : Chess 10^{120} moves
- NP
  - EXP problems where a proof exists that that can be checked in polynomial time
  - Favorite example : checkmate, factoring a 10000 digit number
- P≠NP, NP-Hard, NP Complete, BQP (Another talk…)
- First 100 pages of Democritus



BIGGER COMPLEXITY ZOO

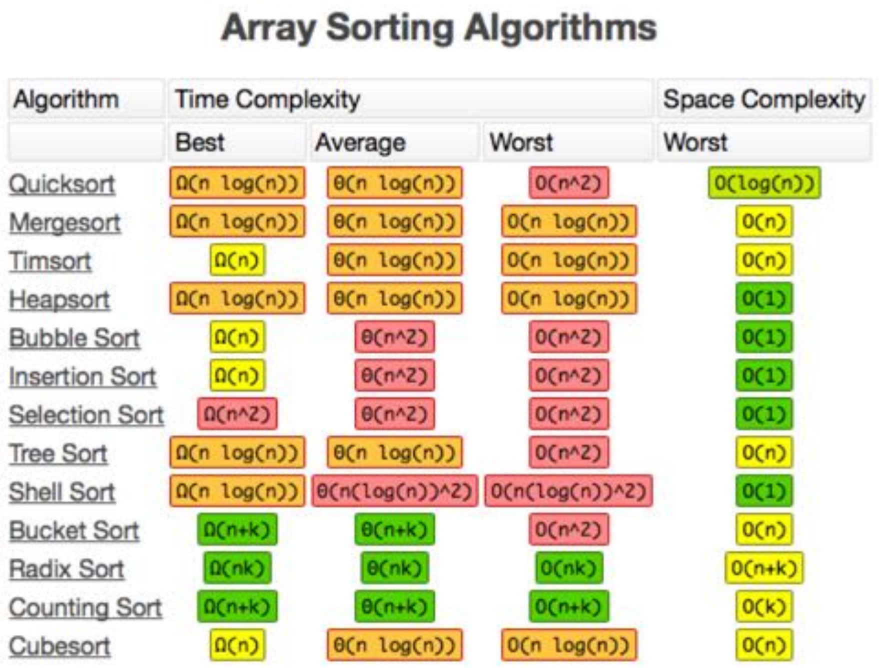


BIG O CHARTS VIA ERIC ROWELL



Common Data Structure Operations

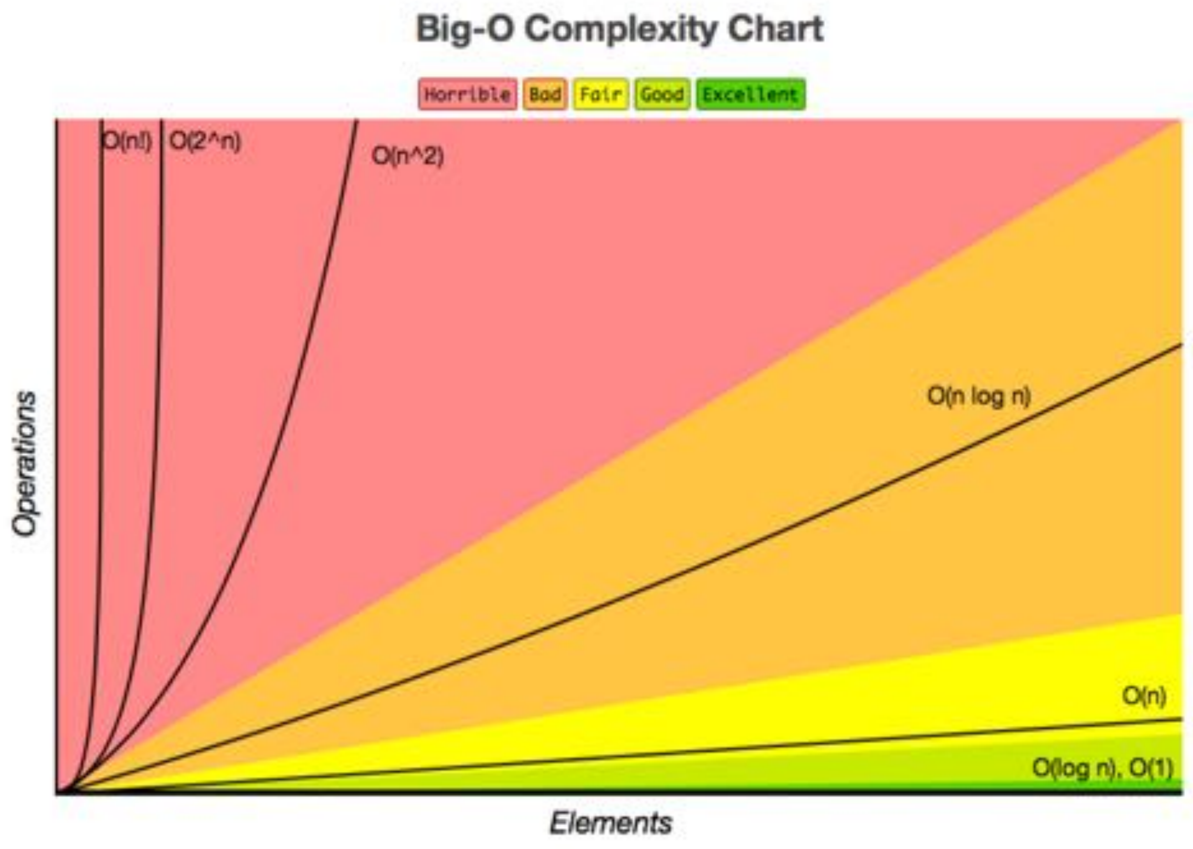
Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)



Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	Θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	Θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)

BIG O CHARTS VIA ERIC ROWELL



PUBLIC KEY CRYPTOGRAPHY (PKI) BACKGROUND

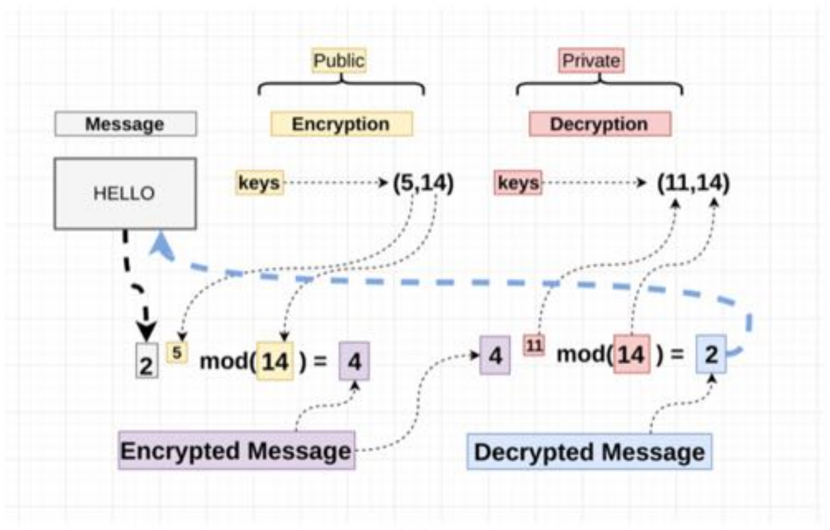
- Different varieties of PKI - RSA is the most popular
- The problem of distributing a shared secret
  - Completely random, one-time use pad is ideal (that's a lot of secrets)
  - Ideal aka "Information-theoretically secure where key size needs to be the size of the message" – provable
- What happens if you can encode using a public key, decode using a private key?
  - `encrypt(pubkey, m) -> c`
  - `decrypt(privkey, c) -> m`
- Can you make encryption computationally easy and decryption "easy yet computationally hard?
  - E.g. Can you define a P-SPACE encrypt and a P-SPACE decrypt + "trapdoor" information function but is NP hard without "trapdoor information"?

PKI : FACTORING LARGE NUMBERS

- Multiplication is easy
  - Multiplication is in P; the effort is based on the total number of digits
- Factoring 200+ digit number composed of 2 x 100+ digit primes is very (assumed, but not yet proven to be) hard
  - NP-Hard
  - Time is $O(2^n)$ where n is the number of bit in N
  - Cleverness has reduced the complexity to $O(2^{n\hat{}1/3})$
  - 750bits is 200 digit number
- However, given one of the primes (e.g. the private key), factoring the number makes it a P space problem

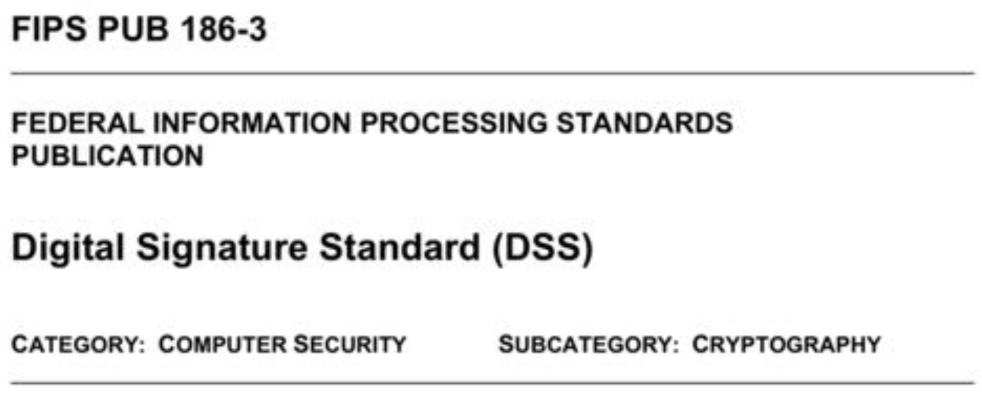
THE MATH FOR RSA

- Public Key
  - n product of two random primes p, q
  - e relative prime to $\theta = (p - 1) * (q - 1)$
  - $(1 < e < \theta$ where gcd $(e, \theta) - 1)$
- Private Key
  - $d * e = \text{mod } (p - 1) \times (q - 1)$
  - $d = e^{-1} \text{ mod } (p - 1) \times (q - 1)$
  - *$(d * e) = k\ (p - 1) \times (q - 1)$ (not mod)*
- Message
  - m = plaintext
- Encryption
  - $c = m^e (\text{mod } n)$
- Decryption
  - $m = c^d (\text{mod } n)$
- Since
  - $c^d = (m^e)^d = m^{ed} = m^{k(p-1)(q-1)+1} = m * m^{k(p-1)(q-1)} = m * 1 = m$ (all mod m)
  - Where by Fermat, $m^{p-1} = 1 \text{ mod } p$, $m^{q-1} = 1 \text{ mod } q$



mini rsa

VERY DEPENDENT ON RANDOM PRIMES

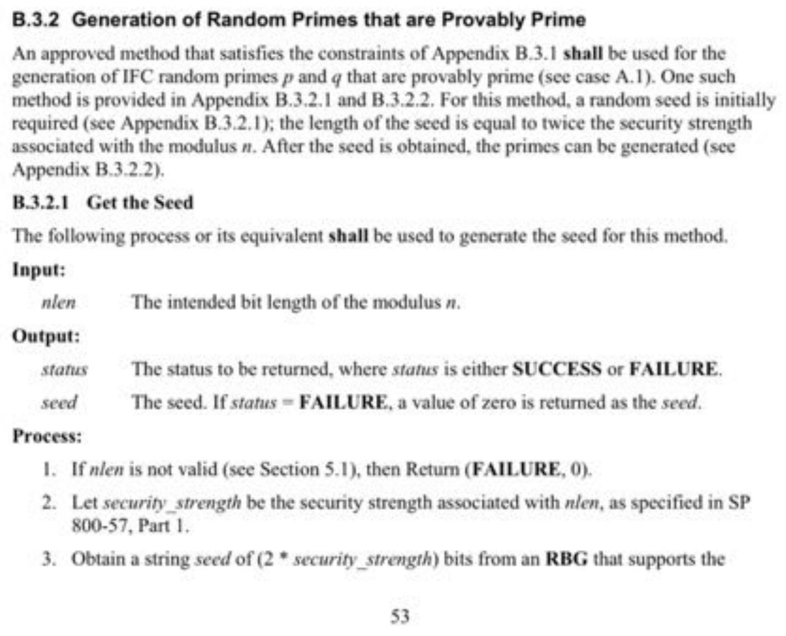


FIPS PUB 186-3

FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION

Digital Signature Standard (DSS)

CATEGORY: COMPUTER SECURITY **SUBCATEGORY: CRYPTOGRAPHY**



B.3.2 Generation of Random Primes that are Provably Prime

An approved method that satisfies the constraints of Appendix B.3.1 **shall** be used for the generation of IFC random primes *p* and *q* that are provably prime (see case A.1). One such method is provided in Appendix B.3.2.1 and B.3.2.2. For this method, a random seed is initially required (see Appendix B.3.2.1); the length of the seed is equal to twice the security strength associated with the modulus *n*. After the seed is obtained, the primes can be generated (see Appendix B.3.2.2).

B.3.2.1 Get the Seed

The following process or its equivalent **shall** be used to generate the seed for this method.

Input:

nlen	The intended bit length of the modulus *n*.

Output:

status	The status to be returned, where *status* is either **SUCCESS** or **FAILURE**.
seed	The seed. If *status* = **FAILURE**, a value of zero is returned as the *seed*.

Process:

1. If *nlen* is not valid (see Section 5.1), then Return (**FAILURE**, 0).
2. Let *security_strength* be the security strength associated with *nlen*, as specified in SP 800-57, Part 1.
3. Obtain a string *seed* of (2 * *security_strength*) bits from an **RBG** that supports the

53

THE MATH FOR FACTORING

- It's all about efficiency and complexity
  - Or separately out the various components of factoring and deciding how quantum techniques can best be used
- The classic tricks for factoring
  - Modular arithmetic O(n)
  - Euclid's theorem O(log n)
  - Efficient calculation of ***gcd*** (greatest common denominator) $O(n^2)$
  - Efficient period finding $O(2^n)$ (Oops! Bad!)

HOW HARD IS BRUTE FORCE FACTORING?

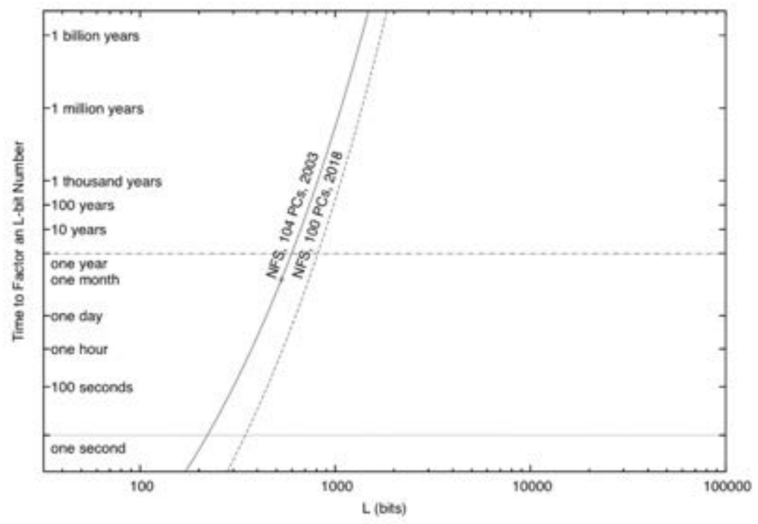


Figure 3.2: Scaling of number field sieve (NFS) on classical computers. Both horizontal and vertical axes are log scale. The horizontal axis is the size of the number being factored, in bits.

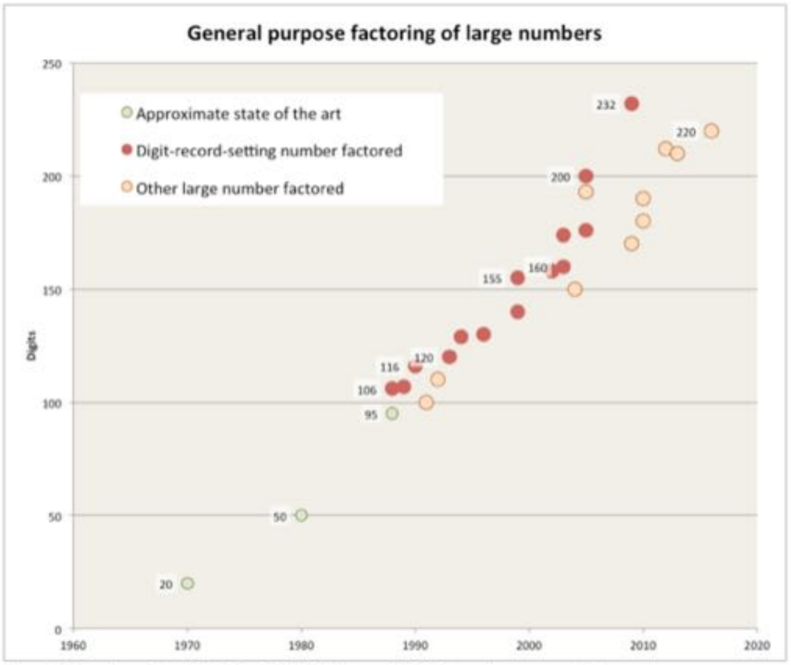


Figure 1: Size of numbers (in decimal digits) that could be factored over recent history. Green 'approximate state of the art' points do not necessarily represent specific numbers or the very largest that could be at that time—they are qualitative estimates. The other points represent specific large numbers being factored, either as the first number of that size ever to be factored (red) or not (orange). Dates are accurate to the year. Some points are annotated with decimal digit size, for ease of reading.

GREATEST COMMON DENOMINATOR

- Euclid's Algorithm
- What does it do?
  - gcd (32, 21), where 32 is 1 x 2^5 and 21 is 1 x 3 x 7 -> gcd is 1
  - gcd (35, 60) where 35 is 1 x 5 x 7 and 60 is 1 x 2^2 x 3 x 5 -> gcd is 5
- What's the algorithm?
  - If a > b, r is remainder when a is divided by b & $r \neq 0$, then gcd(a, b) = gcd (b, r)
  - Repeatedly use gcd(a, b) = gcd (b, r) = gcd (r, r') when r = 0 -> a = kb,
- What's the upper bound running time?
  - O(log a)

MODULAR ARITHMETIC

- N = 35
- a ≡ b mod N (b = qN + a, q is any integer)
- 24 = 24 mod 35
- 14 = 49 mod 35
- 34 = -1 mod 35
- Addition
  - x = 24 + 49 mod 35
  - 3 = 73 mod 35
  - 3 = (14 + 24) mod 35
- Multiplication
  - 24 * 49 mod 35
  - 14 * 24 mod 35
- Modular arithmetic is very efficient

THE SCHEME

- Factor N
- Select a random value

  $x \in \{2, 3.., N - 1\}$, $\gcd(x, N) = 1$ (prime test)
- ***Determine the period r of the sequence***

  $x^0, x^1, x^2, x^3, \ldots \bmod N$ ($r > 0$, s.t. $x^r = 1 \bmod N$)
  - (… r is even, $x^{r/2} \pm 1 \neq 0 \bmod N$)
- Then

  $x^r = x^{(r/2)2} \equiv 1 \bmod N$

  $(x^{r/2} + 1)(x^{r/2} - 1) \equiv 0 \bmod N$ (no remainder)

  $(x^{r/2} + 1)(x^{r/2} - 1) = kN$ for some k
- Divisors : $\gcd(x^{r/2} + 1, N)$ & $\gcd(x^{r/2} - 1, N)$

PERIOD FINDING (THE MATH)

N	35		x^r mod N			
x	**2***	**3**	**5**	**7**	**43**	**31**
0	1	1	1	1	1	1
1	2	3	5	7	8	31
2	4	9	25	14	29	16
3	8	27	20	28	22	6
4	16	11	30	21	1	11
5	32	33	10	7	8	26
6	29	29	15	14	29	1
7	23	17	5	28	22	31
8	11	16	25	21	1	16
9	22	13	20	7	#NUM!	6
10	9	4	30	14	#NUM!	#NUM!
11	18	12	10	28	#NUM!	#NUM!
12	1	1	15	21	#NUM!	#NUM!
13	2	3	5	7	#NUM!	#NUM!
14	4	9	25	14	#NUM!	#NUM!
15	8	27	20	28	#NUM!	#NUM!
16	16	11	30	21	#NUM!	#NUM!
17	32	33	10	#NUM!	#NUM!	#NUM!
18	29	29	15	#NUM!	#NUM!	#NUM!
19	23	17	5	#NUM!	#NUM!	#NUM!
20	11	16	#NUM!	#NUM!	#NUM!	#NUM!
By Hand - r	**12**	**12**	**r = ?**	**r = ?**	**4**	**6**
(r2+1)(r2-1)	**4095**	**531440**			**3418800**	**887503680**
GCD (r^2+1)	**5**	**5**			**5**	**7**
GCD (r^2-1)	**7**	**7**			**7**	**5**

JUST FIND THE PERIOD, RIGHT?

- 750 bits is 200 digit number
- If there's just two primes in the 750 bit number, where's the period?
- In fact, you need at least $2N^2$ bits to figure out the period
- $(2*750)^{2} = 2250000$ bits
- Note that's it a polymonial number of bits

THE MATH FOR FACTORING – CONT'D

- ….
- The classic tricks for factoring
  - ….
  - Efficient period finding $O(2^n)$ (Oops! Bad!)
- Replacing classic period finding with "This one weird Quantum Trick"
  - Very efficient period finding – Shor's QFT
  - (Hint: Use quantum superposition to solve period finding)
  - Superposition reduces the complexity to a $O(n^2)$

END OF SLIDES / QUESTIONS

AARONSON – “THE COMPLEXITY PETTING ZOO"

- Classic Search f:[N] = {0,1} takes ~N queries
- Grover Quantum Search, f:[N] = $\sqrt{N}$ queries
- Classic Period Finding f:[N] = N takes ~N queries
- Quantum Period Finding f:[N] = N takes $\sqrt{N}$ queries
- Shor/Simon Quantum Period Finding f:[N] = N takes O(log N) queries