Lecture 03: Induction and Sorting

COSC 311 *Algorithms*, Fall 2022

Accountability Group Survey

Office Hours

1. Evening TA Sessions
    - Sunday, 5–7pm in SCCE A131
    - Wednesday, 7–9pm in SCCE A131
2. Drop-in Office Hours
    - Thursdays 11–12, 2:30–3:30
    - King & Wieland Quad, Tent 2 (South of SCCE)
3. Office Hours by Appointment, starting next week
    - Wednesdays 3–4pm
    - Another time too...

Today

1. Induction
2. Selection Sort, Revisited
3. Efficiency

Last Time

Selection Sort!

```
01  SelectionSort(a):
02    n <- size(a)
03    for j = 1 to n - 1 do
04      min <- j
05      for i = j+1 to n do
06        if compare(a, min, i)
07          min <- i
08        endif
09      endfor
10      swap(a, j, min)
11    endfor
```

Goal: at end of loop, min is index of minimum value in a[j...n]

Arguing Correctness

Goal. Logically deduce that algorithm succeeds on all inputs.

To do:

- specify task: sorting
- specify allowed operations and effects: compare, swap
- specify algorithm: pseudocode
- demonstrate that on all possible inputs, algorithm output satisfies task specification
  - use induction!

Induction

A Bit of Formalism

A **logical predicate** P is a statement that can be true or false

Examples.

- P = "it is raining today"
- P = "57 is a prime number"
- P = "the output of `SelectionSort([5,2,7,1])` is sorted"

Sequences of Predicates

A logical principle to establish the truth of a *sequence* of predicates

Example. a an array of numbers of size n:

- $P(1)$: $a[1] \leq a[2]$
- $P(2)$: $a[2] \leq a[3]$
- $P(3)$: $a[3] \leq a[4]$

If all are true

$a[1] \leq a[2] \leq a[3] \leq \ldots \leq a[n]$

...

- $P(n-1)$: $a[n-1] \leq a[n]$

Question. How does "sortedness" relate to the predicates $P(1), P(2), \ldots, P(n-1)$?

Principle of Induction

Setup. $P(1), P(2), P(3), \ldots$ a sequence of predicates

Hypotheses. Suppose:

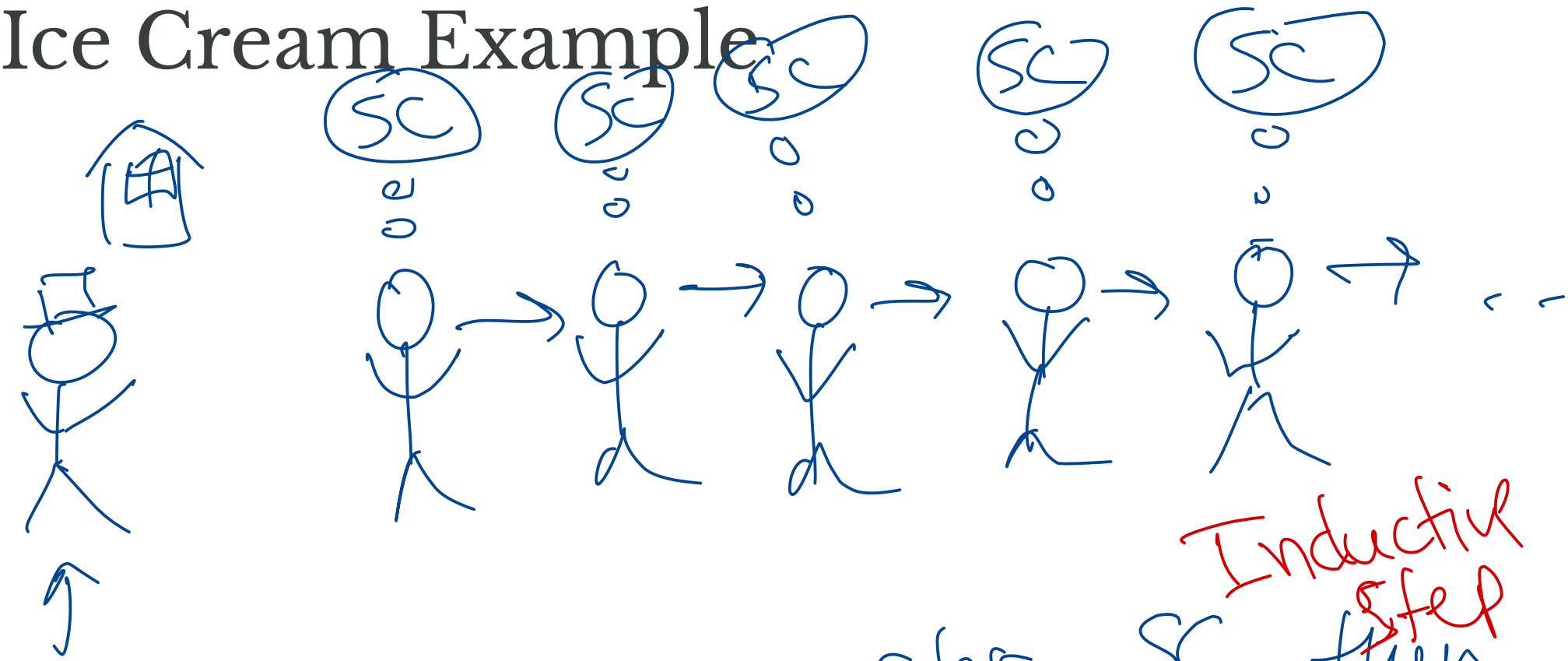
1. $P(1)$ is true (**base case**)
2. If $P(i)$ is true, then $P(i+1)$ is true (**inductive step**)

Conclusion. All of $P(1), P(2), \ldots$ are true

- "For all i, $P(i)$"
- logical notation: $\forall i, P(i)$

Ice Cream Example

Salted Caramel (SC)



Obs. If a person orders SC, then next person in line orders SC.

Inductive Step

Obs. First person in line orders SC.

Base

Selection Sort Pseudocode

Proposition. The output of `SelectionSort(a)` is sorted.

```
01  SelectionSort(a):
02    n <- size(a)
03    for i = 1 to n - 1 do
04      min <- i
05      for j = i+1 to n do
06        if compare(a, min, j)
07          min <- j
08        endif
09      endfor
10      swap(a, i, min)
11    endfor
12  end
```

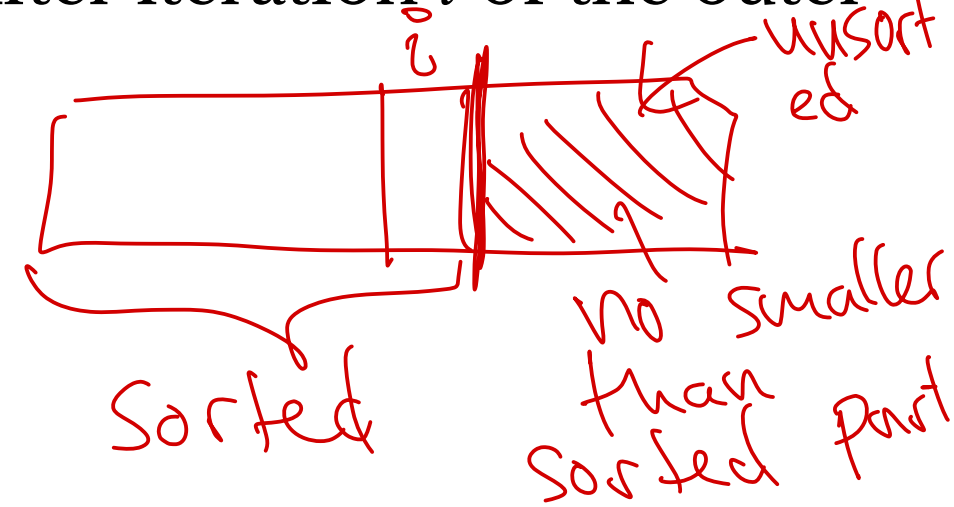
Induction and Selection Sort

```
01  SelectionSort(a):
02    n <- size(a)
03    for i = 1 to n - 1 do
04      min <- i
05      for j = i+1 to n do
06        if compare(a, min, j)
07          min <- j
08        endif
09      endfor
10      swap(a, i, min)
11    endfor
12  end
```

Claim 1 (inner loop invariant). After iteration j of the inner loop, `min` stores the index of the smallest value in $a[i..j]$.

Claim 2 (outer loop invariant). After iteration i of the outer loop,

1. $a[1..i]$ is sorted, and
2. for every $k > i$, $a[k] \geq a[i]$



Strategy

Claim 1 (inner loop invariant). After iteration j of the inner loop, `min` stores the index of the smallest value in $a[i..j]$.

Claim 2 (outer loop invariant). After iteration i of the outer loop,

1. $a[1..i]$ is sorted, and
2. for every $k > i$, $a[k] \geq a[i]$

When $i = n-1$, $a[1..n-1]$ is sorted and $a[n] \geq a[n-1]$

Argument overview:

1. Use induction to argue Claim 1.
2. Use Claim 1 and induction to argue Claim 2.
3. Correctness of SelectionSort follows from Claim 2.
   - why?

Induction and Claim 1

Claim 1 (inner loop invariant). After iteration j of the inner loop, `min` stores the index of the smallest value in $a[i..j]$.

```
04      min <- i
05      for j = i+1 to n do
06        if compare(a, min, j)
07          min <- j
08        endif
09      endfor
```

Use Induction:

- **Base Case.** Show Claim 1 holds for $j = i + 1$
- **Inductive step.** Show that if Claim 1 holds for j, then it also holds for $j + 1$.

Conclusion: Claim 1 holds for all j

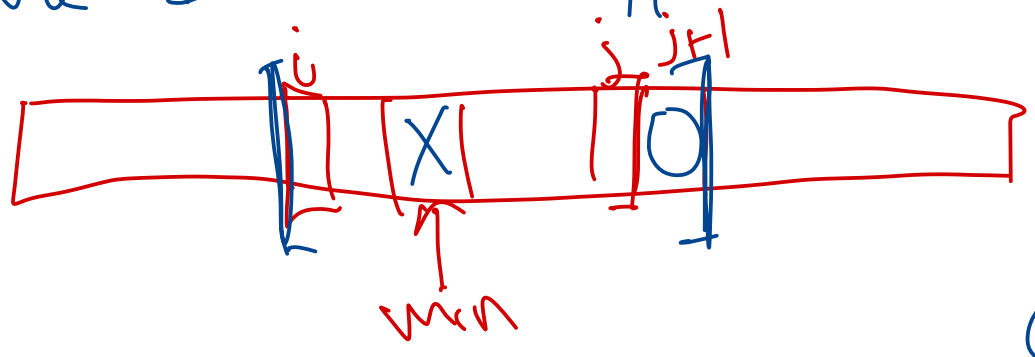
Proof of Claim 1

Claim 1 (inner loop invariant). After iteration j of the inner loop, `min` stores the index of the smallest value in $a[i..j]$.

```
04      min <- i
05      for j = i+1 to n do
06        if compare(a, min, j)
07          min <- j
08        endif
09      endfor
```

Proof of Base Case. Iteration $j = i+1$.
Consider two cases either $a[i] \leq a[i+1]$
→ no change in min ⇒ min = i
otherwise $a[i] > a[i+1]$ ⇒ change min to $i+1$

Proof of Inductive Step.
Assume Inductive hypothesis → claim is true for j



Case 1: $a[min] \leq$
⇒ line 7 not exec $a[j+1]$
Case 2: $a[j+1] < a[min]$
⇒ line 7 exec

Establishing Claim 2

```
01  SelectionSort(a):
02    n <- size(a)
03    for i = 1 to n - 1 do
04      min <- i
05      for j = i+1 to n do
06        if compare(a, min, j)
07          min <- j
08        endif
09      endfor
10      swap(a, i, min)
11    endfor
12  end
```

Claim 2 (outer loop invariant). After iteration i of the outer loop,

1. $a[1..i]$ is sorted, and
2. for every $k > i$, $a[k] \geq a[i]$

Argue by induction!

Apply Conclusion of Claim 1

```
01  SelectionSort(a):
02    n <- size(a)
03    for i = 1 to n - 1 do
..      min <- index of smallest value in a[i..n]
10      swap(a, i, min)
11    endfor
12  end
```

← true by Claim 1

Claim 2 (outer loop invariant). After iteration i of the outer loop,

1. $a[1..i]$ is sorted, and → true b/c every array of size 1 is sorted!
2. for every $k > i$, $a[k] \geq a[i]$

Base case. Show for $i=1$

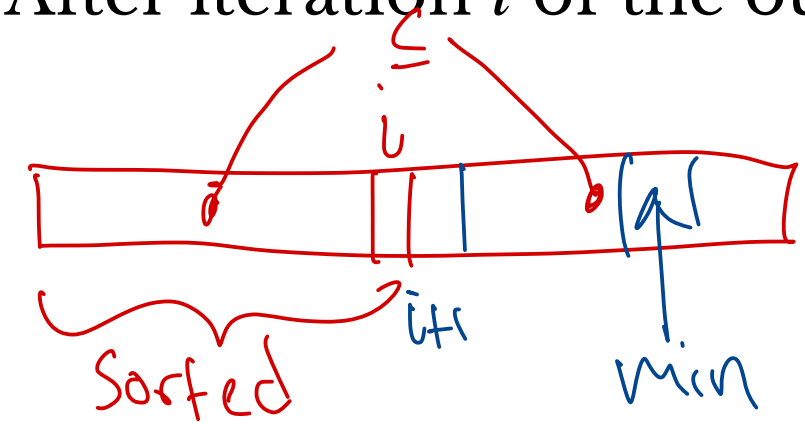
↳ follows from swap in line 10 and def. of min value.

Apply Conclusion of Claim 1

```
01  SelectionSort(a):
02    n <- size(a)
03    for i = 1 to n - 1 do
..      min <- index of smallest value in a[i..n]
10      swap(a, i, min)
11    endfor
12  end
```

Claim 2 (outer loop invariant). After iteration i of the outer loop,

1. $a[1..i]$ is sorted, and
2. for every $k > i$, $a[k] \geq a[i]$



Inductive Step.

Assume claim true for iter. i.

Consider iter. i+1:

before line 10, min is index of min value in a[i+1..n]

After a[i+1] stores min value in a[i+1..n]

Conclusion

SelectionSort sorts every possible array!

Swap Efficiency?

How many swap operations does SelectionSort use? Could we do better?

```
01  SelectionSort(a):
02    n <- size(a)
03    for i = 1 to n - 1 do
04      min <- i
05      for j = i+1 to n do
06        if compare(a, min, j)
07          min <- j
08        endif
09      endfor
10      swap(a, i, min)
11    endfor
12  end
```

Compare Efficiency?

How many compare operations does SelectionSort use? Could we do better?

```
01  SelectionSort(a):
02    n <- size(a)
03    for i = 1 to n - 1 do
04      min <- i
05      for j = i+1 to n do
06        if compare(a, min, j)
07          min <- j
08        endif
09      endfor
10      swap(a, i, min)
11    endfor
12  end
```

Next Time

Sorting by “Divide and Conquer”

- MergeSort

To read: notes on “Big O” notation (forthcoming)