CS3230 – Design and Analysis of Algorithms

(S1 AY2024/25)

Lasture 3a: Prast of Carrathees (S1 AY2024/25)

Lecture 3a: Proof of Correctness

Correctness of an algorithm

- Goal: For a given algorithm A , prove that it is correct.
	- Iterative algorithms.
	- Recursive algorithms.

$\textbf{Fib}(n)$

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

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

Iterative algorithms

- Do $\{$ some work $\}$
- Goal: For a given algorithm 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

- Do $\{$ some work $\}$
- Goal: For a given algorithm 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

- Do $\{$ some work $\}$
- Goal: For a given algorithm 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.

the current iteration.

Iterative algorithms

- Do $\{$ some work $\}$
- Goal: For a given algorithm 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
	- Termination:
		- Loop invariant at the end of the last iteration \rightarrow The algorithm outputs a correct answer.

Iterative algorithms

- Do $\{$ some work $\}$
- Goal: For a given algorithm A , prove that it is correct.
- Analysis of an iterative algorithm:
	- Loop invariant: $\left[$ Induction hypothesis $\right]$
		- 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.

• Goal: For a given algorithm A , prove that it is correct. *A*, prove that it is correct.

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

• **Fib**(0) = 0

• **Fib**(1) = 1 *A*, prove that it is correct.

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

• Fib(0) = 0

• Fib(1) = 1

 $IFib(n)$

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

-
-

• Goal: For a given algorithm A , prove that it is correct. **• prove that it is correct.**

consider the case of $n \ge 2$.
 invariant:
 : the start of iteration *i*,

• prev2 = $\text{Fib}(i - 2)$

• prev1 = $\text{Fib}(i - 1)$ **•** prove that it is correct.

consider the case of $n \ge 2$.
 invariant:
 the start of iteration *i***,

•** prev2 = **Fib**(*i* − 2)

• prev1 = **Fib**(*i* − 1) $\mathcal A$, prove that it is correct.
Now, consider the case of $n \geq 2$.
Loop invariant:
• At the start of iteration *i*,

 $IFib(n)$

- 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

Loop invariant:

- At the start of iteration i ,
	-
	-

• Goal: For a given algorithm A , prove that it is correct. **• prove that it is correct.**

consider the case of $n \ge 2$.
 invariant:
 : the start of iteration *i*,

• prev2 = $\text{Fib}(i - 2)$

• prev1 = $\text{Fib}(i - 1)$
 lization:
 : the start of iteration *i* = 2 **• prove that it is correct.**

consider the case of $n \ge 2$.
 invariant:
 • prev2 = **Fib**(*i* − 2)

• prev1 = **Fib**(*i* − 1)
 ilization:
 the start of iteration *i* **= 2,

• prev2 = Fib**(0) = 0 *A*, prove that it is correct.

Now, consider the case of $n \ge 2$.
 Loop invariant:

• At the start of iteration *i*,

• prev2 = $Fib(i - 2)$

• prev1 = $Fib(i - 1)$

Initialization:

• At the start of iteration *i* = 2,

• p $\mathcal A$, prove that it is correct.
Now, consider the case of $n \geq 2$.
Loop invariant:
• At the start of iteration *i*,

 $IFib(n)$

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

• invariant:

invariant:

• the start of iteration *i*,

• prev2 = Fib(*i* - 2)

• prev1 = Fib(*i* - 1)

lization:

• the start of iteration *i* = 2,

• prev2 = Fib(0) = 0

• prev1 = Fib(1) = 1

Loop invariant:

- At the start of iteration i .
	-
	-

Initialization:

- temp = prev1

 prev1 = prev1+prev2 At the start of iteration $i = 2$,

 prev2 = $\text{Fib}(0) = 0$
	-
	-

• Goal: For a given algorithm A , prove that it is correct. **• prove that it is correct.**

consider the case of $n \ge 2$.
 invariant:
 Consider the case of $n \ge 2$ **.**
 Consider the start of iteration *i***,

• prev2 = Fib(***i* **− 2)

• prev2 = Fib(***i* **− 2)

• prev2 = Fib(***i* **− 2)
 • prove that it is correct.**

consider the case of $n \ge 2$.
 invariant:
 the start of iteration *i***,

• prev2 = Fib(***i* **− 2)

• prev1 = Fib(***i* **− 1)

• prev1 = Fib(***i* **− 1)

• Then at the end of the iteration,

• pre** *A*, prove that it is correct.

Now, consider the case of $n \ge 2$.
 Loop invariant:

• At the start of iteration *i*,

• prev2 = $\text{Fib}(i - 2)$

• prev2 = $\text{Fib}(i - 1)$

• prev2 = $\text{Fib}(i - 1)$

• prev1 = $\text{Fib}(i - 1)$
 \mathcal{A} , prove that it is correct.

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

Loop invariant:

• At the start of iteration *i*,

• Suppose at the start of iteration *i*, given algorithm *A*, prove that it is correct.

Now, consider the case of *n* ≥ 2.

Now, consider the case of *n* ≥ 2.
 1.
 1.

- If $n \leq 1$
	- return n
- Else,
	- prev $2 = 0$
	- $prev1 = 1$

\n- \n**Goal:** For a given algorithm
$$
\mathcal{A}
$$
, prove that it is consistent with the image:\n
	\n- **q** f f f
	\n\n
\n- \n**q** f f
\n- \n**q** f
\n- \n**q** f
\n
\n\n- \n**q** f
\n- \n**q** f
\n\n\n- \n**q** f
\n- \n**q** f
\n- \n**q** f
\n\n\n- \n**q** f
\n- \n**q** f
\n

• return prev1

Loop invariant:

- At the start of iteration i .
	-
	-

Initialization:

- -
	-

Maintenance:

- invariant:

 Maintenance:

 the start of iteration *i*,

 prev2 = Fib(*i* 2)

 prev1 = Fib(*i* 1)

 prev1 = Fib(*i* 1)

 prev1 = Fib(*i* 1)

 Then at the end of the iteration,

 prev2 = Fib(*i* 1)

 • Suppose at the start of iteration i , **Example 3**
 example 3
 example 3
 example 3
 example 4
 tenance:
 expropenent in the start of iteration *i***,

• prev2 = Fib**(*i* - 2)

• prev1 = **Fib**(*i* - 1)

en at the end of the iteration,

• prev2 = **Fib**(*i* - 1)

• prev1 = **Fib**(*i*) **tenance:**

uppose at the start of iteration *i*,

• prev2 = **Fib**(*i* - 2)

• prev1 = **Fib**(*i* - 1)

• prev2 = **Fib**(*i* - 1)

• prev1 = **Fib**(*i*) **tenance:**

uppose at the start of iteration *i*,

• prev2 = **Fib**(*i* - 2)

• prev1 = **Fib**(*i* - 1)

en at the end of the iteration,

• prev2 = **Fib**(*i* - 1)

• prev1 = **Fib**(*i*)
	-
	-
	- Then at the end of the iteration,
		-
		-

• Goal: For a given algorithm A , prove that it is correct. **•** prove that it is correct.

consider the case of $n \ge 2$.
 invariant:
 Consider the case of $n \ge 2$ **.**
 Consider the start of iteration *i***,

• prev2 = Fib(***i* **− 2)

• prev2 = Fib(***i* **− 2)

• prev2 = Fib(***i* **− 2)
 • prove that it is correct.**

consider the case of $n \ge 2$.
 invariant:
 Consider the case of $n \ge 2$ **.**
 Consider the case of $n \ge 2$ **.**
 Consider the start of iteration *i***,

• prev2 =** $\text{Fib}(i - 2)$ **

• prev2 = \text{** *i A*, prove that it is correct.

Now, consider the case of $n \ge 2$.
 Loop invariant:

• At the start of iteration *i*,

• prev2 = $\text{Fib}(i - 2)$

• prev2 = $\text{Fib}(i - 1)$

• prev1 = $\text{Fib}(i - 1)$

• prev1 = $\text{Fib}(i -$ \mathcal{A} , prove that it is correct.

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

Loop invariant:

• At the start of iteration *i*,

• Suppose at the start of iteration *i*,

 $IFib(n)$

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

Loop invariant:

- At the start of iteration i .
	-
	-

Initialization:

- -
	-

Maintenance:

- invariant:

 Maintenance:

 holder the case of $n \ge 2$ **.**

 holder interation i,

 prev2 = **Fib**($i 2$)

 prev2 = **Fib**($i 1$)

 prev1 = **Fib**($i 1$)

 prev1 = **Fib**($i 1$)

 holder interation, • Suppose at the start of iteration i , **Example 3**
 example 3
 example 3
 example 3
 example 4
 tenance:
 expropenent in the start of iteration *i***,

• prev2 = Fib**(*i* - 2)

• prev1 = **Fib**(*i* - 1)

en at the end of the iteration,

• prev2 = **Fib**(*i* - 1)

• prev1 = **Fib**(*i*) **tenance:**

uppose at the start of iteration *i*,

• prev2 = **Fib**(*i* - 2)

• prev1 = **Fib**(*i* - 1)

• prev2 = **Fib**(*i* - 1)

• prev2 = **Fib**(*i* - 1)

• prev1 = **Fib**(*i*)

ination:

• algorithm returns **Fib**(*n*) **tenance:**

uppose at the start of iteration *i*,

• prev2 = **Fib**(*i* - 2)

• prev1 = **Fib**(*i* - 1)

en at the end of the iteration,

• prev2 = **Fib**(*i* - 1)

• prev1 = **Fib**(*i*)

ination:

ne algorithm returns **Fib**(Pration i ,
 $i = n$
Pration,
	-
	-
	- \bullet Then at the end of the iteration.
		-
		-

Termination:

• The algorithm returns $\text{Fib}(n)$.

Question 1 @ VisuAlgo online quiz

- What is a suitable loop invariant to analyze this sorting algorithm? at the start of iteration i
	- \bullet A is sorted.
	- $A[1..i-1]$ is sorted.
	- $x \leq y$ for all $x \in A[1..j-1]$ and $y \in A[j..n]$.
	- $(A[1..j-1]$ is sorted) and $(x \leq y$ for all $x \in A[1..j-1]$ and $y \in A[j..n]$.

SelectionSort $(A[1..n])$

- For $j = 1$ to $n 1$
	- Select $s \in \{j, j + 1, ..., n\}$ so that $A[s]$ is a smallest number in $A[j..n]$.
	- Swap $A[j]$ and $A[s]$.

VisuAlgo (Selection sort): https://visualgo.net/en/sorting?mode=Selection

Weighted directed graphs Weighted directed graphs
• Let $G = (V, E)$ be a <u>directed</u> graph.
• Each edge $e \in E$ has a <u>positive weight</u> $w(e)$.
• If $(u, v) \in E$, then $w(u, v) = w(e)$.

-
- Each edge $e \in E$ has a positive weight $w(e)$.
	- If $(u, v) \in E$, then $w(u, v) = w(e)$.
	- If $(u, v) \notin E$, then $w(u, v) = \infty$.

Single-source shortest paths Single-source shortest paths
• Let $G = (V, E)$ be a <u>directed</u> graph.
• Each edge $e \in E$ has a <u>positive weight</u> $w(e)$.
• If $(u, v) \in E$, then $w(u, v) = w(e)$.

-
- Each edge $e \in E$ has a positive weight $w(e)$.
	- If $(u, v) \in E$, then $w(u, v) = w(e)$.
	- 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. *e*).
shortest-path distance from
dist(*s*, *s*) = 0
dist(*s*, *u*) = 1
dist(*s*, *v*) = 2
dist(*s*, *w*) = 3 *e*).
shortest-path distance from
dist(*s*, *s*) = 0
dist(*s*, *u*) = 1
dist(*s*, *v*) = 2
dist(*s*, *w*) = 3 *e*).
shortest-path distance from
dist(*s*, *s*) = 0
dist(*s*, *u*) = 1
dist(*s*, *v*) = 2
dist(*s*, *w*) = 3 *e*).
shortest-path distance from
dist(*s*, *s*) = 0
dist(*s*, *u*) = 1
dist(*s*, *v*) = 2
dist(*s*, *w*) = 3

Dijkstra's algorithm

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

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

VisuAlgo (Dijkstra): https://visualgo.net/en/sssp?slide=7

VisuAlgo (Dijkstra): https://visualgo.net/en/sssp?slide=7

VisuAlgo (Dijkstra): https://visualgo.net/en/sssp?slide=7

VisuAlgo (Dijkstra): https://visualgo.net/en/sssp?slide=7

- At the start of an iteration:
	-

The computed distances are correct.

Proof of correctness $\cdot \forall u \in R, d(u) = \text{dist}(s, u)$

VisuAlgo (Dijkstra): https://visualgo.net/en/sssp?slide=7

- At the start of an iteration:
	-

invariant:

• the start of an iteration:

• $\forall u \in R, d(u) = \textbf{dist}(s, u)$ distances are correct. The computed distances are correct. st(s, u) $\left\{\n\begin{array}{l}\n\text{the computed} \\
\text{distances are correct.}\n\end{array}\n\right\}$ $\sqrt{\text{Initialization}}$

Maintenance $\sqrt{\text{Termination}}$

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

Proof of correctness $\cdot \forall u \in R, d(u) = \text{dist}(s, u)$

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

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

 $\sqrt{}$ Initialization

Maintenance

$\sqrt{}$ Termination

- At the start of an iteration:
	-

The computed distances are correct.

Proof of correctness $\cdot \forall u \in R, d(u) = \text{dist}(s, u)$

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

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

•
$$
d(v) = \min_{u \in R} (d(u) + w(u, v))
$$

• Add
$$
v
$$
 to R

invariant:

• $\forall u \in R$, $d(u) = \textbf{dist}(s, u)$ The computed

• $\forall u \in R$, $d(u) = \textbf{dist}(s, u)$ distances are correct. **Claim:** For the selected vertex v : iteration:
 u) = **dist**(*s*, *u*) distances are correct.
 Claim: For the selected vertex *v*:
 dist(*s*, *v*) = $\min_{u \in R} (d(u) + w(u, v))$
 \checkmark Initialization • dist(s, v) = $\min_{u \in R} (d(u) + w(u, v))$ $\begin{array}{c} \textnormal{computed} \ \textnormal{inces are correct.} \ \end{array}$
 $\begin{array}{c} \textnormal{d vertex v:} \ \hline \begin{array}{c} d(u) + w(u,v) \end{array} \end{array}$

- At the start of an iteration:
	-

The computed distances are correct.

Proof of correctness $\cdot \forall u \in R, d(u) = \text{dist}(s, u)$

• $d(s) = 0$

•
$$
R = \{s\}
$$

- While $R \neq V$
	- Select $v \in V \setminus R$ to minimize $\min_{v \in R} (d(u))$ $u \in R$

•
$$
d(v) = \min_{u \in R} (d(u) + w(u, v))
$$

• Add
$$
v
$$
 to R

invariant:

• $\forall u \in R$, $d(u) = \textbf{dist}(s, u)$ The computed

• $\forall u \in R$, $d(u) = \textbf{dist}(s, u)$ distances are correct. **Claim:** For the selected vertex v : iteration:
 u) = **dist**(*s*, *u*) distances are correct.
 Claim: For the selected vertex *v*:
 dist(*s*, *v*) = $\min_{u \in R} (d(u) + w(u, v))$
 \checkmark Initialization • dist(s, v) = $\min_{u \in R} (d(u) + w(u, v))$ $\begin{array}{c} \textnormal{computed} \ \textnormal{inces are correct.} \ \end{array}$
 $\begin{array}{c} \textnormal{d vertex v:} \ \hline \begin{array}{c} d(u) + w(u,v) \end{array} \end{array}$

- At the start of an iteration:
	-

The computed distances are correct.

Proof of correctness $\cdot \forall u \in R, d(u) = \text{dist}(s, u)$

• $d(s) = 0$

•
$$
R = \{s\}
$$

- While $R \neq V$
	- Select $v \in V \setminus R$ to minimize $\min_{v \in R} (d(u))$ $u \in R$

•
$$
d(v) = \min_{u \in R} (d(u) + w(u, v))
$$

• Add
$$
v
$$
 to R


```
\sqrt{} Termination
```


- At the start of an iteration:
	-

The computed distances are correct.

Proof of correctness $\cdot \forall u \in R, d(u) = \text{dist}(s, u)$

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

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

•
$$
d(v) = \min_{u \in R} (d(u) + w(u, v))
$$

• Add
$$
v
$$
 to R

invariant:

• $\forall u \in R$, $d(u) = \textbf{dist}(s, u)$ The computed

• $\forall u \in R$, $d(u) = \textbf{dist}(s, u)$ distances are correct. **Claim:** For the selected vertex v : iteration:
 u) = **dist**(*s*, *u*) distances are correct.
 Claim: For the selected vertex *v*:
 $\sqrt{\textbf{dist}(s, v)} = \min_{u \in R} (d(u) + w(u, v))$
 $\sqrt{\text{Initialization}}$ \checkmark dist(s, v) = $\min_{u \in R} (d(u) + w(u, v))$ iteration:
 u) = **dist**(*s*, *u*) distances are correct.
 Claim: For the selected vertex *v*:
 $\sqrt{\textbf{dist}(s, v)} = \min_{u \in R} (d(u) + w(u, v))$
 $\sqrt{\textbf{Initialization}}$

uestion 2 @ VisuAlgo online
ho is the **Master of Algorithms** pion
• Stephen Cook
• Edsger Dijkstra
• Robert Tarjan Question 2 @ VisuAlgo online quiz
Who is the **Master of Algorithms** pictured below?

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

- Stephen Cook
-
- Robert Tarjan
- Avi Wigderson

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

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

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

•
$$
d(v) = \min_{u \in R} (d(u) + w(u, v))
$$

• Add
$$
v
$$
 to R

- $d(s) = 0$
- $R = \{s\}$
- While $R \neq V$
- Efficiency

Dijkstra(G = (V, E), s ∈ V)

 $d(s) = 0$

 $R = \{s\}$

 While $R \neq V$

 Select $v \in V \setminus R$ to minimize $\min(d(u) + w(u, v))$

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

 Add v to R

Dijkstra(G = (V, E), s ∈ V)

 $d(s) = 0$ for al **THICIENCY**
 Dijkstra(*G* = (*V*, *E*), *s* ∈ *V*)

• $d(s) = 0$

• $R = \{s\}$

• While $R \neq V$

• Select $v \in V \setminus R$ to minimize $\min_{u \in R} (d(u) + w(u))$

• $d(v) = \min_{u \in R} (d(u) + w(u, v))$

• Add v to *R*

• $d(s) = 0$

• $d(s) = 0$ for **Dijkstra**(*G* = (*V*, *E*), *s* ∈ *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
 Dijkstra(*G* = (*V*, *E*), *s* ∈ *V*) **Dijkstra** $(G = (V, E), s \in V)$

• $d(s) = 0$

• $R = \{s\}$

• While $R \neq V$

• Select $v \in V \setminus R$ to minimize $\min_{u \in R} (d(u) +$

• $d(v) = \min_{u \in R} (d(u) + w(u, v))$

• Add v to R
 Dijkstra $(G = (V, E), s \in V)$

• $d(s) = 0$

• $x(v) = \infty$ for al **Dijkstra**(*G* = (*V*, *E*), *s* ∈ *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*

• $f(v) = \min_{u \in R} (d(u) + w(u, v))$

• Add v (s) = 0

= {s}

/hile $R \neq V$

• Select $v \in V \setminus R$ to minimize $\min(d(u) + w(u, v))$

• $d(v) = \min_{u \in R} (d(u) + w(u, v))$

• Add v to R

rad(s = $(V, E), s \in V$)
 $s) = 0$
 $v) = \infty$ for all $v \in V \setminus \{s\}$

rall $z \in V$ such that $(s, z) \in E$

• • $a(s) = 0$

• $R = \{s\}$

• 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

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

• $d(s) = 0$

• $d(s) = 0$

• $R = \{s\}$

• For all $z \in V$ = {s}

Vhile R ≠ V

• Select *v* ∈ V \ 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
 $\text{tra}(G = (V, E), s \in V)$
 $s) = 0$
 er
 $s = \{s\}$
 er
 ar $z \in V$ such that $(s, z) \in E$

• x Thile $R \neq V$

• Select $v \in V \setminus R$ to minimize $\min(d(u) + w(u, v))$

• $d(v) = \min_{u \in R} (d(u) + w(u, v))$

• Add v to R
 $\text{trag}(G = (V, E), s \in V)$
 $\text{grad } g = (S)$
 $\text{grad } g = V \setminus \{s\}$
 $= (S)$

• $x(z) = \min\{x(z), d(s) + w(s, z)\}$

hile $R \neq V$

• $\text{det$ • Select $v \in V \setminus R$ to minimize $\min_{v \in R} (d(u))$ $u \in R$
	- $d(v) = \min_{u \in R} (d(u) + w(u, v))$
	- Add ν to R

-
-
-
- -
- -
	-
	- Add ν to R
	- For all $z \in V \setminus R$ such that $(v, z) \in E$

• $x(z) = min\{x(z), d(v) + w(v, z)\}$

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

• The algorithm can be implemented using a priority queue.

• The algorithm can be implemented using a priority queue.

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

```
example 3 are provided<br>
Example 3 are complexity of<br>
Example 3 are complexity of<br>
Example 3 are complexity by any of the sum of O(|E| + |V|) \log|V|<br>
<b>Dijkstra(G = (V, E), s \in V)<br>
Example 3 are contraded<br>
Example 3 are
                                                                                                                                                     cd using a priority queue.<br>
Decrease-key<br>
Time complexity of<br>
Dijkstra's algorithm<br>
O(\log n)<br>
O((|E| + |V|) \log|V|)<br>
O(1) amortized<br>
O(|E| + |V| \log|V|)<br>
Dijkstra(G = (V, E), s \in V)<br>
• R = \emptyset<br>
• x(v) = \infty for all v \in V \setminus \{s\}<br>
                                                                                                                                                     1 are the property of<br>
Decrease-key<br>
Example 2 Time complexity of<br>
Dijkstra's algorithm<br>
O(\log n)<br>
O((|E| + |V|) \log|V|)<br>
O(1) amortized<br>
O(|E| + |V| \log|V|)<br>
\sum_{R \in \emptyset}<br>
\sum_{x(s) = 0}^{\infty}<br>
\sum_{x(s) = 0}^{\infty}<br>
\sum_{x(s) = 0}Sing a priority queue.<br>
Decrease-key Time complexity of<br>
Dijkstra's algorithm<br>
O(\log n) O((|E| + |V|) \log|V|)<br>
(1) amortized O(|E| + |V| \log|V|)<br>
= \Phi<br>
= \Phi<br>
\psi) = ∞ for all \nu \in V \setminus \{s\}<br>
\mathbf{s}) = 0<br>
\mathbf{h} \in R \neq V<br>
                                                                                                                                                                   Becrease-key<br>
Time complexity of<br>
Dijkstra's algorithm<br>
O(\log n)<br>
O((|E| + |V|) \log|V|)<br>
(1) amortized<br>
O(|E| + |V| \log|V|)<br>
\lim_{v \to 0} \cos \theta<br>
= 0<br>
= 0<br>
\sinh R \neq V<br>
\therefore Select v \in V \setminus R to minimize x(v)<br>
= \frac{d(v)}{dvR}<br>
\• Add \nu to R• For all  ∈  ∖  such that ,  ∈ 
                                                                                                                                                                                • x(z) = min\{x(z), d(v) + w(v, z)\}\|V| insert \begin{vmatrix} \cdot & x(v) = \infty \text{ for all } v \in V \setminus \{s\} \\ \cdot & x(s) = 0 \end{vmatrix}|V| extract-min \begin{array}{|c|c|c|c|c|}\n\hline\n\text{•} & \text{Select } v \in V \setminus R \text{ to minimize } x(v) \\
\hline\n\text{•} & d(v) = x(v)\n\end{array}At most |E| decrease-key<br>At most |E| decrease-key<br>At most \log \log \log x and \log x(z) = \min\{x(z), d(y) + w(y) \}Amortized = average over n operations.<br>
Dijkstra(G = (V, E), s \in V)
```
Recursive algorithms $\begin{array}{c|c} \text{Fib}(n) \\ \cdot & \text{If } n \leq n \end{array}$

- If $n \leq 1$, return n.
- Else, return $\textbf{Fib}(n-1) + \textbf{Fib}(n-2)$.
- Goal: For a given algorithm 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 $\begin{array}{|c|c|c|} \hline \text{Fib}(n) & \text{Fib}(n) \hline \end{array}$

- If $n \leq 1$, return n.
- Else, return $\text{Fib}(n-1) + \text{Fib}(n-2)$.
- Goal: For a given algorithm A , prove that it is correct.
- Analysis of a recursive algorithm:
	- Base case:

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

- $\text{Fib}(0) = 0$
- $\textbf{Fib}(1)=1$ •
- 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 .

Recursive algorithms $\begin{array}{ccc} \begin{array}{c} \text{Fib}(n) \\ \cdot & \text{If } n \end{array} \end{array}$

- If $n \leq 1$, return n. Else, return $\textbf{Fib}(n-1) + \textbf{Fib}(n-2)$.
- Goal: For a given algorithm A , prove that it is correct.
- Analysis of a recursive algorithm:
	- Base case:

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

- $\text{Fib}(0) = 0$
- $\textbf{Fib}(1)=1$
- 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 $n > 1$, the algorithm is correct: $Fib(n) = Fib(n-1) + Fib(n-2)$ •

Searching in a sorted array • arching in a sorted array

• A <u>sorted</u> array A.

• Two indices:

• Ib (lower bound)

• ub (upper bound)

• Input:

-
- -
	-
- A number x .
- Goal:
	- Decide if $x \in A[lb..ub]$ If $(lb > ub)$, the answer is NO.

Searching in a sorted array **Example 12**
 Example 12
 Example 2

• Two indices:

• Ib (lower bound)

• ub (upper bound)

• up (upper bound)

• If (x = A[mi

• Input:

-
- -
	-
- A number x .

- If $(lb > ub)$, return NO.
- Else
	- mid \leftarrow $\left| \frac{\text{lb+ub}}{2} \right|$. .
	- 2 \int • 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 $(A, \text{lb}, \text{mid} 1, x)$.

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

Searching in a sorted array a sorted array

BinarySearch(A, lb, ub, x)

F(lb > ub), return NO.

Fise

F(x = A[mid]), return Y

F(x = A[mid]), return Y

F(x = A[mid]), Binary

F(x = A[mid]), Binary

F(x = A[mid]), Binary

F(x = A[mid]), Binary Searching in a sorted a
 $x = 14$
 $x = 14$
 $x = 14$
 $\frac{1}{2}$
 $\frac{$

- If $(lb > ub)$, return NO.
- Else

• mid
$$
\leftarrow \left[\frac{\text{lb+ub}}{2} \right]
$$
.

- 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: \int BinarySearch(A, lb, ub, x)

- $n = ub lb + 1$
• If $(lb > ub)$, return NO.
	- Else

• mid
$$
\leftarrow \left[\frac{\text{lb+ub}}{2} \right]
$$
.

- 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: \int BinarySearch(A, lb, ub, x)
	- $n = ub lb + 1$
- Base case:
	- If $n < 1$, the algorithm correctly returns NO.

- If $(lb > ub)$, return NO.
- Else

• mid
$$
\leftarrow \left[\frac{\text{lb} + \text{ub}}{2} \right]
$$
.

- 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: \int BinarySearch(A, lb, ub, x)
	- $n =$ ub lb + 1
- Inductive step:
- Assume the algorithm works correctly for any input of size smaller than n . **Induction** on the array size:

• $n = \text{ub} - \text{lb} + 1$

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

• Else

• mid ← $\frac{|\text{lb} > \text{ub}|}{2}$.

• Assume the algorithm works

• correctly for any input of size

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

- If $(lb > ub)$, return NO.
- Else

• mid
$$
\leftarrow \left[\frac{\text{lb+ub}}{2} \right]
$$
.

- 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 $(A, \text{lb}, \text{mid} 1, x)$.

- $(x \in A[lb..ub])$ if and only if $(x \in A[mid + 1..ub])$. A is sorted
- Induction hypothesis

- Input size:
	- $n = ub lb + 1$
- Subproblem input size: $\bullet \leq \lfloor n/2 \rfloor$ • $n = ub - lb + 1$

• If $(lb > ub)$, return NO.

• Else

• $\leq \lfloor n/2 \rfloor$

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

• If $(x > A[\text{mid}])$, BinarySear

• $O(log n)$

• T $(n) \in O(log n)$

• The complexity:
- Depth of recursion:
	- \bullet $O(log n)$
- Time complexity:
	-

BinarySearch (A, lb, ub, x)

-
- - mid $\leftarrow \left| \frac{\text{lb+ub}}{2} \right|$.
	- 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 $(A, \text{lb}, \text{mid} 1, x)$.

Acknowledgement

• The slides are modified from previous editions of this course and similar course elsewhere. **Example Channois Concernent**
 he slides are modified from previous editio
 milar course elsewhere.
 st of credits:

• Diptarka Chakraborty

• Yi-Jun Chang

• Erik Demaine

• Steven Halim

• List of credits:

-
- Yi-Jun Chang
- Erik Demaine
- Steven Halim
- Sanjay Jain
- Wee Sun Lee
- Charles Leiserson
- Hon Wai Leong
- Wing-Kin Sung