Challenges in quantum algorithms for integer factorization

D. J. Bernstein

University of Illinois at Chicago

Prelude: What is the fastest algorithm to sort an array?

```
def blindsort(x):
  while not issorted(x):
    permuterandomly(x)
```

```
def bubblesort(x):
  for j in range(len(x)):
    for i in reversed(range(j)):
      x[i],x[i+1] = (
        min(x[i],x[i+1]),
        max(x[i],x[i+1])
      )
```

`bubblesort` takes poly time.
$\Theta(n^2)$ comparisons.
Huge speedup over `blindsort`!

Is this the end of the story?

```
def bubblesort(x):
  for j in range(len(x)):
    for i in reversed(range(j)):
      x[i],x[i+1] = (
        min(x[i],x[i+1]),
        max(x[i],x[i+1])
      )
```

`bubblesort` takes poly time.
$\Theta(n^2)$ comparisons.
Huge speedup over `blindsort`!

Is this the end of the story?
No, still not optimal.

Analogous: What is the fastest algorithm to factor integers?

Shor's algorithm takes poly time. Huge speedup over NFS!

$b^2(\log b)^{1+o(1)}$ qubit operations to factor b-bit integer, using standard subroutines for fast integer arithmetic.

Is this the end of the story?

Analogous: What is the fastest algorithm to factor integers?

Shor's algorithm takes poly time. Huge speedup over NFS!

$b^2(\log b)^{1+o(1)}$ qubit operations to factor b-bit integer, using standard subroutines for fast integer arithmetic.

Is this the end of the story? No, still not optimal.

"Shor's algorithm: the bubble sort of integer factorization."

A simple exercise to illustrate suboptimality of Shor's algorithm: Find a prime divisor of $\lfloor 10^{3009}\pi \rfloor$.

31415926535897932384626433832795028841971693993751058209749445923078164062862089
98628034825342117067982148086513282306647093844609550582231725359408128481117450
28410270193852110555964462294895493038196442881097566593344612847564823378678316
52712019091456485669234603486104543266482133936072602491412737245870066063155881
74881520920962829254091715364367892590360011330530548820466521384146951941511609
43305727036575959195309218611738193261179310511854807446237996274956735188575272
48912279381830119491298336733624406566430860213949463952247371907021798609437027
70539217176293176752384674818467669405132000568127145263560827785771342757789609
17363717872146844090122495343014654958537105079227968925892354201995611212902196
08640344181598136297747713099605187072113499999983729780499510597317328160963185
95024459455346908302642522308253344685035261931188171010003137838752886587533208
38142061717766914730359825349042875546873115956286388235378759375195778185778053
21712268066130019278766111959092164201989380952572010654858632788659361533818279
68230301952035301852968995773622599413891249721775283479131515574857242454150695
95082953311686172785588907509838175463746493931925506040092770167113900984882401
28583616035637076601047101819429555961989467678374494482553797747268471040475346
46208046684259069491293313677028989152104752162056966024058038150193511253382430
03558764024749647326391419927260426992279678235478163600934172164121992458631503
02861829745557067498385054945885869269956909272107975093029553211653449872027559
60236480665499119881834797753566369807426542527862551818417574672890977772793800
08164706001614524919217321721477235014144197356854816136115735255213347574184946
84385233239073941433345477624168625189835694855620992192221842725502542568876717
90494601653466804988627232791786085784383827967976681454100953883786360950680064
22512520511739298489608412848862694560424196528502221066118630674427862203919494
50471237137869609563643719172874677646575739624138908658326459958133904780275900
99465764078951269468398352595709825822620522489407726719478268482601476990902640
13639443745530506820349625245174939965143142980919065925093722169646151570985838
74105978859597729754989301617539284681382686838689427741559918559252459539594310
49972524680845987273644695848653836736222626099124608051243884390451244136549762
78079771569143599770012961608944169486855584840635342207222582848864815845602850
60168427394522674676788952521385225499546667278239864565961163548862305774564980
35593634568174324112515076069479451096596094025228879710893145669136867228748940
56010150330861792868092087476091782493858900971490967598526136554978189312978482
16829989487226588048575640142704775551323796414515237462343645428584447952658678
21051141354735739523113427166102135969536231442952484937187110145765403590279934
40374200731057853906219838744780847848968332144571386875194350643021845319104848
10053706146806749192781911979399520614196634287544406437451237181921799983910159
19561814675142691239748940907186494231961567945208

Important variations in the factorization problem:

- Maybe need one factor.
- Maybe need all factors.
- Maybe factors are small.
- Maybe factors are large.
- Maybe there are many inputs.
- Maybe inputs in superposition.

Important variations in metrics (even assuming perfect devices):

- Qubits.
- Area ("A", including wire area).
- Qubit operations ("gates").
- Depth.
- Time ("T": latency).

Short-term RSA security

1995 Kitaev, 1996 Vedral–Barenco–Ekert, 1996 Beckman–Chari–Devabhaktuni–Preskill, 1998 Zalka, 1999 Mosca–Ekert, 2000 Parker–Plenio, 2001 Seifert, 2002 Kitaev–Shen–Vyalyi, 2003 Beauregard, 2006 Takahashi–Kunihiro, 2010 Ahmadi–Chiang, 2014 Svore–Hastings–Freedman, 2015 Grosshans–Lawson–Morain–Smith, 2016 Häner–Roetteler–Svore, 2017 Ekerå–Håstad, 2017 Johnston: try to squeeze constant factors out of Shor's algorithm.

2003 Beauregard: $2b + 3$ qubits.

… 2016 Häner–Roetteler–Svore: $2b + 2$ qubits; $64b^3(\lg b + O(1))$ Toffoli gates; similar number of CNOT gates; depth $O(b^3)$.

2003 Beauregard: $2b + 3$ qubits.

... 2016 Häner–Roetteler–Svore: $2b + 2$ qubits; $64b^3(\lg b + O(1))$ Toffoli gates; similar number of CNOT gates; depth $O(b^3)$.

Conventional wisdom: cannot avoid $2b$ qubits for controlled mulmod.

e.g. 4096 qubits for $b = 2048$, very common RSA key size.

So 2048-bit factorization needs 4096 qubits?

2003 Beauregard: $2b + 3$ qubits.
… 2016 Häner–Roetteler–Svore: $2b + 2$ qubits; $64b^3(\lg b + O(1))$ Toffoli gates; similar number of CNOT gates; depth $O(b^3)$.

Conventional wisdom: cannot avoid $2b$ qubits for controlled mulmod.

e.g. 4096 qubits for $b = 2048$, very common RSA key size.

So 2048-bit factorization needs 4096 qubits?
No: NFS uses 0 qubits.

NFS takes $L^{p+o(1)}$ operations with $p = \sqrt[3]{92 + 26\sqrt{13}}/3 > 1.9$, $\log L = (\log 2^b)^{1/3}(\log\log 2^b)^{2/3}$.

Analysis for $b = 2048$ (not easy!): very roughly 2^{112} operations.

NFS takes $L^{p+o(1)}$ operations with $p = \sqrt[3]{92 + 26\sqrt{13}}/3 > 1.9$, $\log L = (\log 2^b)^{1/3}(\log\log 2^b)^{2/3}$.

Analysis for $b = 2048$ (not easy!): very roughly 2^{112} operations.

2017 Bernstein–Biasse–Mosca: $L^{q+o(1)}$ operations with $q = \sqrt[3]{8/3} \approx 1.387$, using $b^{2/3+o(1)}$ qubits (and many non-quantum bits).

NFS takes $L^{p+o(1)}$ operations
with $p = \sqrt[3]{92 + 26\sqrt{13}}/3 > 1.9$,
$\log L = (\log 2^b)^{1/3}(\log\log 2^b)^{2/3}$.

Analysis for $b = 2048$ (not easy!):
very roughly 2^{112} operations.

2017 Bernstein–Biasse–Mosca:
$L^{q+o(1)}$ operations
with $q = \sqrt[3]{8/3} \approx 1.387$,
using $b^{2/3+o(1)}$ qubits
(and many non-quantum bits).

Open: Analyze for $b = 2048$.
Fewer than 4096 qubits?
Fewer than 2048 qubits?

Counting operations is an oversimplified cost model: ignores communication costs, parallelism. See, e.g., 1981 Brent–Kung *AT* theorem for realistic chip model.

Counting operations is an oversimplified cost model: ignores communication costs, parallelism. See, e.g., 1981 Brent–Kung AT theorem for realistic chip model.

NFS suffers somewhat from communication costs inside big linear-algebra subroutine.

2001 Bernstein:
$AT = L^{p'+o(1)}$ with $p' \approx 1.976$.

2017 Bernstein–Biasse–Mosca:
$AT = L^{q'+o(1)}$ with $q' \approx 1.456$ using $b^{2/3+o(1)}$ qubits.
Open: Analyze for $b = 2048$.

Actually have many inputs.

Lower cost for *some* output?

Lower cost for *many* outputs?

Actually have many inputs.

Lower cost for *some* output?

Lower cost for *many* outputs?

1993 Coppersmith:
$L^{1.638\ldots+o(1)}$ operations
after precomp(b) involving
$L^{2.006\ldots+o(1)}$ operations.

Actually have many inputs.
Lower cost for *some* output?
Lower cost for *many* outputs?

1993 Coppersmith:
$L^{1.638...+o(1)}$ operations
after precomp(b) involving
$L^{2.006...+o(1)}$ operations.

2014 Bernstein–Lange:
$AT = L^{2.204...+o(1)}$
to factor $L^{0.5+o(1)}$ inputs;
$L^{1.704...+o(1)}$ per input.

Actually have many inputs.
Lower cost for *some* output?
Lower cost for *many* outputs?

1993 Coppersmith:
$L^{1.638\ldots+o(1)}$ operations
after precomp(b) involving
$L^{2.006\ldots+o(1)}$ operations.

2014 Bernstein–Lange:
$AT = L^{2.204\ldots+o(1)}$
to factor $L^{0.5+o(1)}$ inputs;
$L^{1.704\ldots+o(1)}$ per input.

Open: Any quantum speedups
for factoring many integers?

Long-term RSA security

Long history of advances in integer factorization.

Long history of RSA users switching to larger key sizes, not far beyond broken sizes.

Long-term RSA security

Long history of advances in integer factorization.

Long history of RSA users switching to larger key sizes, not far beyond broken sizes.

"Expert" cryptographers: "Obviously they won't react to Shor's algorithm this way! They'll switch to codes, lattices, etc. long before quantum computers break RSA-2048! We don't need to analyze the security of RSA-4096, RSA-8192, RSA-16384, etc.!"

We consider possible impact of quantum computers. Shouldn't we also consider possible impact of users wanting to stick to RSA?

We consider possible impact of quantum computers. Shouldn't we also consider possible impact of users wanting to stick to RSA?

2017 Bernstein–Heninger–Lou–Valenta "Post-quantum RSA" (pqRSA): Generated 1-terabyte RSA key; 2000000 core-hours. Shor's algorithm: $>2^{100}$ gates.

We consider possible impact of quantum computers. Shouldn't we also consider possible impact of users wanting to stick to RSA?

2017 Bernstein–Heninger–Lou–Valenta "Post-quantum RSA" (pqRSA): Generated 1-terabyte RSA key; 2000000 core-hours. Shor's algorithm: $>2^{100}$ gates.

Bernstein–Fried–Heninger–Lou–Valenta: Draft NIST submission proposing 1-gigabyte RSA keys. Much faster to generate.

The secret primes are small:
4096 bits in terabyte key;
1024 bits in gigabyte key.
Important time-saver in keygen, signing, decryption.

Is this a weakness?

ECM finds any prime $<y$ using $L^{\sqrt{2}+o(1)}$ mulmods, where $\log L = (\log y \log\log y)^{1/2}$.
Beats Shor for $\log y$ below $(\log\log \text{modulus})^{2+o(1)}$.

Public ECM record:
274-bit factor of $7^{337} + 1$.

Analysis for $y \approx 2^{1024}$:
$>2^{125}$ mulmods, huge depth;
and 2^{33}-bit mulmod is slow.

2^{23} target primes, but
finding just one isn't enough.

Analysis for $y \approx 2^{1024}$:
$>2^{125}$ mulmods, huge depth;
and 2^{33}-bit mulmod is slow.

2^{23} target primes, but
finding just one isn't enough.

2017 Bernstein–Heninger–Lou–Valenta: Grover+ECM
finds any prime $<y$
using $L^{1+o(1)}$ mulmods.

Analysis for $y \approx 2^{1024}$:
$>2^{125}$ mulmods, huge depth;
and 2^{33}-bit mulmod is slow.

2^{23} target primes, but
finding just one isn't enough.

2017 Bernstein–Heninger–Lou–Valenta: Grover+ECM
finds any prime $<y$
using $L^{1+o(1)}$ mulmods.

Seems swamped by overhead.

Open: Better ways for quantum algorithms to find small factors?

Minimum security level that NIST allows for post-quantum submissions: brute-force/Grover search for a 128-bit AES key.

Is a gigabyte key so difficult for Shor's algorithm to break?

Minimum security level that NIST allows for post-quantum submissions: brute-force/Grover search for a 128-bit AES key.

Is a gigabyte key so difficult for Shor's algorithm to break?

$64b^3 \lg b \approx 2^{110}$ for $b = 2^{33}$.

Not totally implausible to argue that Grover's algorithm could break AES-128 faster than this.

Minimum security level that NIST allows for post-quantum submissions: brute-force/Grover search for a 128-bit AES key.

Is a gigabyte key so difficult for Shor's algorithm to break?

$64b^3 \lg b \approx 2^{110}$ for $b = 2^{33}$.

Not totally implausible to argue that Grover's algorithm could break AES-128 faster than this.

But Shor's algorithm can (with more qubits) use faster mulmods.

NIST allows submissions to assume reasonable time limits:

"Plausible values for MAXDEPTH range from 2^{40} logical gates (the approximate number of gates that presently envisioned quantum computing architectures are expected to serially perform in a year) through 2^{64} logical gates (the approximate number of gates that current classical computing architectures can perform serially in a decade), to no more than 2^{96} logical gates . . ."

What is the minimum time for b-bit integer multiplication?

Light takes time $\Omega(b^{1/2})$ to cross a $b^{1/2} \times b^{1/2}$ chip.

1981 Brent–Kung AT theorem: $AT \geq \text{small constant} \cdot b^{3/2}$, even if wire latency is 0.

(Work around obstacles using faster-than-light communication through long-distance EPR pairs? Haven't seen plausible designs, even if reversible computation avoids FTL impossibility proofs.)

What is the minimum time for Shor's algorithm?

Main bottleneck: $a^e \bmod N$ for $2b$-bit superposition e.

Traditional approach: series of controlled multiplications by a and $1/a \bmod N$; $a^2 \bmod N$ and $1/a^2 \bmod N$; $a^4 \bmod N$ and $1/a^4 \bmod N$; etc.

Can multiply these in parallel, using many more qubits; but hard to parallelize initial computation of $a^{2^i} \bmod N$.

Why gigabyte keys are reasonable:
big enough to push latency
beyond the 2^{64} limit,
under reasonable assumptions.

Gigabyte inputs are
millions of times larger
than 2048-bit inputs.
These algorithms will take
billions of times longer.
More cost to find *all* primes.

Why gigabyte keys are reasonable:
big enough to push latency
beyond the 2^{64} limit,
under reasonable assumptions.

Gigabyte inputs are
millions of times larger
than 2048-bit inputs.
These algorithms will take
billions of times longer.
More cost to find *all* primes.

Open: What is minimum time
for integer factorization?

NIST's middle security level
is defined by an AES-192 key.

NIST's middle security level is defined by an AES-192 key.

With maximum depth 2^{64}, finding an AES-192 key requires $\approx 2^{144}$ cores.

NIST's middle security level is defined by an AES-192 key.

With maximum depth 2^{64}, finding an AES-192 key requires $\approx 2^{144}$ cores.

This is nonsense! There is not enough time to broadcast the input to 2^{144} parallel computations, and not enough time to collect the results.

NIST's middle security level is defined by an AES-192 key.

With maximum depth 2^{64}, finding an AES-192 key requires $\approx 2^{144}$ cores.

This is nonsense! There is not enough time to broadcast the input to 2^{144} parallel computations, and not enough time to collect the results.

Is NIST implicitly assuming a higher latency limit?

Some improvements to Shor

(2017 Bernstein–Biasse–Mosca)

Consider Shor's algorithm factoring $N = p_1^{e_1} \cdots p_f^{e_f}$. Write $(p_j - 1)p_j^{e_j - 1}$ as $2^{t_j} u_j$ with u_j odd.

Unit group is isomorphic to $\mathbf{Z}/2^{t_1} \times \cdots \times \mathbf{Z}/2^{t_f} \times \mathbf{Z}/u_1 \times \cdots$.

Some improvements to Shor

(2017 Bernstein–Biasse–Mosca)

Consider Shor's algorithm factoring $N = p_1^{e_1} \cdots p_f^{e_f}$. Write $(p_j - 1)p_j^{e_j - 1}$ as $2^{t_j} u_j$ with u_j odd.

Unit group is isomorphic to $\mathbf{Z}/2^{t_1} \times \cdots \times \mathbf{Z}/2^{t_f} \times \mathbf{Z}/u_1 \times \cdots$.

Shor's algorithm (hopefully) computes order r of random unit. Order 2^{c_j} in $\mathbf{Z}/2^{t_j}$ is
2^{t_j} with probability $1/2$;
$2^{t_j - 1}$ with probability $1/4$; etc.

Shor computes $\gcd\{N, a^{r/2} - 1\}$.
Divisible by p_j exactly when
$c_j < \max\{c_1, \ldots, c_f\}$.

Factorization fails iff all c_j are
equal. Chance $\leq 1/2^{f-1}$.

Shor computes $\gcd\{N, a^{r/2} - 1\}$.
Divisible by p_j exactly when
$c_j < \max\{c_1, \ldots, c_f\}$.

Factorization fails iff all c_j are equal. Chance $\le 1/2^{f-1}$.

More subtle problem:
Factorization is likely to split off some of the primes with maximum t_j.

Can iterate Shor's algorithm enough times to completely factor. Many full-size iterations; many more for adversarial inputs.

Better method, inspired by primality testing: compute gcd with $a^{r/2}+1$, $a^{r/4}+1$, $a^{r/8}+1$, $\ldots$, a^d+1, a^d-1, with odd d.

This splits p_j according to c_j. Any two primes have chance $\geq 1/2$ of being split.

Factors are around half size. Much less overhead for recursion.

Also "parallel construction": Run several times in parallel, giving several factorizations. Then factor into coprimes.

These methods use $>b$ qubits.
Didn't we claim $b^{2/3+o(1)}$ qubits?

We actually use Grover's method to search for smooth $b^{2/3+o(1)}$-bit numbers in NFS.

Oracle for Grover's method: factor thoroughly enough to recognize smooth inputs.

We tweak (improved) Shor to work in superposition. Careful with qubit budget for continued fractions, power detection, etc.

A different way to improve randomness of factorizations in Shor's algorithm: replace group $(\mathbf{Z}/N)^*$ with $E(\mathbf{Z}/N)$ for a random elliptic curve E.

A different way to improve randomness of factorizations in Shor's algorithm: replace group $(\mathbf{Z}/N)^*$ with $E(\mathbf{Z}/N)$
for a random elliptic curve E.

Gal Dor suggests unifying Grover+ECM with Shor: e.g., compute esP on $E(\mathbf{Z}/N)$ where
e is superposition of scalars,
s is smooth scalar,
E is superposition of curves.

A different way to improve randomness of factorizations in Shor's algorithm: replace group $(\mathbf{Z}/N)^*$ with $E(\mathbf{Z}/N)$ for a random elliptic curve E.

Gal Dor suggests unifying Grover+ECM with Shor: e.g., compute esP on $E(\mathbf{Z}/N)$ where e is superposition of scalars, s is smooth scalar, E is superposition of curves.

Open: What are minimum costs for this unification?