Speed-ups for Quantum Algorithms with Easier Inputs

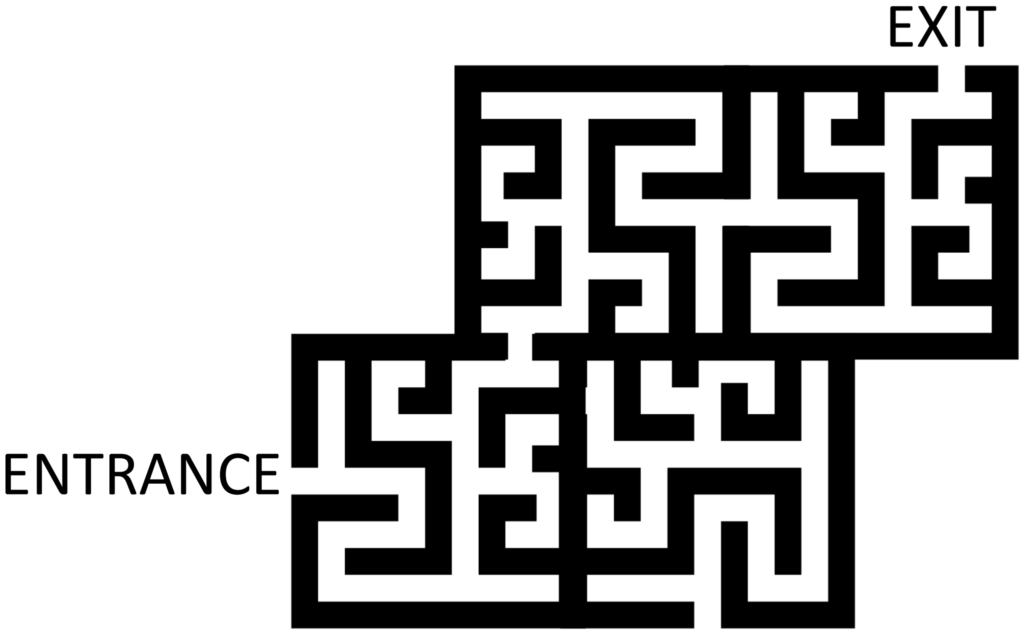
Shelby Kimmel, Jay-U Chung, Noel Anderson

Middlebury College

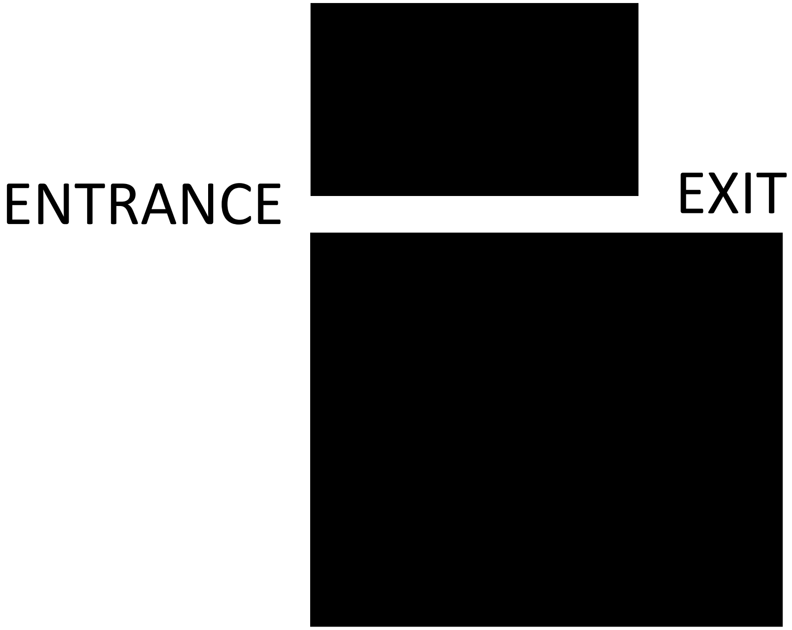
Easy vs Hard Instances

Path-Detection:

Harder:



Easier:



Easy vs Hard Instances: Classically

Ex: Path-Detection

Run search from ENTRANCE for time T (based on size of maze).

- If find EXIT, stop and output *YES,* otherwise continue
- If after time T don't find EXIT, output *NO*

Easy vs Hard Instances: Classically

Ex: Path-Detection

Run search from ENTRANCE for time T (based on size of maze).

- If find EXIT, stop and output *YES,* otherwise continue
- If after time T don't find EXIT, output *NO*

If easier: shorter run time

If harder: longer run time

Easy vs Hard Instances: Classically

Ex: Path-Detection

Run search from ENTRANCE for time T (based on size of maze).

- If find EXIT, stop and output *YES,* otherwise continue
- If after time T don't find EXIT, output *NO*

If easier: shorter run time
If harder: longer run time

Key properties:

1. Can check status mid-algorithm and continue running
2. Witness of completion (if find EXIT, convinced there is a path)

Easy vs Hard Instances: Quantumly

Run search from ENTRANCE time ased on size of maze).

- If find EXIT, stop and output *YES*, ot wise continue
- If after time T don't find EXIT, ou *NO*

If easier: shorter run time
If harder: longer run time

Key properties:

1. Can check status mid-algorithm and continue running
2. Witness of completion (if find EXIT, convinced there is a path)

Our Result

For a large class of quantum algorithms that previously used worst-case runtime for all instances:

Our Result

For a large class of quantum algorithms that previously used worst-case runtime for all instances:

Create a modified algorithm:

- If worst-case instance: (approximately) previous worst case run time
- If easier instance: shorter run time

Talk Outline

1. Oracle Model
2. Challenges:
    a. Can’t check property of algorithm and then continue running ⟵ **Easy**
        i. Why challenge exists quantumly
        ii. How to overcome
    b. (Frequently) No (easily accessible) witness of completion ⟵ **Harder**
        i. Why challenge exists quantumly
        ii. How to overcome
3. Applications & Future Work

Oracle Model

Given:

- Description of a Boolean function f
- Set X of possible instances

$\Rightarrow$

Design an algorithm to decide any instance in X

Oracle Model

Is there a path?

Given:
- Description of a Boolean function f
- Set X of possible instances

Graph/maze

Design an algorithm to decide any instance in X

Oracle Model

Given:
- Description of a Boolean function f
- Set X of possible instances

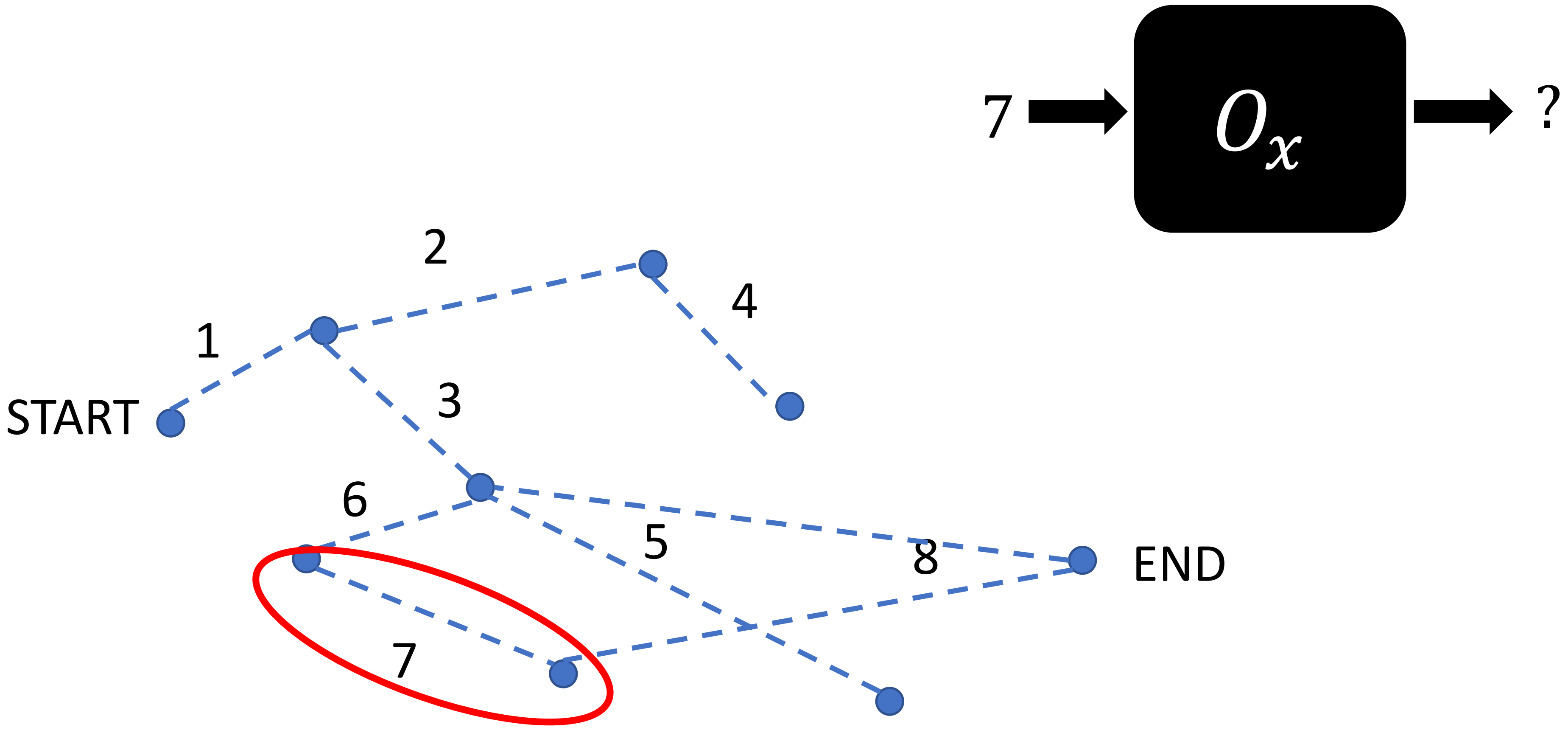
Design an algorithm to decide any instance in X

Input
O_x for specific instance $x \in X$

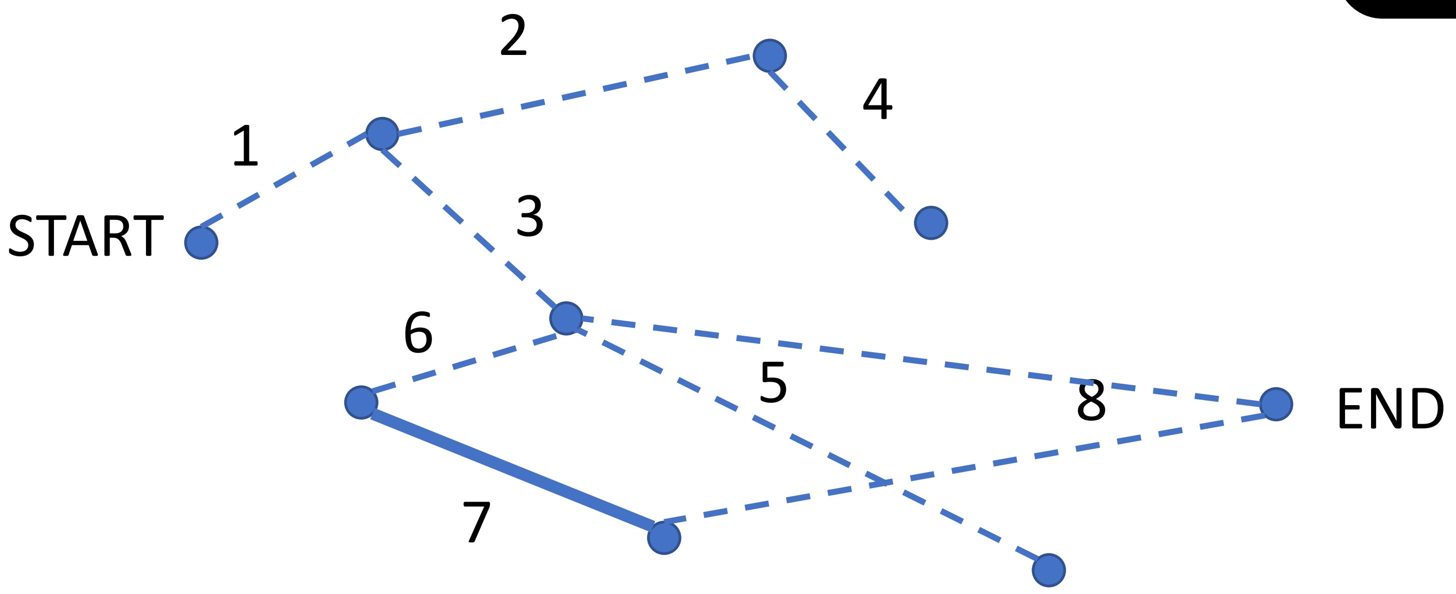
$$i \longrightarrow \boxed{O_x} \longrightarrow x_i$$

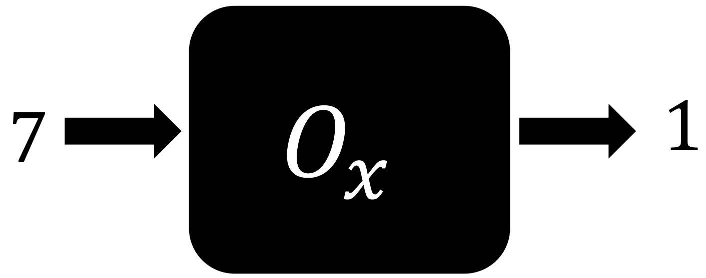
Output:
$f(x)$, using as few queries as possible
…in worst case
…while using fewer queries on easier instances

Oracle Model

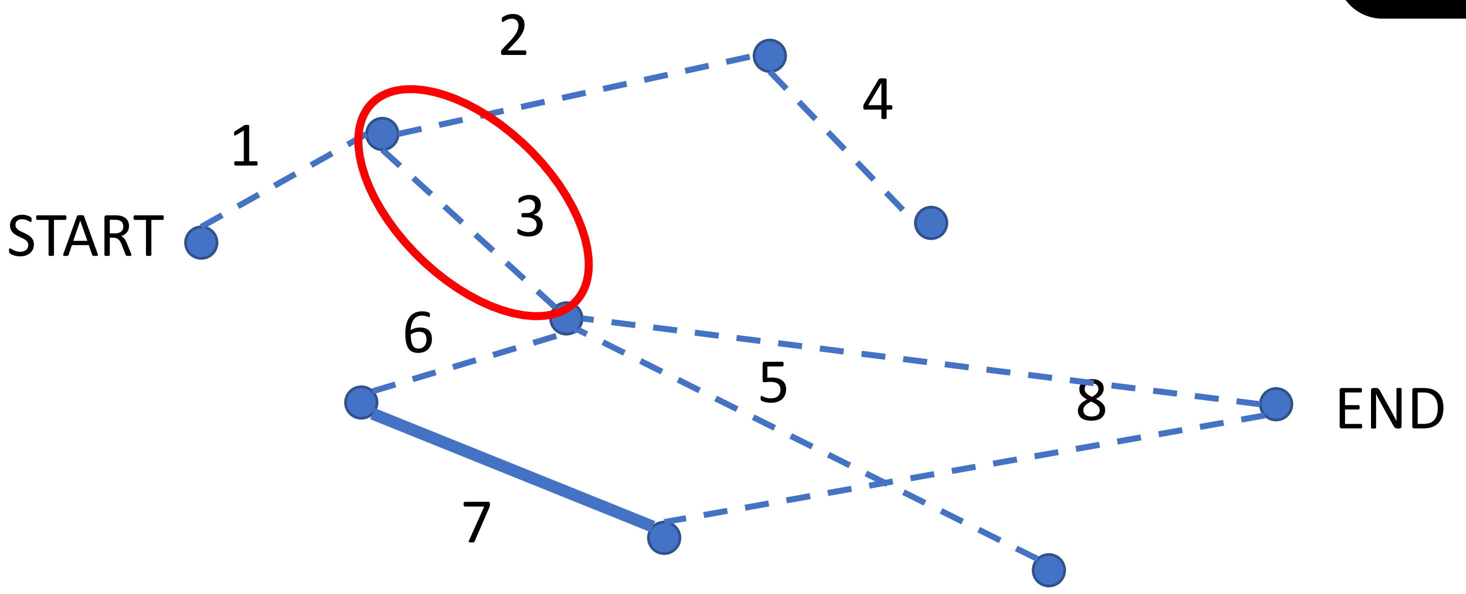


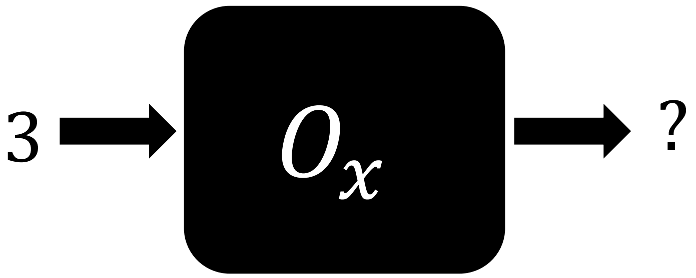
Oracle Model



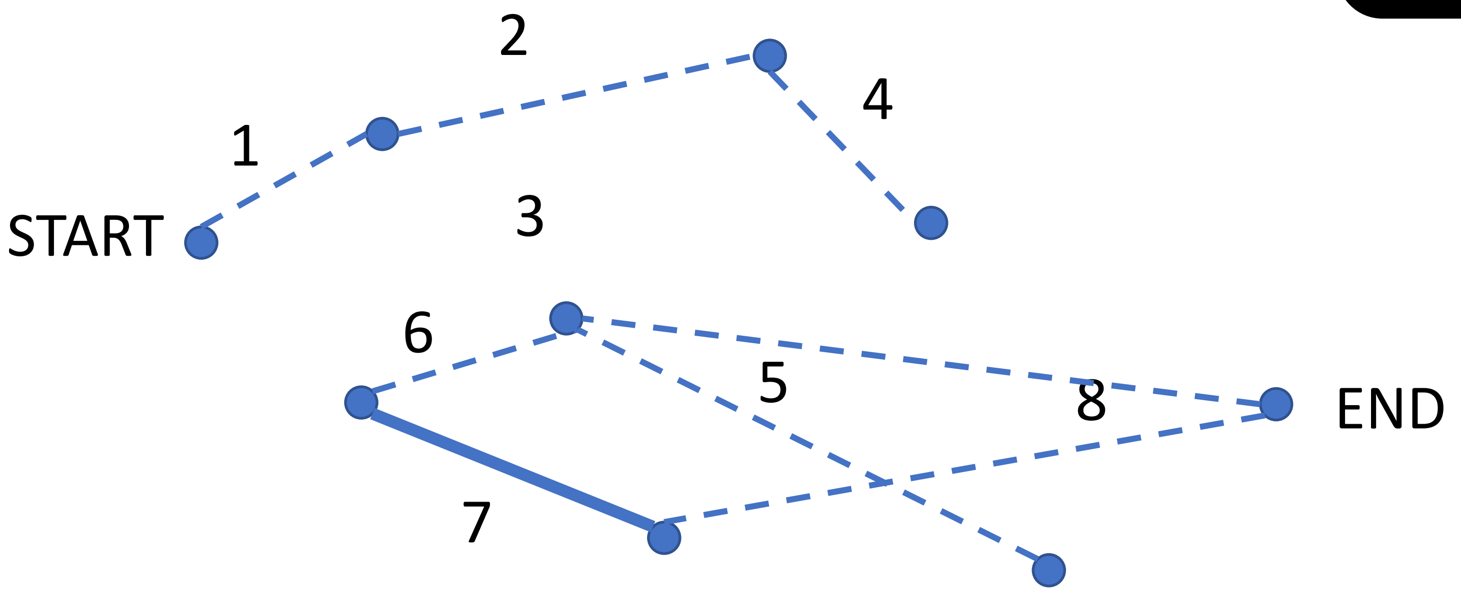


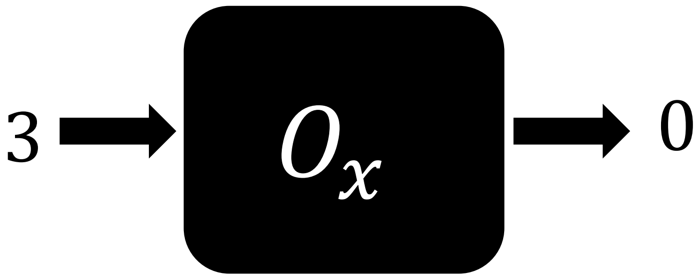
Oracle Model





Oracle Model





Quantum Oracle Model

Given:

- Description of a Boolean function f
- Set X of possible instances

Design an quantum algorithm to decide any instance in X

Input

O_x for specific instance $x \in X$

$$|i\rangle|0\rangle \longrightarrow \boxed{O_x} \longrightarrow |i\rangle|x_i\rangle$$

Output:

$f(x)$, using as few queries as possible

…in worst case

…while using fewer queries on easier instances

# of queries – “runtime” – query complexity

Talk Outline

1. Oracle Model
2. Challenges:
   a. Can’t check property of algorithm and then continue running ← **Easy**
      i. Why challenge exists quantumly
      ii. How to overcome
   b. (Frequently) No (easily accessible) witness of completion ← **Harder**
      i. Why challenge exists quantumly
      ii. How to overcome
3. Applications & Future Work

If … Continue

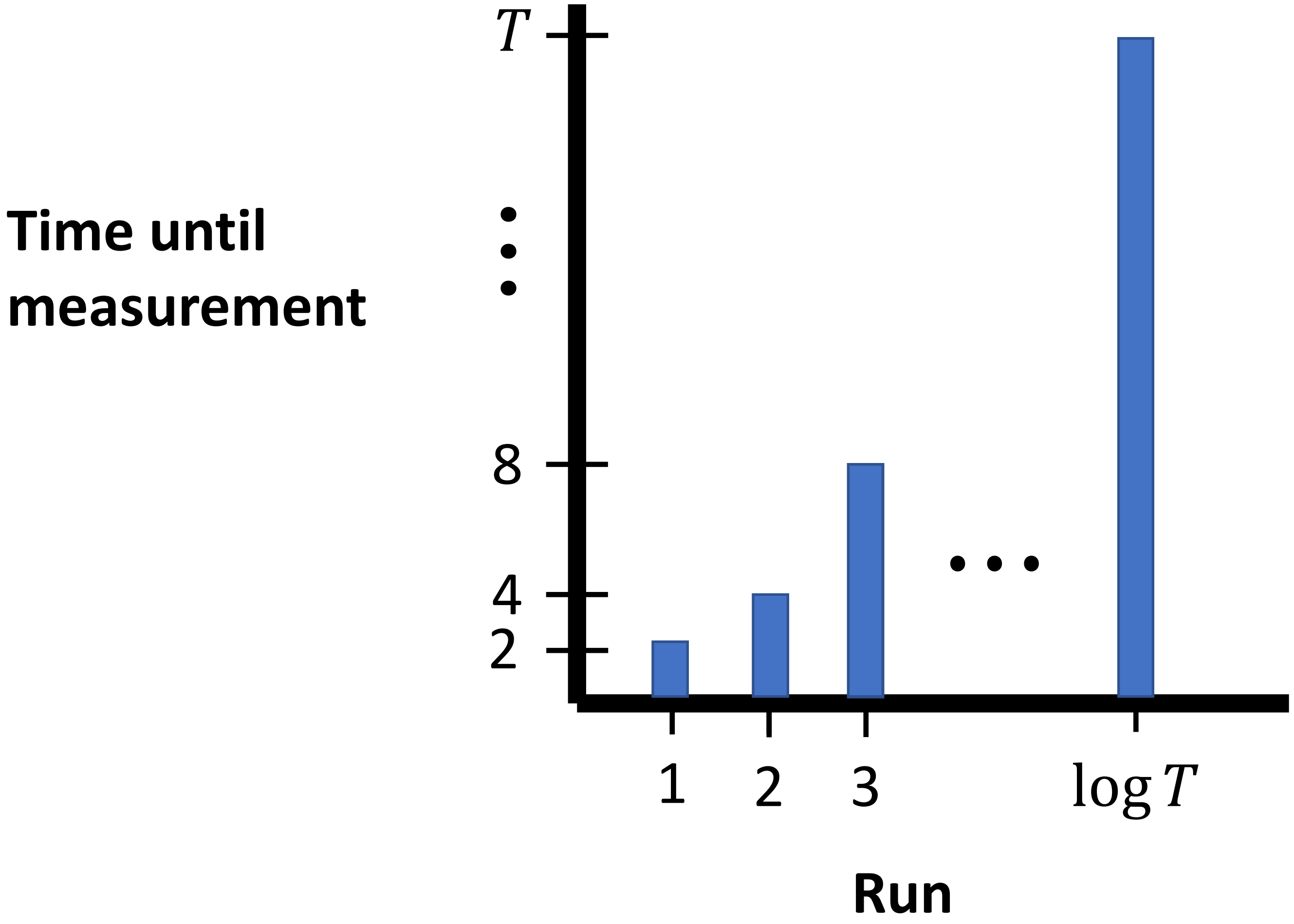
If … Continue

Classically: Can check property of algorithm and then continue running

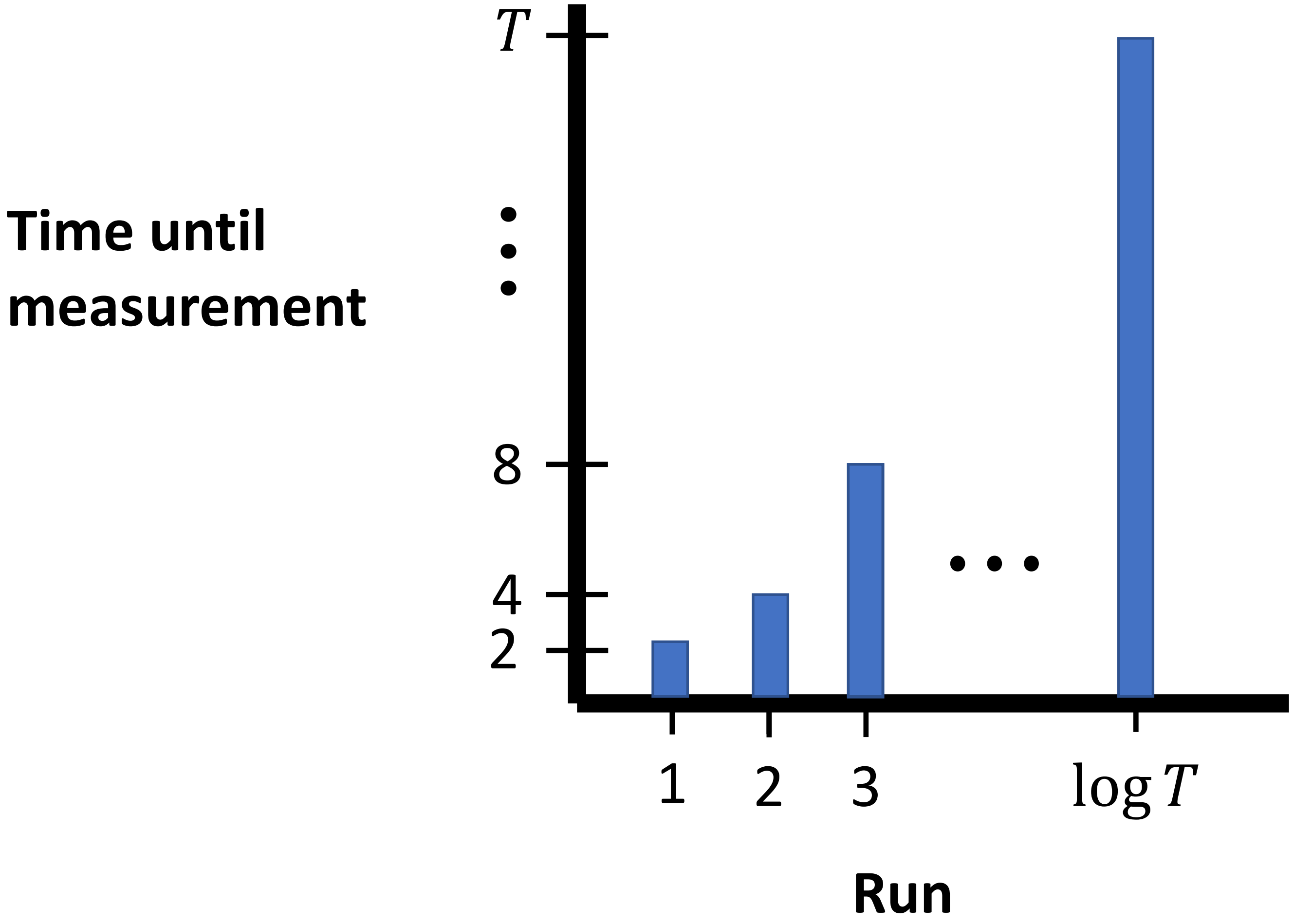
Quantumly: to check property, need to measure

- Measurement $\rightarrow$ collapse
- Can't continue

If … Continue



If … Continue



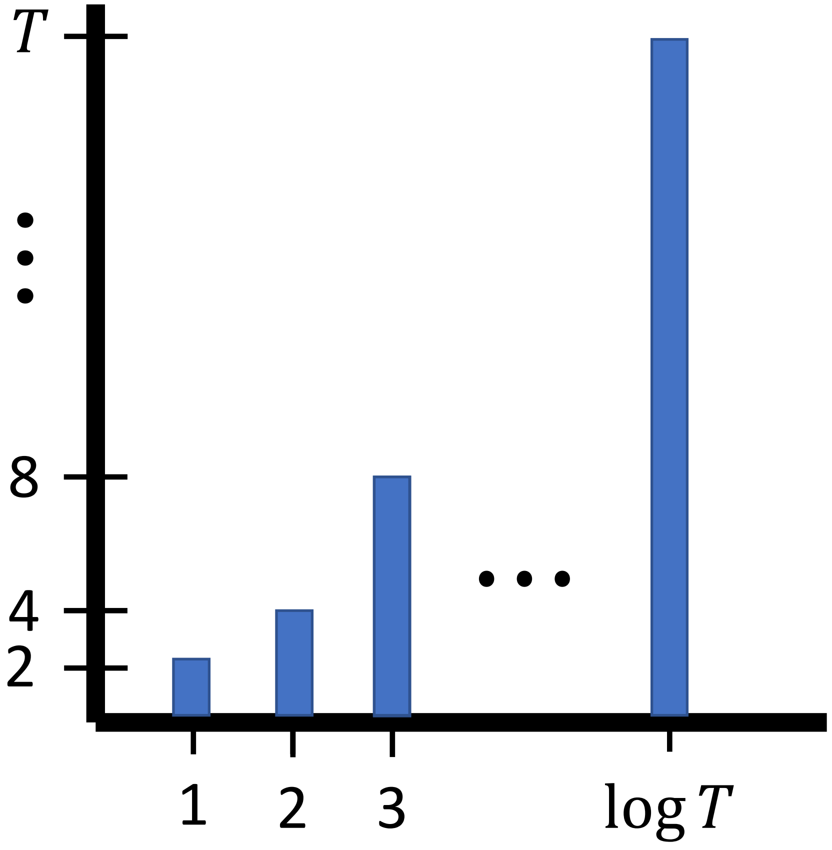
Total Runtime:

$O(T)$

(Geometric series)

If … Continue

Time until measurement



Run

Total Runtime:

$O(T)$

(Geometric series)

Runtime with Error Reduction:

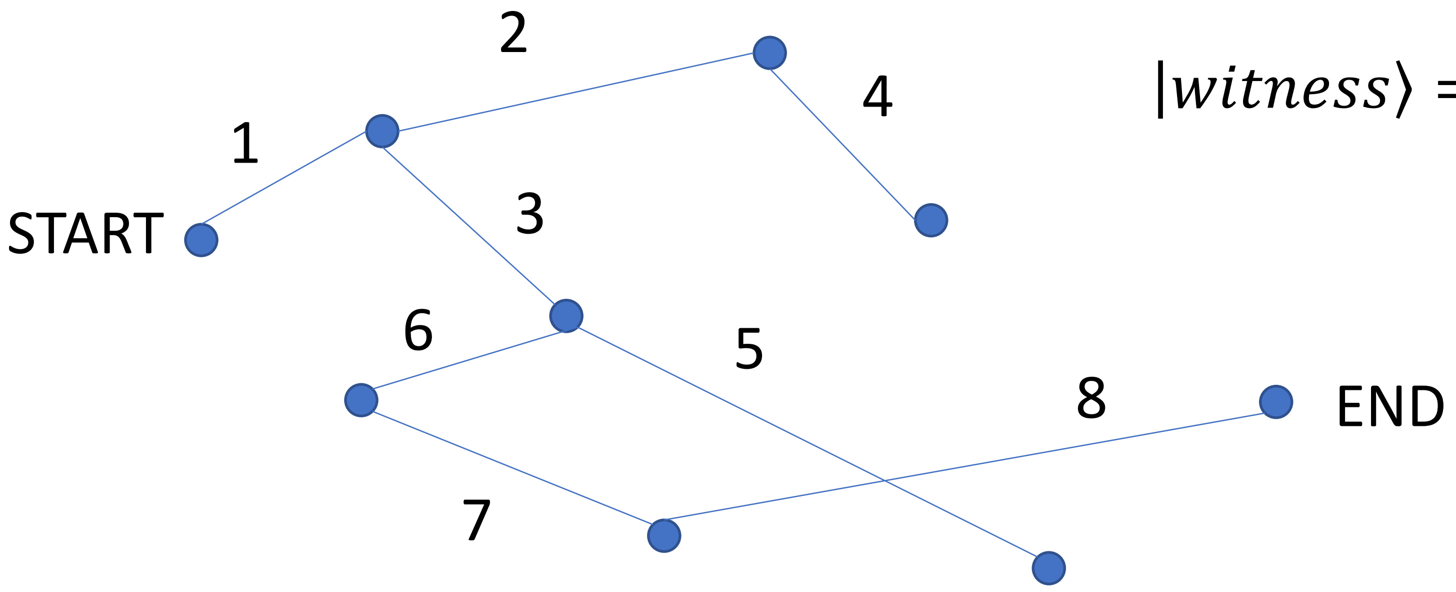
$\tilde{O}(T)$

Talk Outline

1. Oracle Model
2. Challenges:
   a. Can't check property of algorithm and then continue running ⟵ **Easy** ☑
      i. Why challenge exists quantumly
      ii. How to overcome
   b. (Frequently) No (easily accessible) witness of completion ⟵ **Harder**
      i. Why challenge exists quantumly
      ii. How to overcome
3. Applications & Future Work

High Level Problem: Witness is a Hard to Characterize Quantum State

Yes Instance, run long enough: $|YES\rangle|witness\rangle$



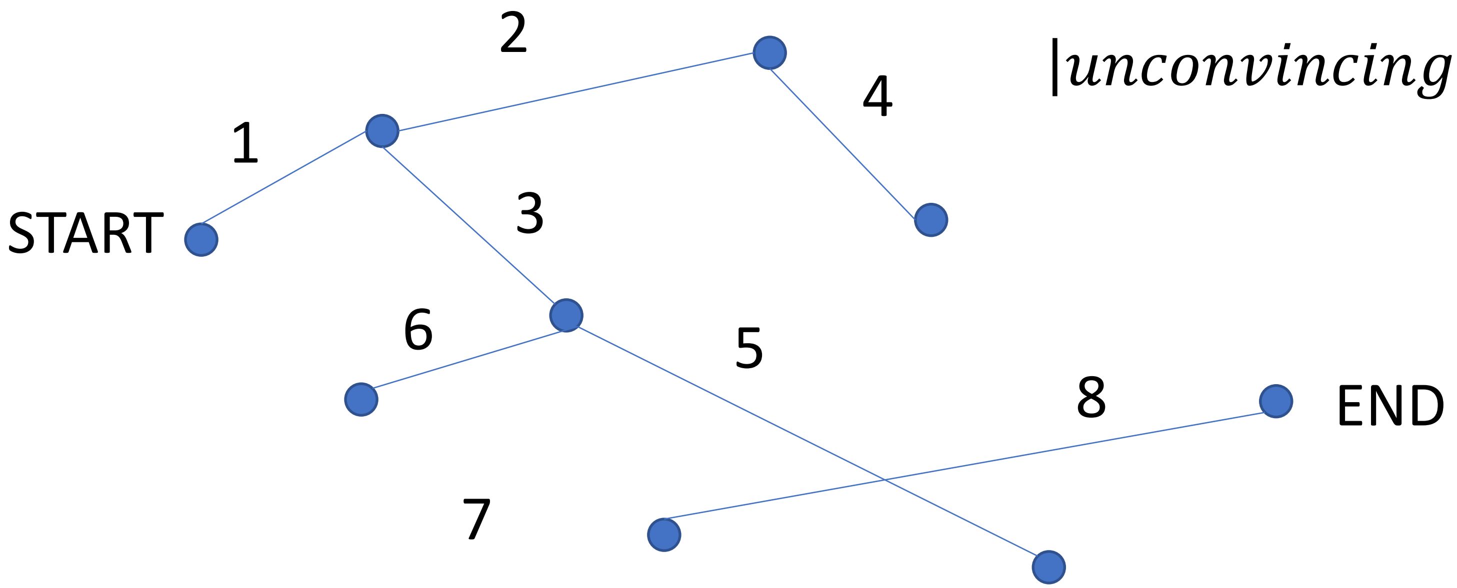
$$|witness\rangle = |1\rangle + |3\rangle + |6\rangle + |7\rangle + |8\rangle$$

High Level Problem: Witness is a Hard to Characterize Quantum State

False positive

No Instance, not run long enough: $|YES\rangle|unconvincing\ witness\rangle$

$$|unconvincing\ witness\rangle = |1\rangle - |3\rangle + |6\rangle - |8\rangle$$



Talk Outline

1. Oracle Model
2. Challenges:
   a. Can’t check property of algorithm and then continue running ⟵ **Easy** ☑
      i. Why challenge exists quantumly
      ii. How to overcome
   b. (Frequently) No (easily accessible) witness of completion ⟵ **Harder**
      i. Why challenge exists quantumly
      ii. How to overcome – avoid dealing with witness states
3. Applications & Future Work

Our Result

Span program algorithms

For a large class of quantum algorithms that previously used worst-case runtime for all instances:

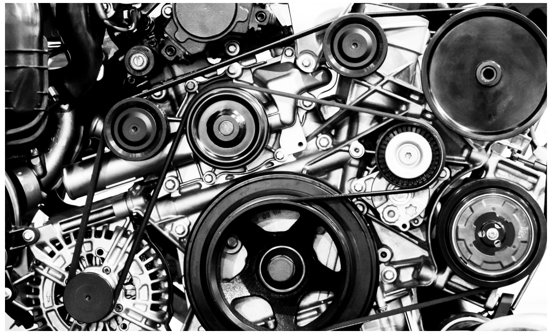
Create a modified algorithm:

- If worst-case instance: (approximately) previous worst case run time
- If easier instance: shorter run time

Span Program Algorithms

$$\begin{pmatrix}\vdots\\ \vdots\\ \vdots\end{pmatrix}\begin{pmatrix}\vdots\\ \vdots\\ \vdots\end{pmatrix}\cdots\begin{pmatrix}\vdots\\ \vdots\\ \vdots\end{pmatrix} \rightarrow$$

Encodes f on domain X



$\rightarrow$ Quantum Query algorithm for f on domain X

∀ functions, ∃ span program:

- Query optimal for worst-case (hardest) inputs
- Not known how to get a speed-up for easier instances*

*If don't know ahead of time that instance is easy

(See Reichardt 2010 https://arxiv.org/pdf/1005.1601.pdf)

Phase Estimation

Key procedure for span program algorithm

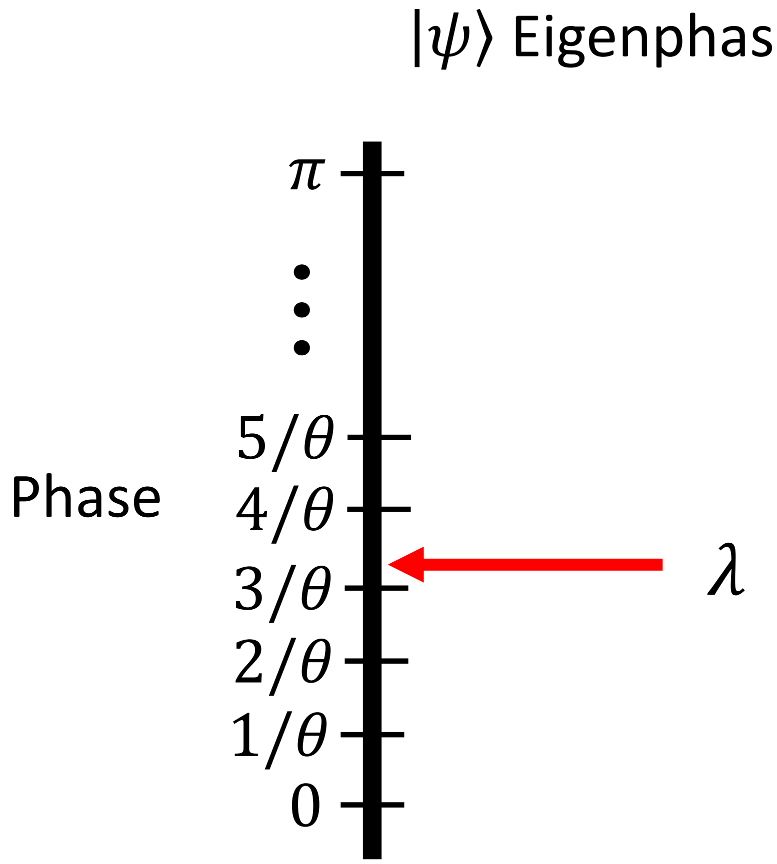
Input:

- Unitary U
- Eigenstate $|\psi\rangle$, s.t $U|\psi\rangle = e^{2\pi i\lambda}|\psi\rangle$
- Precision θ

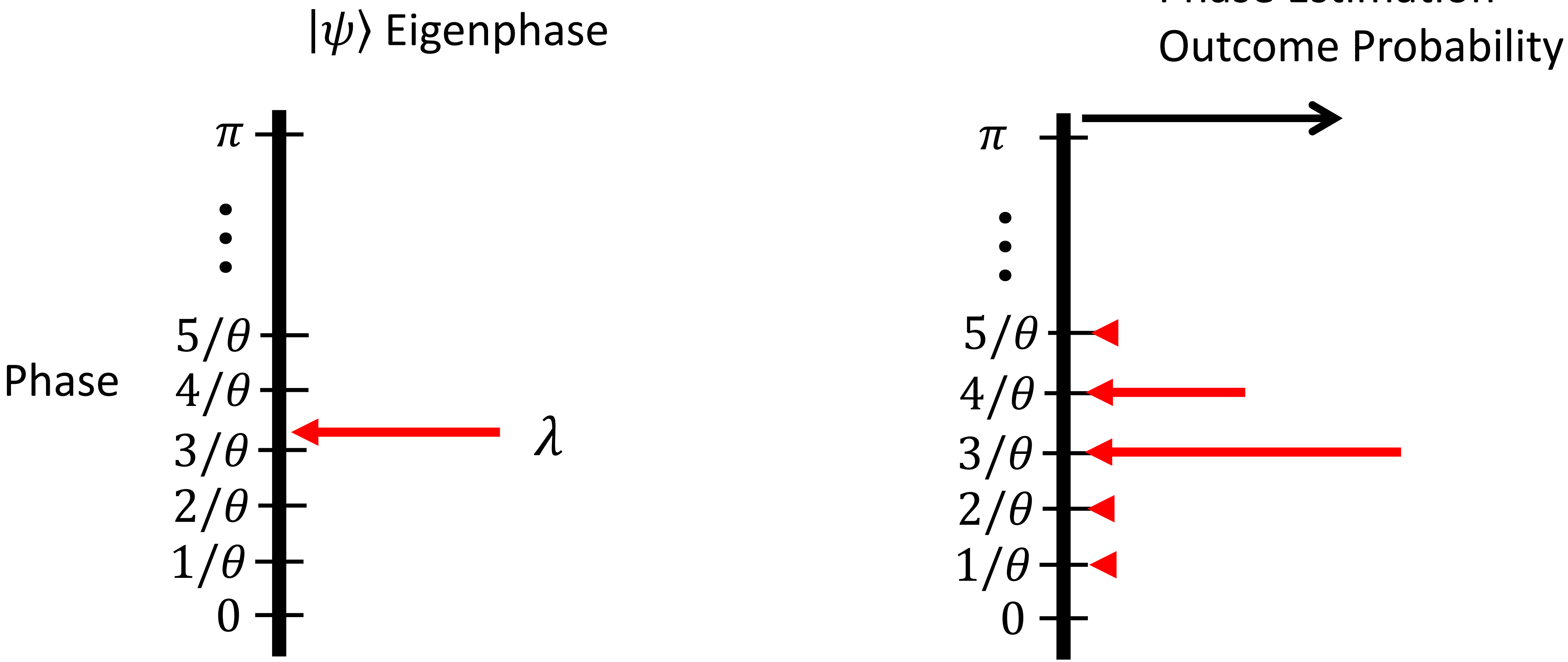
Output: $|\tilde{\lambda}|$ (approximation of $|\lambda|$ to precision θ),

(See Reichardt 2010 https://arxiv.org/pdf/1005.1601.pdf for 3 different algorithms!)

Phase Estimation



Phase Estimation



Phase Estimation for Span Programs

Span program for f on $X \rightarrow \exists$ unitary U (created using O_x), state $|\psi\rangle$ $s.t. \forall x \in X$:

Phase Estimation for Span Programs

Span program for f on $X \rightarrow \exists$ unitary U (created using O_x), state $|\psi\rangle$ $s.t. \forall x \in X$:

- If $f(x) = YES$,
  - Output phase is 0 w.h.p
- If $f(x) = NO$
  - Output phase is ***not*** 0 w.h.p, if use large enough θ (precision)

Phase Estimation for Span Programs

Span program for f on $X \rightarrow \exists$ unitary U (created using O_x), state $|\psi\rangle$ $s.t. \forall x \in X$:

- If $f(x) = YES$,
  - Output phase is 0 w.h.p
- If $f(x) = NO$
  - Output phase is ***not*** 0 w.h.p, if use large enough θ (precision)

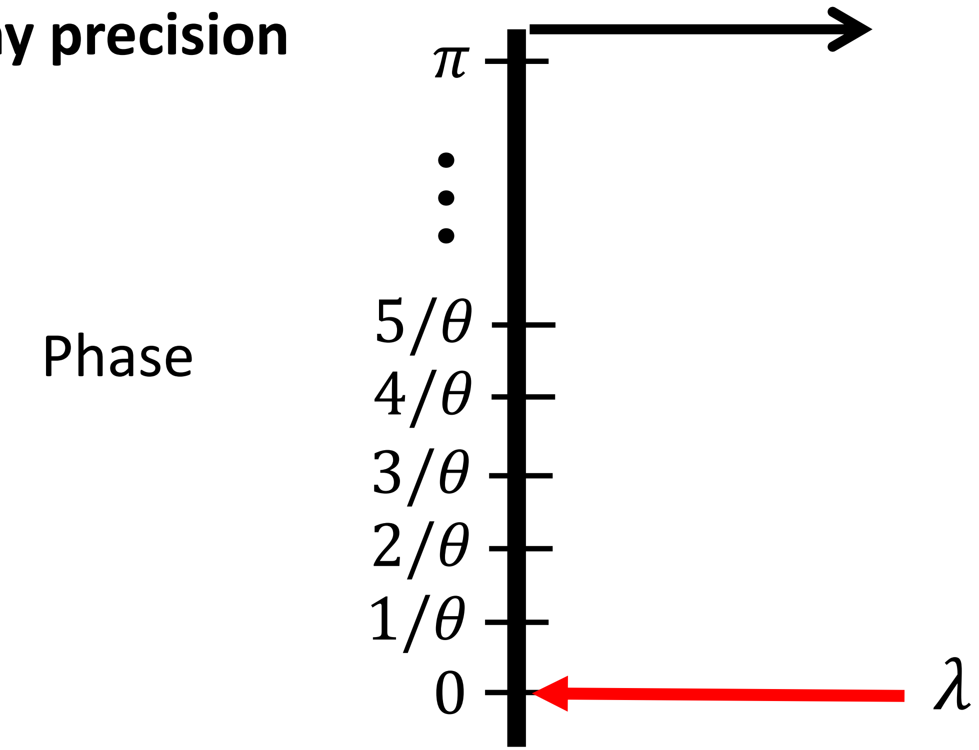
Larger θ $\Rightarrow$ Longer runtime

Reduce Precision (θ)

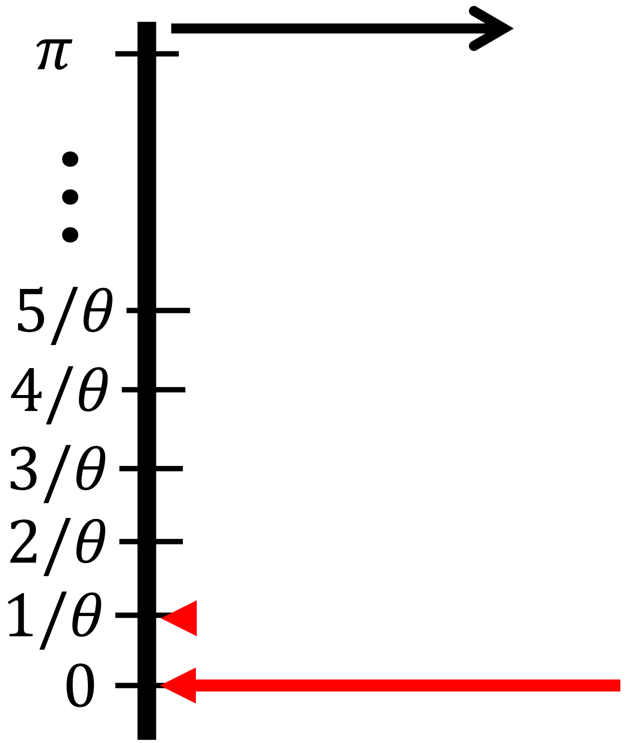
Case 0:

- **YES INSTANCE**
- **Any precision**

$|\psi\rangle$ Eigenphase



Phase estimation
Outcome Probability

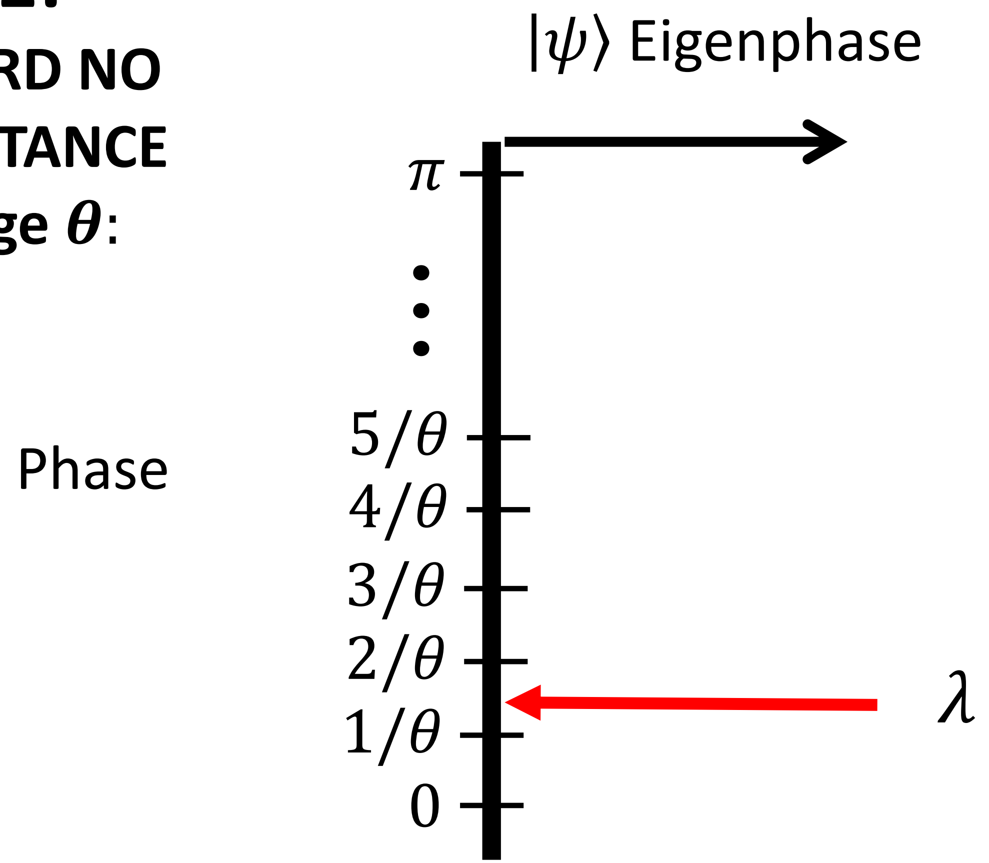


YES Instance:
Want high probability of outcome 0

Reduce Precision (θ)

Case 1:

- **HARD NO INSTANCE**
- **Large $\boldsymbol{\theta}$**:



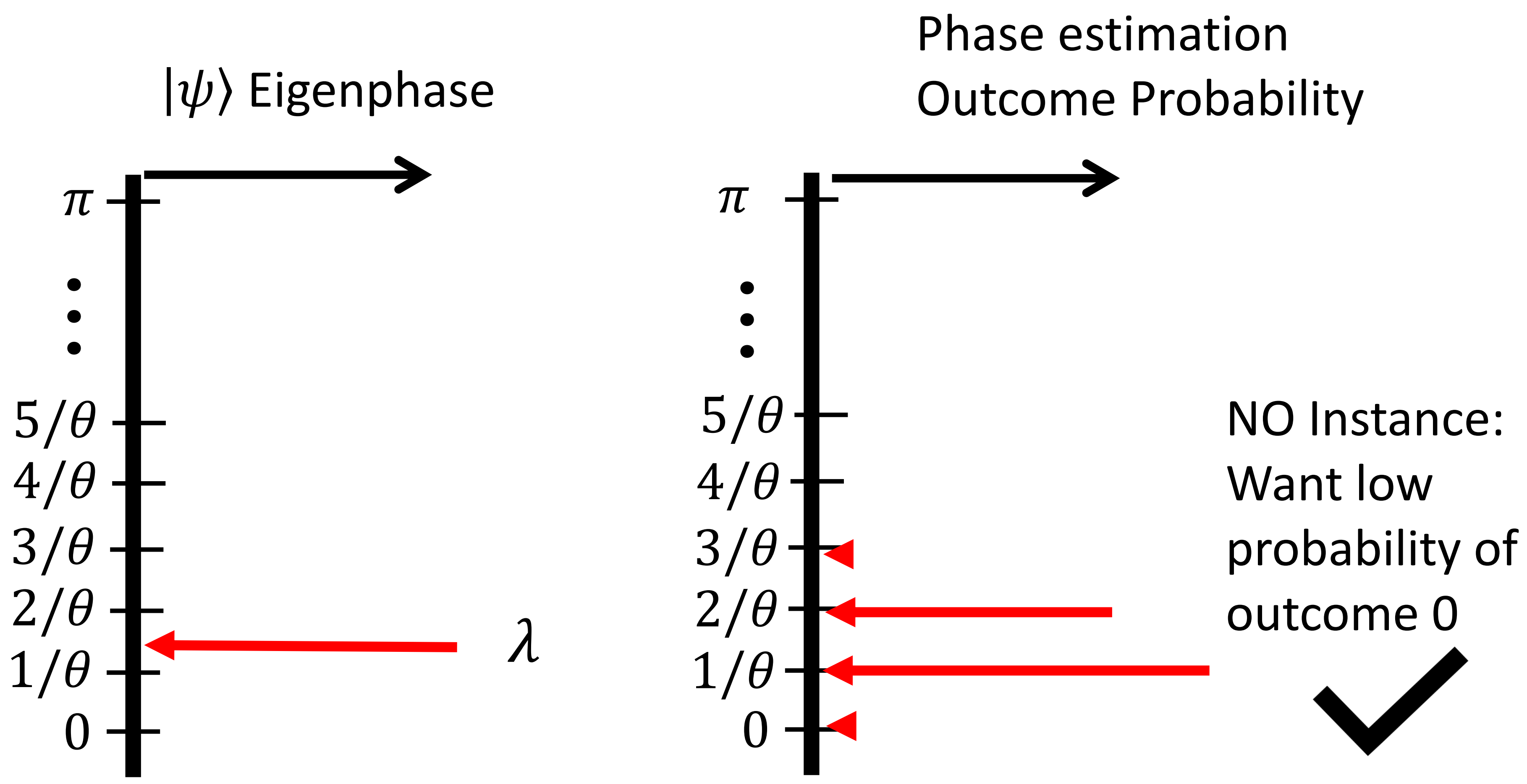
NO Instance: Want low probability of outcome 0

Reduce Precision (θ)

Case 1:

- **HARD NO INSTANCE**
- **Large θ:**

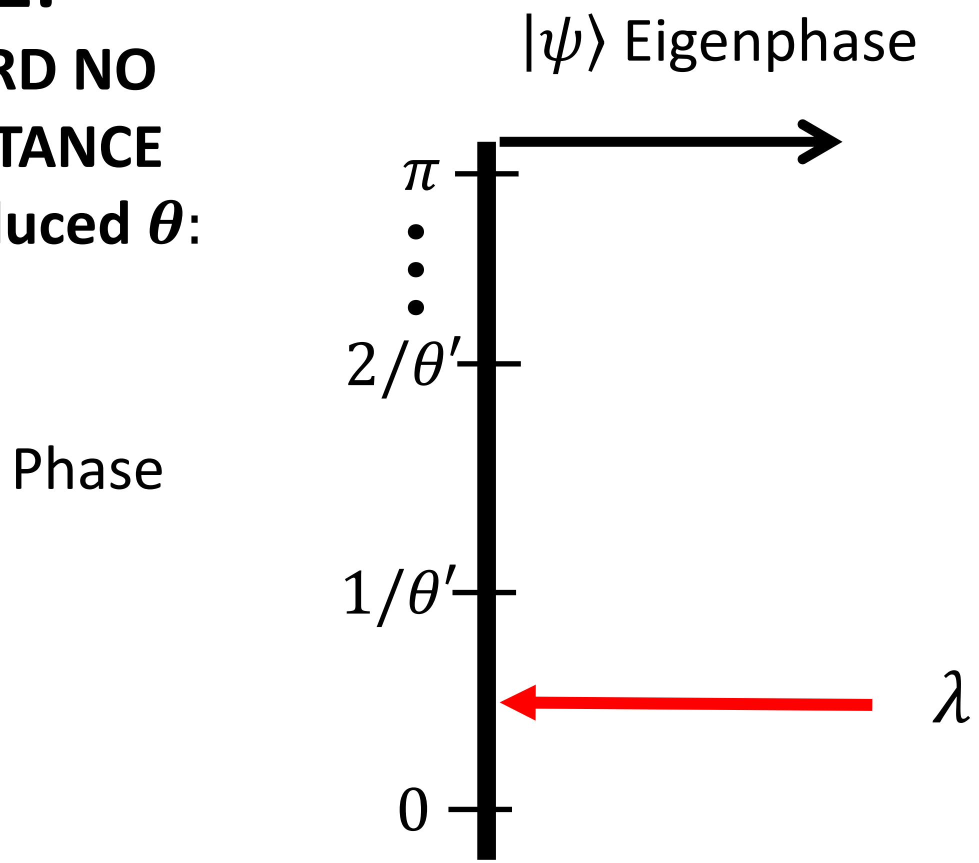
Phase

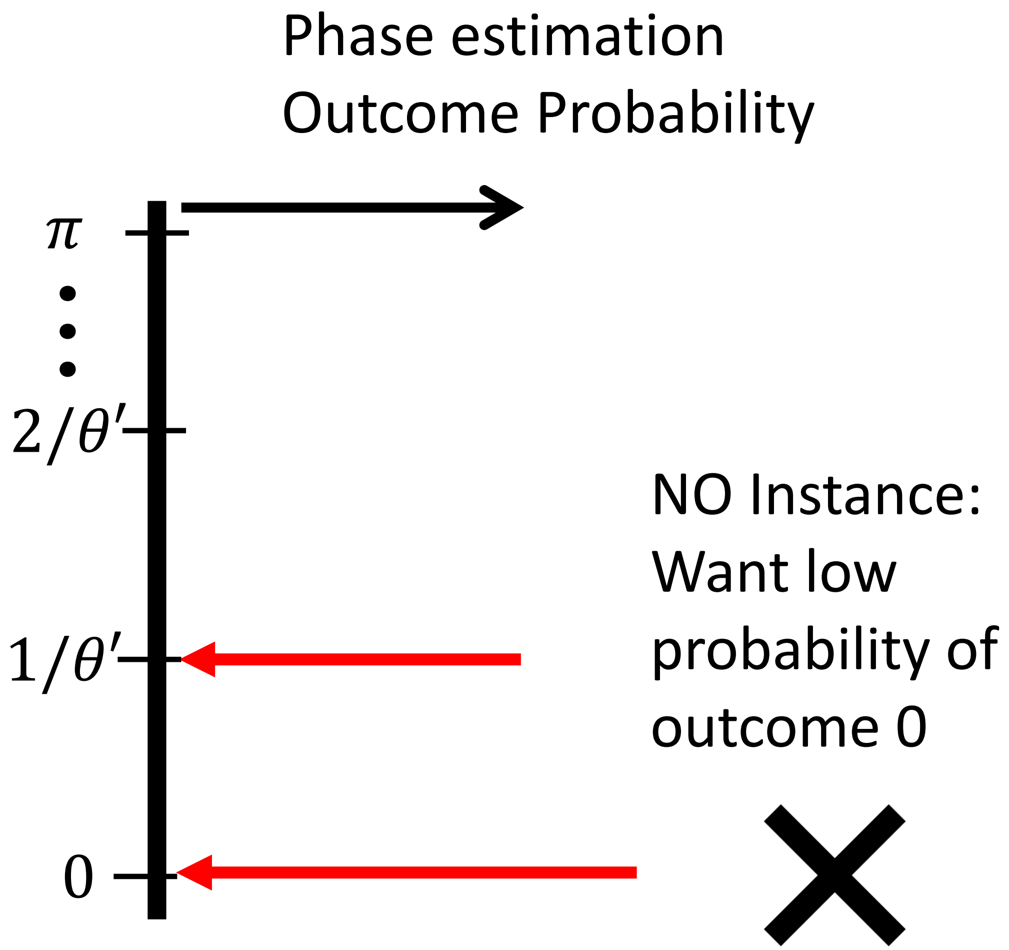


Reduce Precision (θ)

Case 2:

- **HARD NO INSTANCE**
- **Reduced $\boldsymbol{\theta}$:**



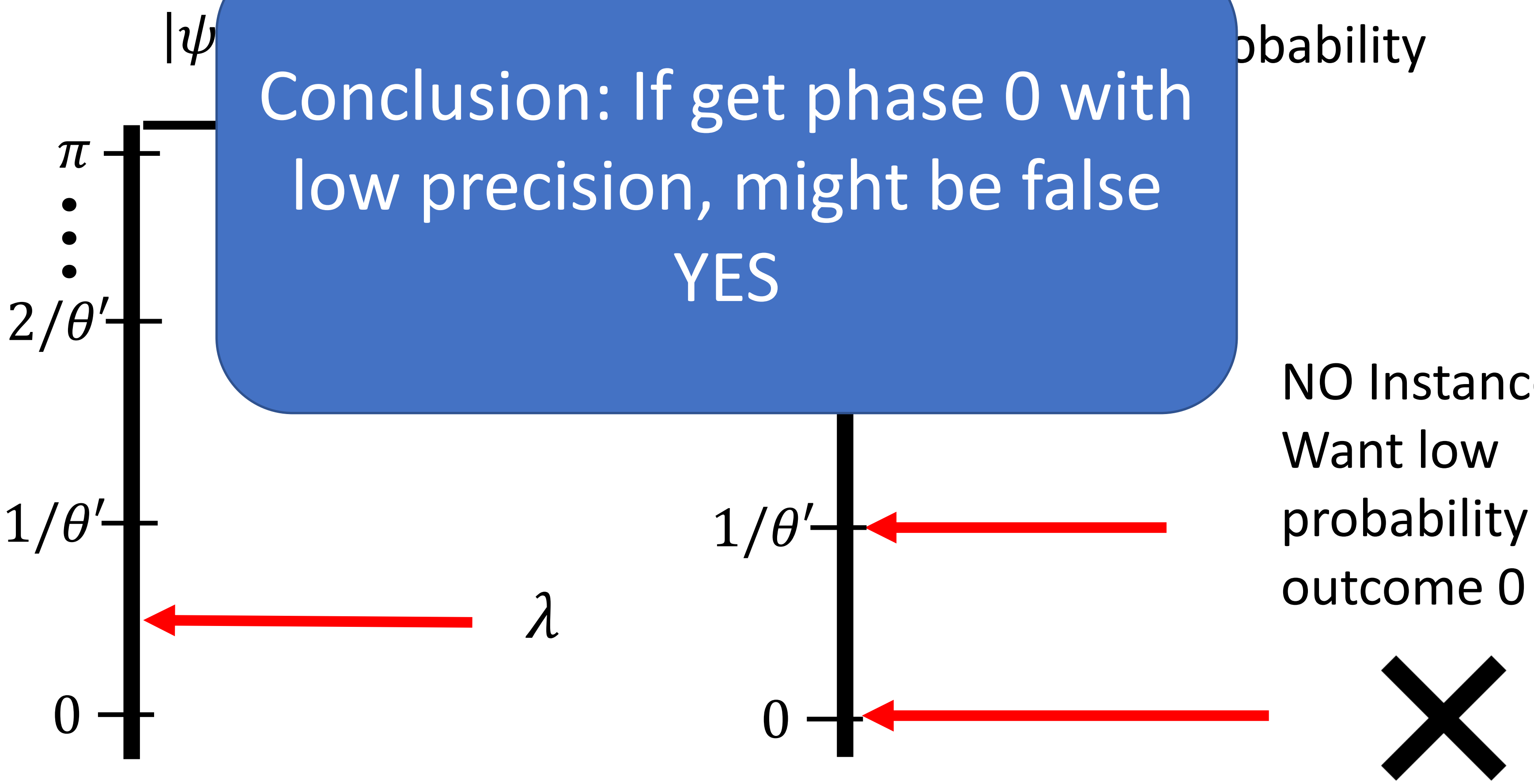


Reduce Precision (θ)

Case 2:

- **HARD NO INSTANCE**
- **Reduced θ:**

Phase



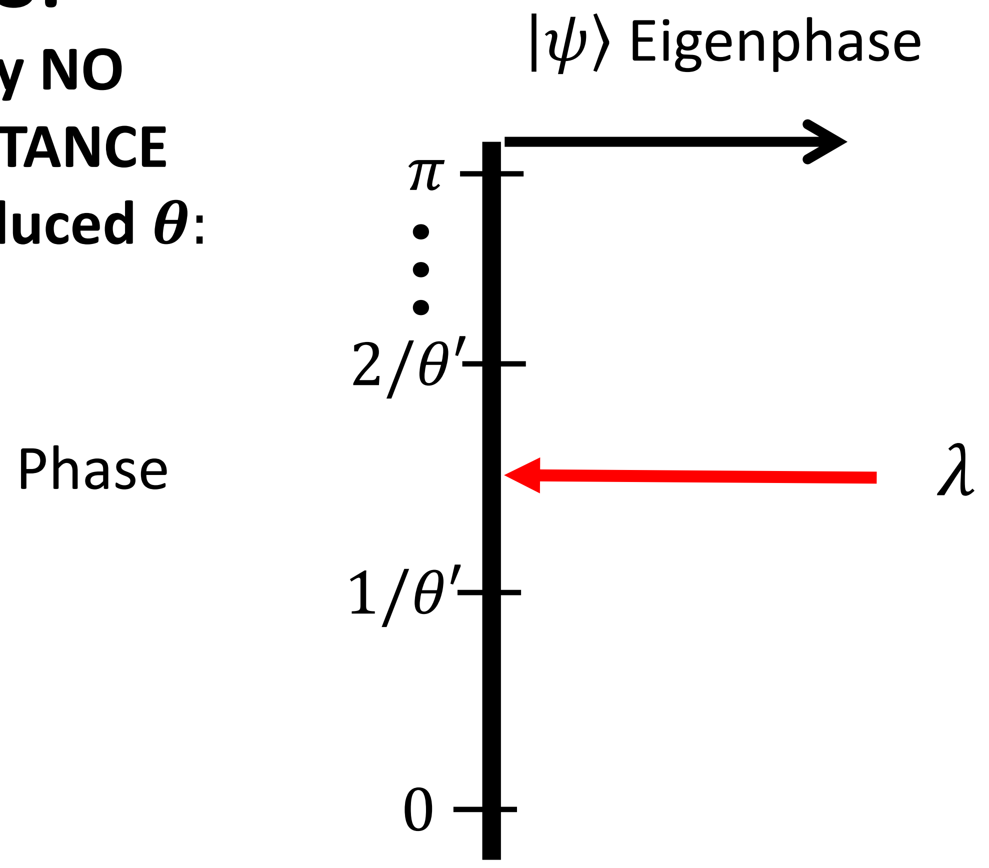
Conclusion: If get phase 0 with low precision, might be false YES

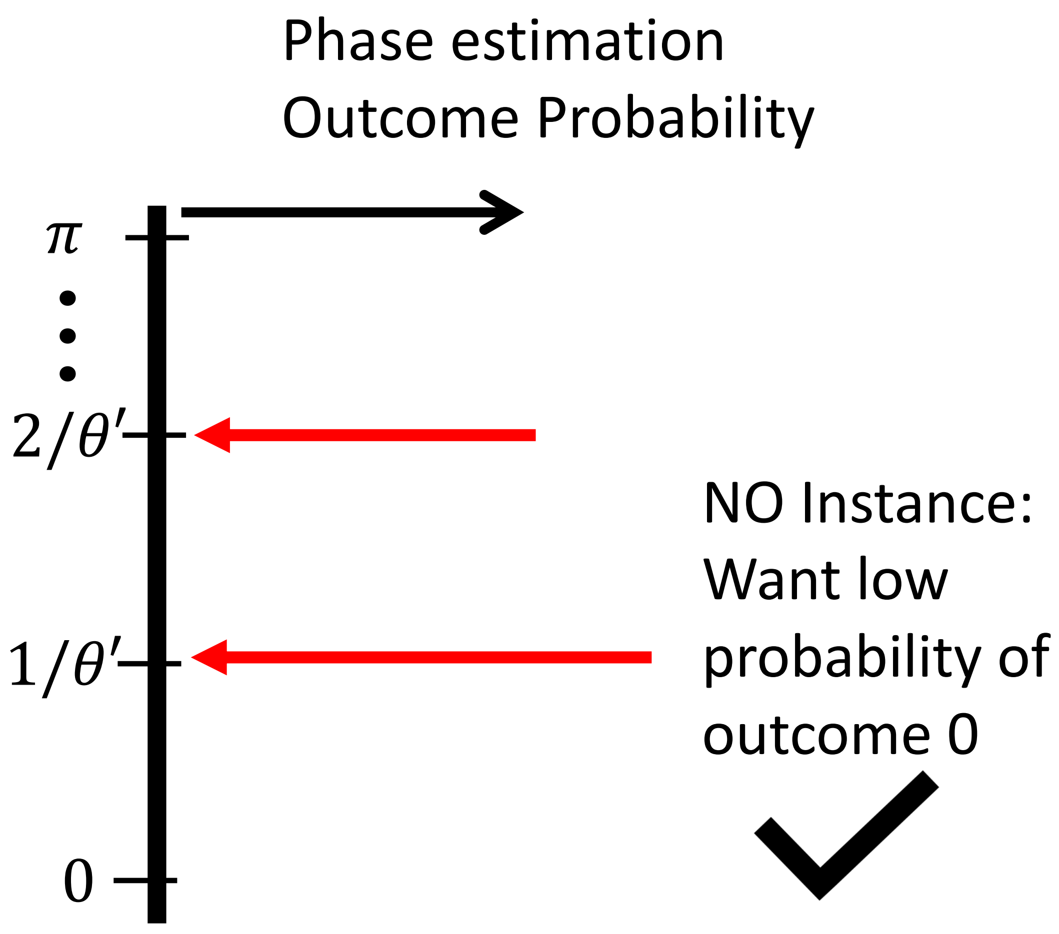
NO Instance: Want low probability of outcome 0

Reduce Precision (θ)

Case 3:

- **Easy NO INSTANCE**
- **Reduced $\boldsymbol{\theta}$:**

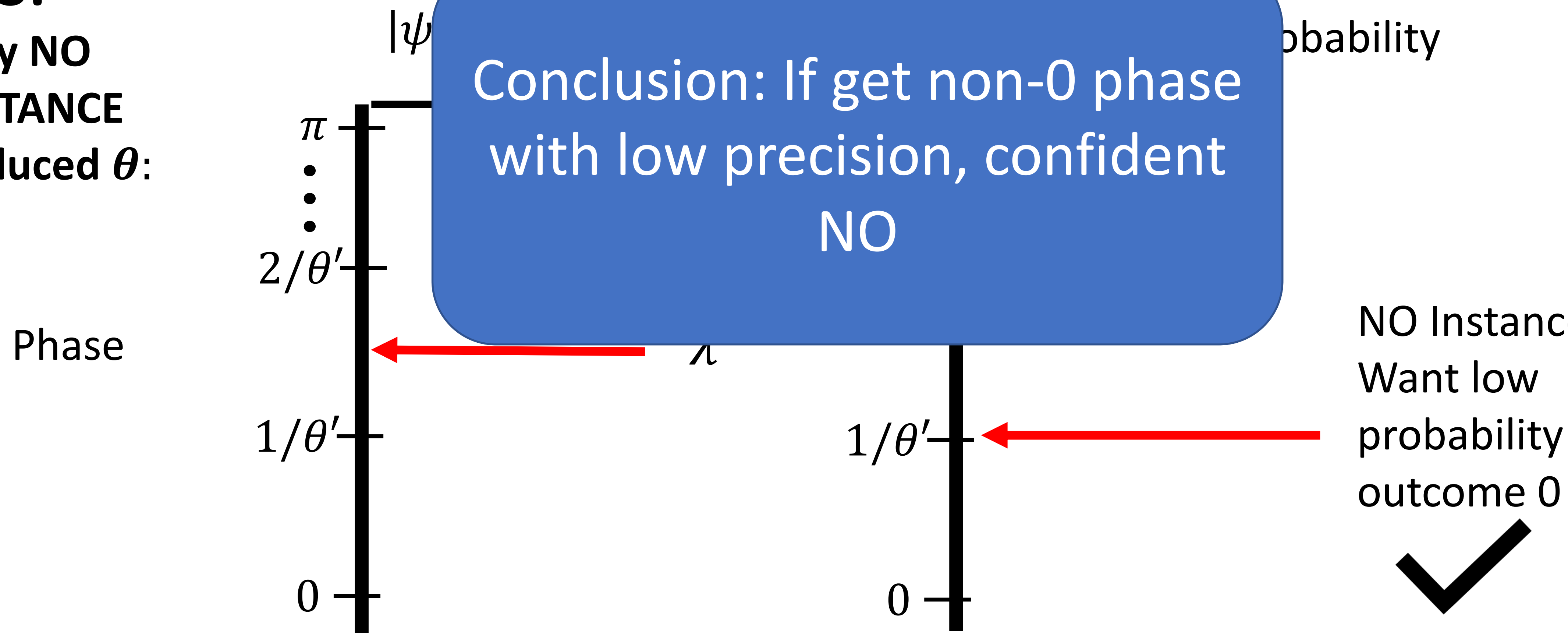




Reduce Precision (θ)

Case 3:

- **Easy NO INSTANCE**
- **Reduced θ:**



Conclusion: If get non-0 phase with low precision, confident NO

NO Instance: Want low probability of outcome 0

Reduce Precision (θ)

- Run span program phase estimation algorithm with exponentially increasing precision θ until reach precision of original algorithm
  - If get 0 phase at any repetition, continue
  - If get non-0 phase at any repetition, stop and output *NO*

Result: faster runtime for easy *NO* instances

Easy Yes Instances?

Design negation procedure to produce a span program where YES/NO instances are exchanged.

Result: Formerly easy YES instances become easy NO instance

All Together

Run with exponentially increasing precision:

- Span program phase estimation algorithm
  - ➢ If get non-0 phase at any repetition, stop and output *NO*
- **Negated** span program phase estimation algorithm
  - ➢ If get non-0 phase at any repetition, stop and output *YES*

Result:

- faster runtime for easy YES and NO instances
- Geometric scaling increases worst-case runtime by only log factor

All Together

Run with exponentially increasing precision:

- Span program phase estimation algorithm
  - If get non-0 phase at any repetition, stop and output *NO*
- **Negated** span program phase estimation algorithm
  - If get non-0 phase at any repetition, stop and output *YES*

Result:

- faster runtime for easy YES and NO instances
- Geometric scaling increases worst-case runtime by only log factor

*Not exactly our algorithm :D

Performance

Given a span program, each instance $x \in X$ has a witness size $w(x)$.

Original span program algorithm query complexity:

$$O\left(\sqrt{\left(\max_{x \in X: f(x)=YES} w(x)\right)\left(\max_{x \in X: f(x)=NO} w(x)\right)}\right)$$

Performance

Given a span program, each instance $x \in X$ has a witness size $w(x)$.

Original span program algorithm query complexity:

$$O\left(\sqrt{\left(\max_{x \in X : f(x) = YES} w(x)\right)\left(\max_{x \in X : f(x) = NO} w(x)\right)}\right)$$

Our query complexity:

- If input instance x' is YES:

$$\tilde{O}\left(\sqrt{w(x')\left(\max_{x \in X : f(x) = NO} w(x)\right)}\right)$$

- If input instance x' is NO

$$\tilde{O}\left(\sqrt{w(x')\left(\max_{x \in X : f(x) = YES} w(x)\right)}\right)$$

Talk Outline

1. Oracle Model
2. Challenges:
   a. Can’t check property of algorithm and then continue running ⟵ **Easy**
      i. Why challenge exists quantumly
      ii. How to overcome
   b. (Frequently) No (easily accessible) witness of completion ⟵ **Harder**
      i. Why challenge exists quantumly
      ii. How to overcome
3. Applications & Future Work

Applications

Frequently: witness size $w(x)$ is related to natural difficulty measures

Applications

Frequently: witness size $w(x)$ is related to natural difficulty measures

- For path detection
  - YES: $w(x) <$ length of shortest path
  - NO: $w(x) <$ size of smallest cut

[Belovs and Reichardt ‘12, Jarret, Jeffery, SK, Piedrafita ’19]

Applications

Frequently: witness size $w(x)$ is related to natural difficulty measures

- For path detection
  - YES: $w(x) <$ length of shortest path
  - NO: $w(x) <$ size of smallest cut

[Belovs and Reichardt ‘12, Jarret, Jeffery, SK, Piedrafita ’19]

- For total connectivity
  - YES: $w(x) <$ average effective resistance
  - NO: $w(x) < 1/$(number of components)

[Jarret, Jeffery, SK, Piedrafita ’19]

Applications

Frequently: witness size $w(x)$ is related to natural difficulty measures

- For path detection
  - YES: $w(x) <$ length of shortest path
  - NO: $w(x) <$ size of smallest cut

[Belovs and Reichardt ‘12, Jarret, Jeffery, SK, Piedrafita ’19]

- For total connectivity
  - YES: $w(x) <$ average effective resistance
  - NO: $w(x) <$ 1/(number of components)

[Jarret, Jeffery, SK, Piedrafita ’19]

- For cycle finding
  - YES: $w(x) =$1/(cycle rank)
  - NO: $w(x) <$ no. of edges

[DeLorenzo, SK, Witter ’20]

Applications

Frequently: witness size $w(x)$ is related to natural difficulty measures

- For path detection
  - YES: $w(x) <$ length of shortest path
  - NO: $w(x) <$ size of smallest cut

[Belovs and Reichardt '12, Jarret, Jeffery, SK, Piedrafita '19]

- For total connectivity
  - YES: $w(x) <$ average effective resistance
  - NO: $w(x) <$ 1/(number of components)

[Jarret, Jeffery, SK, Piedrafita '19]

- For cycle finding
  - YES: $w(x) =$1/(cycle rank)
  - NO: $w(x) <$ no. of edges

[DeLorenzo, SK, Witter '20]

- For search
  - YES: $w(x) =$ no. of marked items

State generation extension

State Generation Problem:
Convert $|\rho_x\rangle$ to $|\sigma_x\rangle$ given access to O_x.

There is a span program-like algorithm that is nearly optimal for worst-case x.
Running faster on easier inputs?

Challenge:
Original algorithm has no measurements!

Our Result:
Use an auxiliary test to determine when can stop running.

Future Work

- Get rid of log factors from error suppression? (Fixed-point methods)
- Opportunities for average case quantum vs. classical speed-ups
- Faster algorithms for producing witness states for easy instances
- Better error parameters for state generation
- Use these ideas to speed up non-span program algorithms on easy inputs

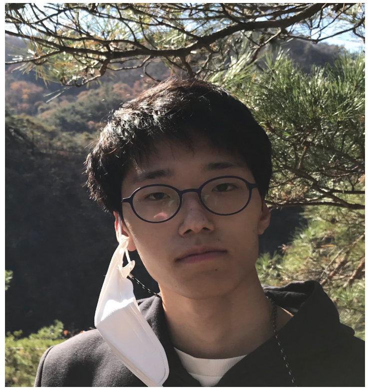
https://arxiv.org/pdf/2012.01276.pdf

Thank you!

Middlebury College

ARO

Noel Anderson



Jay-U Chung