

CS3230 – Design and Analysis of
Algorithms (S1 AY2025/26)

Lecture 3a: Proof of Correctness

Glossary

• **Dominant term:** $n^2 + n \log n + 5000 \in \Theta(n^2)$
↑
Dominant term

$n^2 + n + 2^n \in \Theta(2^n)$
↑
Dominant term

• **Linearity:** $f(n) \in \Theta(n)$
↕
 $f(n)$ is linear in n .

$f(n) \in \Theta(n^2)$
↕
 $f(n)$ is linear in n^2 .

• **Polylog:** $f(n) = n^c$ for some $c \in \Theta(1)$
↕
 $f(n)$ is polynomial.

$f(n) = (\log n)^c$ for some $c \in \Theta(1)$
↕
 $f(n)$ is polylog.

Answer to the exercise from Week 2

- $T(n) = T(n/2) + 2^{\sqrt{\log n}}$
- $T(n) = \sum_{x=0}^{\log n} 2^{\sqrt{x}} \approx \int_0^{\log n} 2^{\sqrt{x}} dx \in \Theta\left(2^{\sqrt{\log n}} \cdot \sqrt{\log n}\right)$

Approximating a sum by an integral

<https://www.wolframalpha.com/input?i=integrate+2%5E%28sqrt%28x%29%29>

(It is totally fine if you cannot solve it)

Recursion tree:

$$\begin{array}{c} 2^{\sqrt{\log n}} \\ | \\ 2^{\sqrt{\log \frac{n}{2}}} = 2^{\sqrt{(\log n)-1}} \\ | \\ 2^{\sqrt{\log \frac{n}{4}}} = 2^{\sqrt{(\log n)-2}} \\ | \\ 2^{\sqrt{\log \frac{n}{8}}} = 2^{\sqrt{(\log n)-3}} \\ | \\ \dots \end{array}$$

Correctness of an algorithm

- **Goal:** For a given algorithm \mathcal{A} , prove that it is correct.
 - Iterative algorithms.
 - Recursive algorithms.

Fib(n)

- If $n \leq 1$, return n .
- Else, return **Fib**($n - 1$) + **Fib**($n - 2$).

IFib(n)

- If $n \leq 1$
 - return n
- Else,
 - prev2 = 0
 - prev1 = 1
 - for $i = 2$ to n
 - temp = prev1
 - prev1 = prev1+prev2
 - prev2 = temp
 - return prev1

Iterative algorithms

While (some condition is met)
• **Do** { some work }

For ($i = 1, 2, \dots$ up to $i = n$)
• **Do** { some work }

- **Goal:** For a given algorithm \mathcal{A} , prove that it is correct.
- Analysis of an iterative algorithm:
 - **Loop invariant:**
 - Some desirable conditions that should be satisfied at the start of each iteration.

Iterative algorithms

While (some condition is met)
• **Do** { some work }

For ($i = 1, 2, \dots$ up to $i = n$)
• **Do** { some work }

- **Goal:** For a given algorithm \mathcal{A} , prove that it is correct.
- Analysis of an iterative algorithm:
 - **Loop invariant:**
 - Some desirable conditions that should be satisfied at the start of each iteration.
 - **Initialization:**
 - The loop invariant is true at the start of the first iteration.

Iterative algorithms

While (some condition is met)
• **Do** { some work }

For ($i = 1, 2, \dots$ up to $i = n$)
• **Do** { some work }

- **Goal:** For a given algorithm \mathcal{A} , prove that it is correct.
- Analysis of an iterative algorithm:
 - **Loop invariant:**
 - Some desirable conditions that should be satisfied at the start of each iteration.
 - **Initialization:**
 - The loop invariant is true at the start of the first iteration.
 - **Maintenance:**
 - If the loop invariant is satisfied at the start of the current iteration, then the loop invariant must be satisfied at the start of the next iteration.

Equivalently, the end of the current iteration.

Iterative algorithms

While (some condition is met)
• **Do** { some work }

For ($i = 1, 2, \dots$ up to $i = n$)
• **Do** { some work }

- **Goal:** For a given algorithm \mathcal{A} , prove that it is correct.
- Analysis of an iterative algorithm:
 - **Loop invariant:**
 - Some desirable conditions that should be satisfied at the start of each iteration.
 - **Initialization:**
 - The loop invariant is true at the start of the first iteration.
 - **Maintenance:**
 - If the loop invariant is satisfied at the start of the current iteration, then the loop invariant must be satisfied at the start of the next iteration.
 - **Termination:**
 - Loop invariant at the end of the last iteration \rightarrow The algorithm outputs a correct answer.

Iterative algorithms

While (some condition is met)
• **Do** { some work }

For ($i = 1, 2, \dots$ up to $i = n$)
• **Do** { some work }

- **Goal:** For a given algorithm \mathcal{A} , prove that it is correct.
- Analysis of an iterative algorithm:
 - **Loop invariant:** Induction hypothesis
 - Some desirable conditions that should be satisfied at the start of each iteration.
 - **Initialization:** Base case
 - The loop invariant is true at the start of the first iteration.
 - **Maintenance:** Inductive step
 - If the loop invariant is satisfied at the start of the current iteration, then the loop invariant must be satisfied at the start of the next iteration.
 - **Termination:** The proof by induction implies the correctness of the algorithm
 - Loop invariant at the end of the last iteration \rightarrow The algorithm outputs a correct answer.

Fibonacci numbers

- **Goal:** For a given algorithm \mathcal{A} , prove that it is correct.

IFib(n)

- If $n \leq 1$
 - return n
- Else,
 - $\text{prev2} = 0$
 - $\text{prev1} = 1$
 - for $i = 2$ to n
 - $\text{temp} = \text{prev1}$
 - $\text{prev1} = \text{prev1} + \text{prev2}$
 - $\text{prev2} = \text{temp}$
 - return prev1

If $n \leq 1$, then the algorithm is correct.

- **Fib**(0) = 0
- **Fib**(1) = 1

Fibonacci numbers

- **Goal:** For a given algorithm \mathcal{A} , prove that it is correct.

IFib(n)

- If $n \leq 1$
 - return n
- Else,
 - $\text{prev2} = 0$
 - $\text{prev1} = 1$
 - for $i = 2$ to n
 - $\text{temp} = \text{prev1}$
 - $\text{prev1} = \text{prev1} + \text{prev2}$
 - $\text{prev2} = \text{temp}$
 - return prev1

Now, consider the case of $n \geq 2$.

Loop invariant:

- At the start of iteration i ,
 - $\text{prev2} = \mathbf{Fib}(i - 2)$
 - $\text{prev1} = \mathbf{Fib}(i - 1)$

Fibonacci numbers

- **Goal:** For a given algorithm \mathcal{A} , prove that it is correct.

IFib(n)

- If $n \leq 1$
 - return n
- Else,
 - $\text{prev2} = 0$
 - $\text{prev1} = 1$
 - for $i = 2$ to n
 - $\text{temp} = \text{prev1}$
 - $\text{prev1} = \text{prev1} + \text{prev2}$
 - $\text{prev2} = \text{temp}$
 - return prev1

Now, consider the case of $n \geq 2$.

Loop invariant:

- At the start of iteration i ,
 - $\text{prev2} = \mathbf{Fib}(i - 2)$
 - $\text{prev1} = \mathbf{Fib}(i - 1)$

Initialization:

- At the start of iteration $i = 2$,
 - $\text{prev2} = \mathbf{Fib}(0) = 0$
 - $\text{prev1} = \mathbf{Fib}(1) = 1$

Fibonacci numbers

- **Goal:** For a given algorithm \mathcal{A} , prove that it is correct.

IFib(n)

- If $n \leq 1$
 - return n
- Else,
 - $\text{prev2} = 0$
 - $\text{prev1} = 1$
 - for $i = 2$ to n
 - $\text{temp} = \text{prev1}$ $\text{Fib}(i-1)$
 - $\text{prev1} = \text{prev1} + \text{prev2}$ $\text{Fib}(i-1) + \text{Fib}(i-2)$
 - $\text{prev2} = \text{temp}$ $\text{Fib}(i-1)$
 - return prev1

Now, consider the case of $n \geq 2$.

Loop invariant:

- At the start of iteration i ,
 - $\text{prev2} = \mathbf{Fib}(i-2)$
 - $\text{prev1} = \mathbf{Fib}(i-1)$

Initialization:

- At the start of iteration $i = 2$,
 - $\text{prev2} = \mathbf{Fib}(0) = 0$
 - $\text{prev1} = \mathbf{Fib}(1) = 1$

Maintenance:

- Suppose at the start of iteration i ,
 - $\text{prev2} = \mathbf{Fib}(i-2)$
 - $\text{prev1} = \mathbf{Fib}(i-1)$
- Then at the end of the iteration,
 - $\text{prev2} = \mathbf{Fib}(i-1)$
 - $\text{prev1} = \mathbf{Fib}(i)$

Fibonacci numbers

- **Goal:** For a given algorithm \mathcal{A} , prove that it is correct.

IFib(n)

- If $n \leq 1$
 - return n
- Else,
 - $\text{prev2} = 0$
 - $\text{prev1} = 1$
 - for $i = 2$ to n
 - $\text{temp} = \text{prev1}$
 - $\text{prev1} = \text{prev1} + \text{prev2}$
 - $\text{prev2} = \text{temp}$
 - return prev1

Now, consider the case of $n \geq 2$.

Loop invariant:

- At the start of iteration i ,
 - $\text{prev2} = \mathbf{Fib}(i - 2)$
 - $\text{prev1} = \mathbf{Fib}(i - 1)$

Initialization:

- At the start of iteration $i = 2$,
 - $\text{prev2} = \mathbf{Fib}(0) = 0$
 - $\text{prev1} = \mathbf{Fib}(1) = 1$

Maintenance:

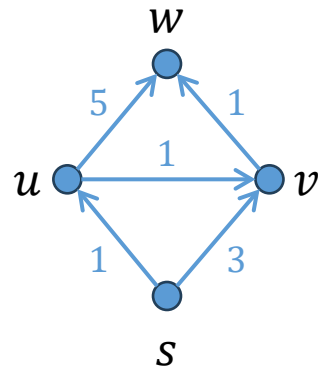
- Suppose at the start of iteration i ,
 - $\text{prev2} = \mathbf{Fib}(i - 2)$
 - $\text{prev1} = \mathbf{Fib}(i - 1)$ $i = n$
- Then at the **end** of the iteration,
 - $\text{prev2} = \mathbf{Fib}(i - 1)$
 - $\text{prev1} = \mathbf{Fib}(i)$

Termination:

- The algorithm returns $\mathbf{Fib}(n)$.

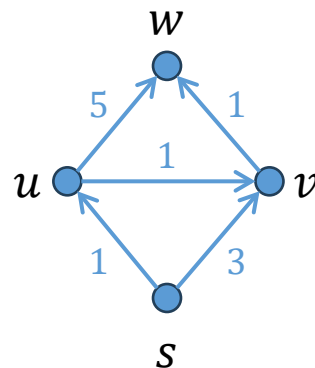
Weighted directed graphs

- Let $G = (V, E)$ be a directed graph.
- Each edge $e \in E$ has a positive weight $w(e)$.
 - If $(u, v) \in E$, then $w(u, v) = w(e)$, where $e = (u, v)$.
 - If $(u, v) \notin E$, then $w(u, v) = \infty$.



Single-source shortest paths

- Let $G = (V, E)$ be a directed graph.
- Each edge $e \in E$ has a positive weight $w(e)$.
 - If $(u, v) \in E$, then $w(u, v) = w(e)$, where $e = (u, v)$.
 - If $(u, v) \notin E$, then $w(u, v) = \infty$.
- **Goal:** Given a source $s \in V$, compute the shortest-path distance from s to all vertices.

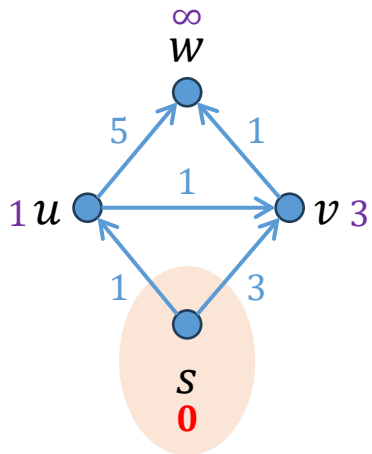


dist(s, s) = 0
dist(s, u) = 1
dist(s, v) = 2
dist(s, w) = 3

Dijkstra's algorithm

Dijkstra($G = (V, E), s \in V$)

- $d(s) = 0$
- $R = \{s\}$
- **While** $R \neq V$
 - Select $v \in V \setminus R$ to minimize $\min_{u \in R} (d(u) + w(u, v))$
 - $d(v) = \min_{u \in R} (d(u) + w(u, v))$
 - Add v to R

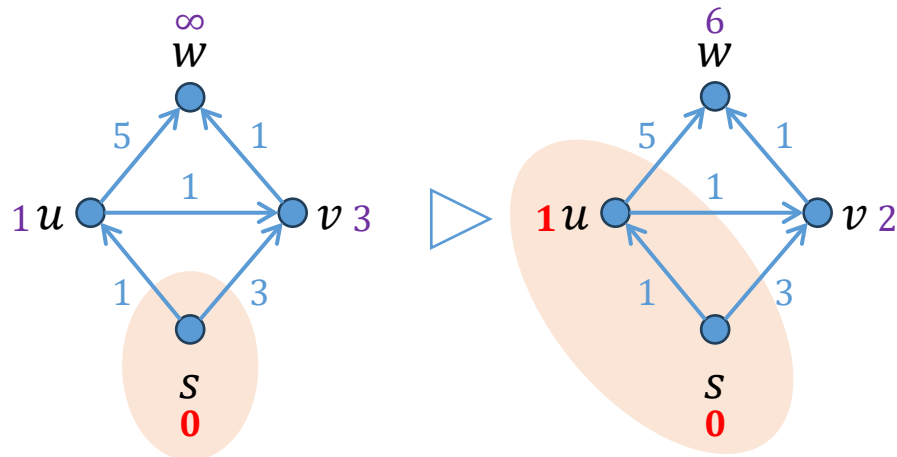


dist(s, s) = 0
dist(s, u) = 1
dist(s, v) = 2
dist(s, w) = 3

Dijkstra's algorithm

Dijkstra($G = (V, E), s \in V$)

- $d(s) = 0$
- $R = \{s\}$
- **While** $R \neq V$
 - Select $v \in V \setminus R$ to minimize $\min_{u \in R} (d(u) + w(u, v))$
 - $d(v) = \min_{u \in R} (d(u) + w(u, v))$
 - Add v to R

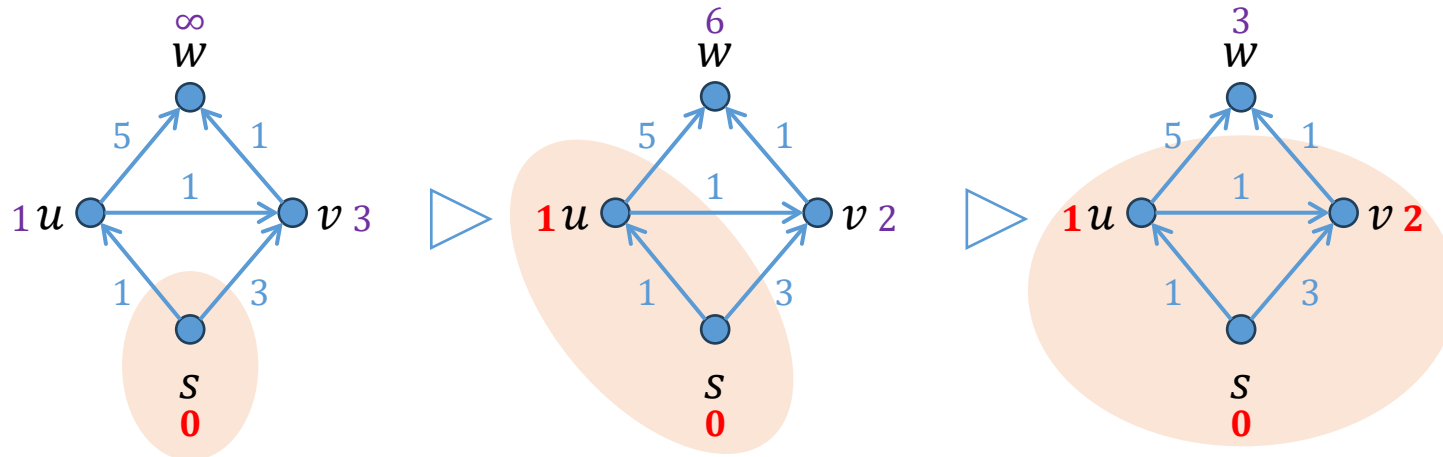


dist(s, s) = 0
dist(s, u) = 1
dist(s, v) = 2
dist(s, w) = 3

Dijkstra's algorithm

Dijkstra($G = (V, E), s \in V$)

- $d(s) = 0$
- $R = \{s\}$
- **While** $R \neq V$
 - Select $v \in V \setminus R$ to minimize $\min_{u \in R} (d(u) + w(u, v))$
 - $d(v) = \min_{u \in R} (d(u) + w(u, v))$
 - Add v to R

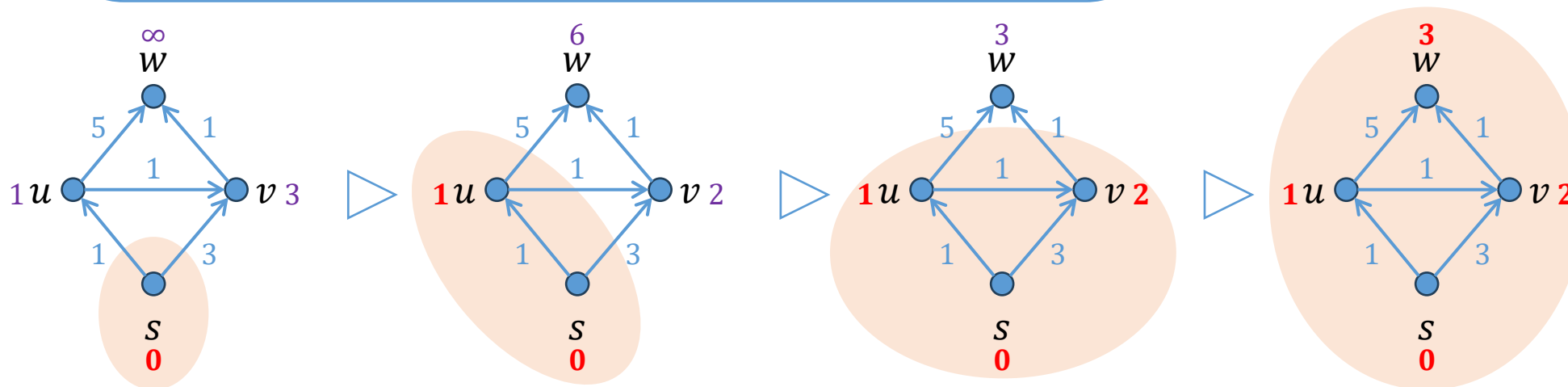


dist(s, s) = 0
dist(s, u) = 1
dist(s, v) = 2
dist(s, w) = 3

Dijkstra's algorithm

Dijkstra($G = (V, E), s \in V$)

- $d(s) = 0$
- $R = \{s\}$
- **While** $R \neq V$
 - Select $v \in V \setminus R$ to minimize $\min_{u \in R} (d(u) + w(u, v))$
 - $d(v) = \min_{u \in R} (d(u) + w(u, v))$
 - Add v to R



dist(s, s) = 0
dist(s, u) = 1
dist(s, v) = 2
dist(s, w) = 3

Proof of correctness

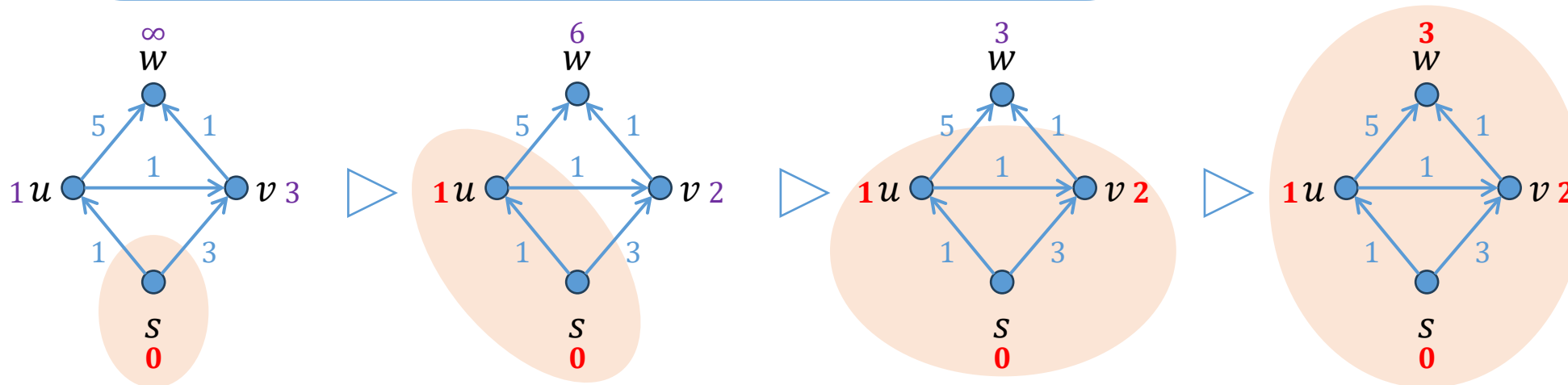
Loop invariant:

- At the start of an iteration:
 - $\forall u \in R, d(u) = \mathbf{dist}(s, u)$

The computed distances are correct.

Dijkstra($G = (V, E), s \in V$)

- $d(s) = 0$
- $R = \{s\}$
- **While** $R \neq V$
 - Select $v \in V \setminus R$ to minimize $\min_{u \in R} (d(u) + w(u, v))$
 - $d(v) = \min_{u \in R} (d(u) + w(u, v))$
 - Add v to R



$\mathbf{dist}(s, s) = 0$
 $\mathbf{dist}(s, u) = 1$
 $\mathbf{dist}(s, v) = 2$
 $\mathbf{dist}(s, w) = 3$

Proof of correctness

Loop invariant:

- At the start of an iteration:
 - $\forall u \in R, d(u) = \mathbf{dist}(s, u)$

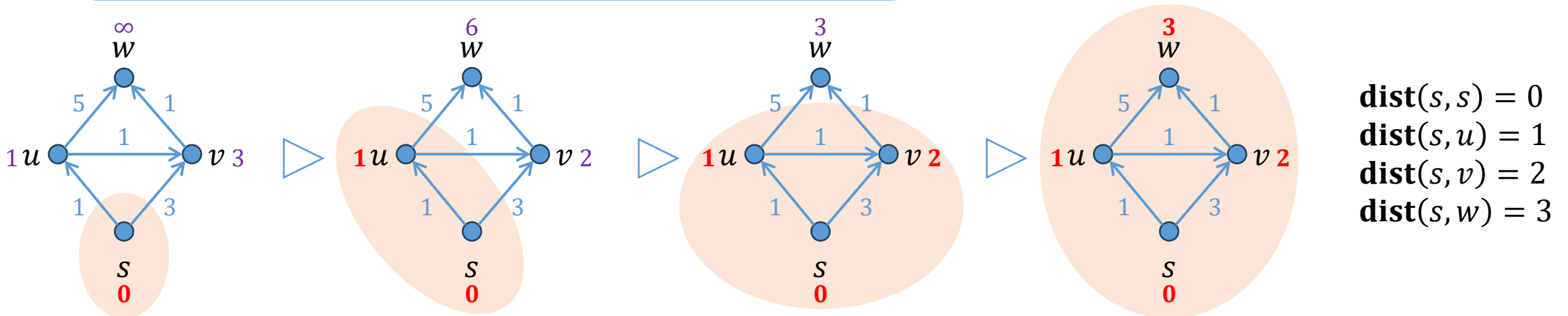
The computed distances are correct.

Dijkstra($G = (V, E), s \in V$)

- $d(s) = 0$
- $R = \{s\}$
- While** $R \neq V$
 - Select $v \in V \setminus R$ to minimize $\min_{u \in R} (d(u) + w(u, v))$
 - $d(v) = \min_{u \in R} (d(u) + w(u, v))$
 - Add v to R $\mathbf{dist}(s, v)$

Observation 1: $d(v) = \min_{u \in R} (d(u) + w(u, v)) \geq \mathbf{dist}(s, v)$

Proof: There is a path from s to v of this length.



Proof of correctness

Loop invariant:

- At the start of an iteration:
 - $\forall u \in R, d(u) = \mathbf{dist}(s, u)$

The computed distances are correct.

$d(s) = 0$ is correct.

Dijkstra($G = (V, E), s \in V$)

- $d(s) = 0$
- $R = \{s\}$
- **While** $R \neq V$
 - Select $v \in V \setminus R$ to minimize $\min_{u \in R} (d(u) + w(u, v))$
 - $d(v) = \min_{u \in R} (d(u) + w(u, v))$
 - Add v to R $\mathbf{dist}(s, v)$

Observation 1: $d(v) = \min_{u \in R} (d(u) + w(u, v)) \geq \mathbf{dist}(s, v)$

- ✓ **Initialization**
- Maintenance
- Termination

Proof of correctness

Loop invariant:

- At the start of an iteration:
 - $\forall u \in R, d(u) = \mathbf{dist}(s, u)$

The computed distances are correct.

Dijkstra($G = (V, E), s \in V$)

- $d(s) = 0$
- $R = \{s\}$
- **While** $R \neq V$
 - Select $v \in V \setminus R$ to minimize $\min_{u \in R} (d(u) + w(u, v))$
 - $d(v) = \min_{u \in R} (d(u) + w(u, v))$
 - Add v to R $\mathbf{dist}(s, v)$

Observation 1: $d(v) = \min_{u \in R} (d(u) + w(u, v)) \geq \mathbf{dist}(s, v)$

- ✓ Initialization
- Maintenance
- ✓ Termination

At the end of the computation, $R = V$

Proof of correctness

Loop invariant:

- At the start of an iteration:
 - $\forall u \in R, d(u) = \mathbf{dist}(s, u)$

The computed distances are correct.

Dijkstra($G = (V, E), s \in V$)

- $d(s) = 0$
- $R = \{s\}$
- **While** $R \neq V$
 - Select $v \in V \setminus R$ to minimize $\min_{u \in R} (d(u) + w(u, v))$
 - $d(v) = \min_{u \in R} (d(u) + w(u, v))$
 - Add v to R $\mathbf{dist}(s, v)$

Observation 1: $d(v) = \min_{u \in R} (d(u) + w(u, v)) \geq \mathbf{dist}(s, v)$

- ✓ Initialization
- Maintenance**
- ✓ Termination

Proof of correctness

Loop invariant:

- At the start of an iteration:
 - $\forall u \in R, d(u) = \mathbf{dist}(s, u)$

The computed distances are correct.

Dijkstra($G = (V, E), s \in V$)

- $d(s) = 0$
- $R = \{s\}$
- **While** $R \neq V$
 - Select $v \in V \setminus R$ to minimize $\min_{u \in R} (d(u) + w(u, v))$
 - $d(v) = \min_{u \in R} (d(u) + w(u, v))$
 - Add v to R $\mathbf{dist}(s, v)$

Observation 1: $d(v) = \min_{u \in R} (d(u) + w(u, v)) \geq \mathbf{dist}(s, v)$

Claim: For the selected vertex v :

- $\mathbf{dist}(s, v) = \min_{u \in R} (d(u) + w(u, v))$

✓ Initialization

Maintenance

✓ Termination

Proof of correctness

Loop invariant:

- At the start of an iteration:
 - $\forall u \in R, d(u) = \mathbf{dist}(s, u)$

The computed distances are correct.

Dijkstra($G = (V, E), s \in V$)

- $d(s) = 0$
- $R = \{s\}$
- **While** $R \neq V$
 - Select $v \in V \setminus R$ to minimize $\min_{u \in R} (d(u) + w(u, v))$
 - $d(v) = \min_{u \in R} (d(u) + w(u, v))$
 - Add v to R $\mathbf{dist}(s, v)$

Observation 1: $d(v) = \min_{u \in R} (d(u) + w(u, v)) \geq \mathbf{dist}(s, v)$

Claim: For the selected vertex v :

- $\mathbf{dist}(s, v) = \min_{u \in R} (d(u) + w(u, v))$

✓ Initialization

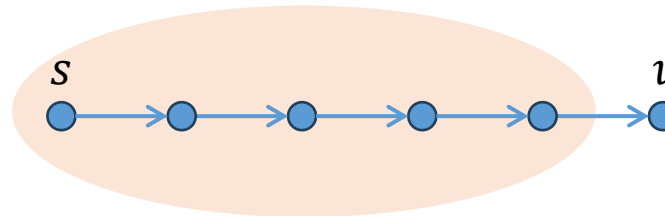
Maintenance

✓ Termination

Proof of the claim:

Consider a shortest path P from s to v .

Observation 2: All vertices in $P \setminus \{v\}$ must be in R .



Proof of correctness

Loop invariant:

- At the start of an iteration:
 - $\forall u \in R, d(u) = \mathbf{dist}(s, u)$

The computed distances are correct.

Dijkstra($G = (V, E), s \in V$)

- $d(s) = 0$
- $R = \{s\}$
- While** $R \neq V$
 - Select $v \in V \setminus R$ to minimize $\min_{u \in R} (d(u) + w(u, v))$
 - $d(v) = \min_{u \in R} (d(u) + w(u, v))$
 - Add v to R $\mathbf{dist}(s, v)$

Observation 1: $d(v) = \min_{u \in R} (d(u) + w(u, v)) \geq \mathbf{dist}(s, v)$

Claim: For the selected vertex v :

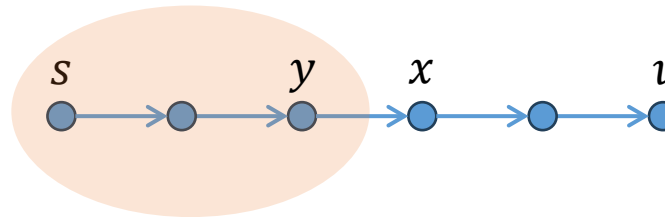
- $\mathbf{dist}(s, v) = \min_{u \in R} (d(u) + w(u, v))$

- ✓ Initialization
- Maintenance**
- ✓ Termination

Proof of the claim:

Consider a shortest path P from s to v .

Observation 2: All vertices in $P \setminus \{v\}$ must be in R .



Proof of Observation 2:

- Otherwise, we should have selected the first vertex that is not in R . The reason is:

$$\min_{u \in R} (d(u) + w(u, v)) \geq \mathbf{dist}(s, v) > d(y) + w(y, x) \geq \min_{u \in R} (d(u) + w(u, x))$$

↑
↑
↑

Observation 1
 $\mathbf{dist}(s, y)$
 $y \in R$

Proof of correctness

Loop invariant:

- At the start of an iteration:
 - $\forall u \in R, d(u) = \mathbf{dist}(s, u)$

The computed distances are correct.

Dijkstra($G = (V, E), s \in V$)

- $d(s) = 0$
- $R = \{s\}$
- While** $R \neq V$
 - Select $v \in V \setminus R$ to minimize $\min_{u \in R} (d(u) + w(u, v))$
 - $d(v) = \min_{u \in R} (d(u) + w(u, v))$
 - Add v to R $\mathbf{dist}(s, v)$

Observation 1: $d(v) = \min_{u \in R} (d(u) + w(u, v)) \geq \mathbf{dist}(s, v)$

Claim: For the selected vertex v :

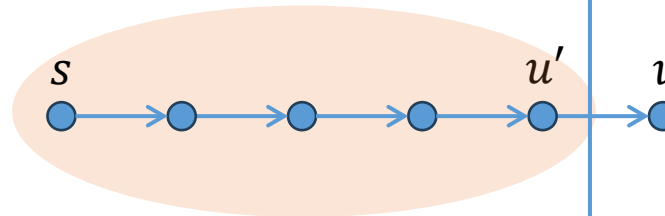
✓ $\mathbf{dist}(s, v) = \min_{u \in R} (d(u) + w(u, v))$

- ✓ Initialization
- Maintenance**
- ✓ Termination

Proof of the claim:

Consider a shortest path P from s to v .

Observation 2: All vertices in $P \setminus \{v\}$ must be in R .



$$\min_{u \in R} (d(u) + w(u, v)) \geq \mathbf{dist}(s, v) = \mathbf{dist}(s, u') + w(u', v) \geq \min_{u \in R} (d(u) + w(u, v))$$

↑
Observation 1
 $\mathbf{dist}(s, u')$
↑
 $u' \in R$

Proof of correctness

Loop invariant:

- At the start of an iteration:
 - $\forall u \in R, d(u) = \mathbf{dist}(s, u)$

The computed distances are correct.

Dijkstra($G = (V, E), s \in V$)

- $d(s) = 0$
- $R = \{s\}$
- **While** $R \neq V$
 - Select $v \in V \setminus R$ to minimize $\min_{u \in R} (d(u) + w(u, v))$
 - $d(v) = \min_{u \in R} (d(u) + w(u, v))$
 - Add v to R $\mathbf{dist}(s, v)$

Observation 1: $d(v) = \min_{u \in R} (d(u) + w(u, v)) \geq \mathbf{dist}(s, v)$

Claim: For the selected vertex v :

✓ $\mathbf{dist}(s, v) = \min_{u \in R} (d(u) + w(u, v))$

- ✓ Initialization
- ✓ **Maintenance**
- ✓ Termination

Question 1 @ VisuAlgo online quiz

Who is the **Master of Algorithms** pictured below?

- Stephen Cook
- Edsger Dijkstra
- Robert Tarjan
- Avi Wigderson



Efficiency

Dijkstra($G = (V, E), s \in V$)

- $d(s) = 0$
- $R = \{s\}$
- **While** $R \neq V$
 - Select $v \in V \setminus R$ to minimize $\min_{u \in R} (d(u) + w(u, v))$
 - $d(v) = \min_{u \in R} (d(u) + w(u, v))$
 - Add v to R

Efficiency

Dijkstra($G = (V, E), s \in V$)

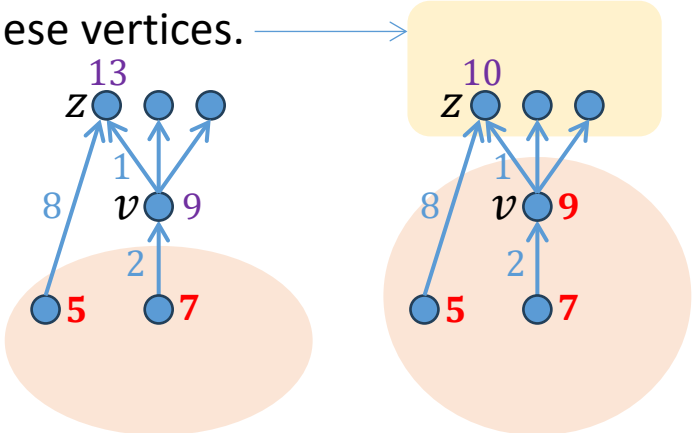
- $d(s) = 0$
- $R = \{s\}$
- **While** $R \neq V$
 - Select $v \in V \setminus R$ to minimize $\min_{u \in R} (d(u) + w(u, v))$
 - $d(v) = \min_{u \in R} (d(u) + w(u, v))$
 - Add v to R

Observation: No need to compute $x(v) = \min_{u \in R} (d(u) + w(u, v))$ from scratch for every iteration.

Adding v to R can only affect the x -value of these vertices.

- $x(z) \leftarrow \min\{x(z), d(v) + w(v, z)\}$

10 13 9 1



Efficiency

Dijkstra($G = (V, E), s \in V$)

- $d(s) = 0$
- $R = \{s\}$
- **While** $R \neq V$
 - Select $v \in V \setminus R$ to minimize $\min_{u \in R} (d(u) + w(u, v))$
 - $d(v) = \min_{u \in R} (d(u) + w(u, v))$
 - Add v to R

Observation: No need to compute $x(v) = \min_{u \in R} (d(u) + w(u, v))$ from scratch for every iteration.

Adding v to R can only affect the x -value of these vertices. \longrightarrow

- $x(z) \leftarrow \min\{x(z), d(v) + w(v, z)\}$
- 10
13
9
1

Dijkstra($G = (V, E), s \in V$)

- $R = \emptyset$
- $x(v) = \infty$ for all $v \in V \setminus \{s\}$
- $x(s) = 0$
- **While** $R \neq V$
 - Select $v \in V \setminus R$ to minimize $x(v)$
 - $d(v) = x(v)$
 - Add v to R
 - For all $z \in V \setminus R$ such that $(v, z) \in E$
 - $x(z) = \min\{x(z), d(v) + w(v, z)\}$

Efficiency

$n = |V|$ is the number of vertices.
 $m = |E|$ is the number of edges.

- The algorithm can be implemented using a **priority queue**.

n insert

n extract-min

At most m decrease-key

Dijkstra($G = (V, E), s \in V$)

- $R = \emptyset$
- $x(v) = \infty$ for all $v \in V \setminus \{s\}$
- $x(s) = 0$
- **While** $R \neq V$
 - Select $v \in V \setminus R$ to minimize $x(v)$
 - $d(v) = x(v)$
 - Add v to R
 - For all $z \in V \setminus R$ such that $(v, z) \in E$
 - $x(z) = \min\{x(z), d(v) + w(v, z)\}$

Efficiency

$n = |V|$ is the number of vertices.
 $m = |E|$ is the number of edges.

- The algorithm can be implemented using a **priority queue**.

| | Insert | Extract-min | Decrease-key | Time complexity of Dijkstra's algorithm |
|----------------|-------------|-----------------------|------------------|---|
| Binary heap | $O(\log n)$ | $O(\log n)$ | $O(\log n)$ | $O((m + n) \log n)$ |
| Fibonacci heap | $O(1)$ | $O(\log n)$ amortized | $O(1)$ amortized | $O(m + n \log n)$ |

https://en.wikipedia.org/wiki/Priority_queue

Amortized = average over n operations.

n insert

n extract-min

At most m decrease-key

Dijkstra($G = (V, E), s \in V$)

- $R = \emptyset$
- $x(v) = \infty$ for all $v \in V \setminus \{s\}$
- $x(s) = 0$
- While** $R \neq V$
 - Select $v \in V \setminus R$ to minimize $x(v)$
 - $d(v) = x(v)$
 - Add v to R
 - For all $z \in V \setminus R$ such that $(v, z) \in E$
 - $x(z) = \min\{x(z), d(v) + w(v, z)\}$

Efficiency

$n = |V|$ is the number of vertices.
 $m = |E|$ is the number of edges.

- The algorithm can be implemented using a **priority queue**.

| | Insert | Extract-min | Decrease-key | Time complexity of Dijkstra's algorithm |
|----------------|-------------|-----------------------|------------------|---|
| Binary heap | $O(\log n)$ | $O(\log n)$ | $O(\log n)$ | $O((m + n) \log n)$ |
| Fibonacci heap | $O(1)$ | $O(\log n)$ amortized | $O(1)$ amortized | $O(m + n \log n)$ |

↑
Is any further improvement possible?

- This is already as fast as sorting!

↑
Intuitively, the $O(n \log n)$ term comes from the fact that the algorithm sorts all vertices according to their distances.

Efficiency

$n = |V|$ is the number of vertices.
 $m = |E|$ is the number of edges.

- The algorithm can be implemented using a **priority queue**.

| | Insert | Extract-min | Decrease-key | Time complexity of Dijkstra's algorithm |
|----------------|-------------|-----------------------|------------------|---|
| Binary heap | $O(\log n)$ | $O(\log n)$ | $O(\log n)$ | $O((m + n) \log n)$ |
| Fibonacci heap | $O(1)$ | $O(\log n)$ amortized | $O(1)$ amortized | $O(m + n \log n)$ |

Breaking the Sorting Barrier for Directed Single-Source Shortest Paths

Ran Duan, Jiayi Mao, Xiao Mao, Xinkai Shu, and Longhui Yin

STOC 2025 Best Paper

<https://arxiv.org/abs/2504.17033>

$O(m \log^{2/3} n)$

Story behind the work: <https://www.quantamagazine.org/new-method-is-the-fastest-way-to-find-the-best-routes-20250806/>

Is any further improvement possible?

- This is already as fast as sorting!

Question 2 @ VisuAlgo online quiz

$$O(m \log^{2/3} n)$$

$$O(m + n \log n)$$

- The new shortest path algorithm is faster than Dijkstra's algorithm
 - when $m \in o(n \log n)$.
 - when $m \in o(n \log^{2/3} n)$.
 - when $m \in o(n \log^{1/3} n)$.
 - for all m .

Recursive algorithms

Fib(n)

- If $n \leq 1$, return n .
- Else, return **Fib**($n - 1$) + **Fib**($n - 2$).

- **Goal:** For a given algorithm \mathcal{A} , prove that it is correct.
- Analysis of a recursive algorithm:
 - **Base case:** without any recursive calls
 - Show that the algorithm is correct for the base case.
 - **Inductive step:** with recursive calls
 - Assume that the algorithm is correct for any input of size smaller than n .
 - Show that the algorithm is correct for any input of size n .

Recursive algorithms

Fib(n)

- If $n \leq 1$, return n .
- Else, return **Fib**($n - 1$) + **Fib**($n - 2$).

- **Goal:** For a given algorithm \mathcal{A} , prove that it is correct.

- Analysis of a recursive algorithm:

✓ **Base case:**

- Show that the algorithm is correct for the base case.

• **Inductive step:**

- Assume that the algorithm is correct for any input of size smaller than n .
- Show that the algorithm is correct for any input of size n .

If $n \leq 1$, the algorithm is correct:

- **Fib**(0) = 0
- **Fib**(1) = 1

Recursive algorithms

Fib(n)

- If $n \leq 1$, return n .
- Else, return **Fib**($n - 1$) + **Fib**($n - 2$).

- **Goal:** For a given algorithm \mathcal{A} , prove that it is correct.

- Analysis of a recursive algorithm:

✓ **Base case:**

- Show that the algorithm is correct for the base case.

✓ **Inductive step:**

- Assume that the algorithm is correct for any input of size smaller than n .
- Show that the algorithm is correct for any input of size n .

If $n \leq 1$, the algorithm is correct:

- **Fib**(0) = 0
- **Fib**(1) = 1

If $n > 1$, the algorithm is correct:

- **Fib**(n) = **Fib**($n - 1$) + **Fib**($n - 2$)

Searching in a sorted array

- **Input:**

- A sorted array A .
- Two indices:
 - lb (lower bound)
 - ub (upper bound)
- A number x .

- **Goal:**

- Decide if $x \in A[lb..ub]$

If $(lb > ub)$, the answer is **NO**.

Searching in a sorted array

- **Input:**

- A sorted array A .
- Two indices:
 - lb (lower bound)
 - ub (upper bound)
- A number x .

BinarySearch(A, lb, ub, x)

- **If** ($lb > ub$), return **NO**.
- **Else**
 - $mid \leftarrow \left\lfloor \frac{lb+ub}{2} \right\rfloor$.
 - **If** ($x = A[mid]$), return **YES**.
 - **If** ($x > A[mid]$), **BinarySearch**($A, mid + 1, ub, x$).
 - **If** ($x < A[mid]$), **BinarySearch**($A, lb, mid - 1, x$).

- **Goal:**

- Decide if $x \in A[lb..ub]$

Searching in a sorted array

$x = 14$

mid = 5
[2 7 14 33 41 50 77 80 82]
lb = 1 ub = 9



mid = 2
[2 7 14 33 41 50 77 80 82]
lb = 1 ub = 4



mid = 3
[2 7 14 33 41 50 77 80 82]
lb = 3 ub = 4

YES

BinarySearch(A, lb, ub, x)

- **If** ($lb > ub$), return **NO**.
- **Else**
 - $mid \leftarrow \lfloor \frac{lb+ub}{2} \rfloor$.
 - **If** ($x = A[mid]$), return **YES**.
 - **If** ($x > A[mid]$), **BinarySearch**($A, mid + 1, ub, x$).
 - **If** ($x < A[mid]$), **BinarySearch**($A, lb, mid - 1, x$).

Proof of correctness

- **Induction** on the array size:
 - $n = ub - lb + 1$

BinarySearch(A, lb, ub, x)

- **If** ($lb > ub$), return **NO**.
- **Else**
 - $mid \leftarrow \left\lfloor \frac{lb+ub}{2} \right\rfloor$.
 - **If** ($x = A[mid]$), return **YES**.
 - **If** ($x > A[mid]$), **BinarySearch**($A, mid + 1, ub, x$).
 - **If** ($x < A[mid]$), **BinarySearch**($A, lb, mid - 1, x$).

Proof of correctness

- **Induction** on the array size:

- $n = \text{ub} - \text{lb} + 1$

- **Base case:**

- If $n < 1$, the algorithm correctly returns **NO**.

BinarySearch($A, \text{lb}, \text{ub}, x$)

- **If** ($\text{lb} > \text{ub}$), return **NO**.

- **Else**

- $\text{mid} \leftarrow \left\lfloor \frac{\text{lb} + \text{ub}}{2} \right\rfloor$.

- **If** ($x = A[\text{mid}]$), return **YES**.

- **If** ($x > A[\text{mid}]$), **BinarySearch**($A, \text{mid} + 1, \text{ub}, x$).

- **If** ($x < A[\text{mid}]$), **BinarySearch**($A, \text{lb}, \text{mid} - 1, x$).

Proof of correctness

- **Induction** on the array size:

- $n = \text{ub} - \text{lb} + 1$

- **Inductive step:**

- Assume the algorithm works correctly for any input of size smaller than n .

BinarySearch($A, \text{lb}, \text{ub}, x$)

- **If** ($\text{lb} > \text{ub}$), return **NO**.

- **Else**

- $\text{mid} \leftarrow \left\lfloor \frac{\text{lb} + \text{ub}}{2} \right\rfloor$.

- **If** ($x = A[\text{mid}]$), return **YES**.

- **If** ($x > A[\text{mid}]$), **BinarySearch**($A, \text{mid} + 1, \text{ub}, x$).

- **If** ($x < A[\text{mid}]$), **BinarySearch**($A, \text{lb}, \text{mid} - 1, x$).

If ($x = A[\text{mid}]$), the answer must be **YES**.

If ($x > A[\text{mid}]$), then:

- ($x \in A[\text{lb}.. \text{ub}]$) if and only if ($x \in A[\text{mid} + 1.. \text{ub}]$). *A is sorted*
- Therefore, the answer must be **BinarySearch**($A, \text{mid} + 1, \text{ub}, x$). *Induction hypothesis*

The case of ($x < A[\text{mid}]$) is similar.

Efficiency

- Input size:
 - $n = ub - lb + 1$
- Subproblem input size:
 - $\leq \lfloor n/2 \rfloor$
- Depth of recursion:
 - $O(\log n)$
- Time complexity:
 - $T(n) \in O(\log n)$

BinarySearch(A, lb, ub, x)

- **If** ($lb > ub$), return **NO**.
- **Else**
 - $mid \leftarrow \lfloor \frac{lb+ub}{2} \rfloor$.
 - **If** ($x = A[mid]$), return **YES**.
 - **If** ($x > A[mid]$), **BinarySearch**($A, mid + 1, ub, x$).
 - **If** ($x < A[mid]$), **BinarySearch**($A, lb, mid - 1, x$).

Acknowledgement

- The slides are modified from previous editions of this course and similar course elsewhere.
- **List of credits:**
 - Diptarka Chakraborty
 - Yi-Jun Chang
 - Erik Demaine
 - Steven Halim
 - Sanjay Jain
 - Wee Sun Lee
 - Charles Leiserson
 - Hon Wai Leong
 - Warut Sukhompong
 - Wing-Kin Sung