Lecture 02: Sorting and Induction

COSC 311 *Algorithms*, Fall 2022

Announcements

1. Accountability groups (message today)
2. Office hours
   - Evening TA sessions Sunday, Wednesday (TBD)
   - My drop-in: Thursday 11-12, 2-3 (?)
   - By appointment: TBD
3. Emails: subject includes [COSC 311]
4. Section enrollment
5. Lecture ticket reminder (read solutions!)

Today

1. Sorting Task
2. Insertion Sort
3. Induction

Task: Sorting

Input:

- Sequence a of n numbers
- e.g., $a = 17, 7, 5, 2, 3, 19, 5, 13$

Output:

- A *sorted* sequence s of same elements as a
  - s contains same elements with same multiplicities as a
  - $s_1 \leq s_2 \leq \cdots \leq s_n$
- e.g., $s = 2, 3, 5, 5, 7, 13, 17, 19$

So Far

Sorting task is underspecified!

- *Why?*
  - what are allowed ops.
  - how fast? (resources)
    space?
  - comparison
  - representation

So Far

Sorting task is underspecified!

- *Why?*

1. *representation*
2. *supported operations*

So Far

Sorting task is underspecified!

- *Why?*

1. *representation*
2. *supported operations*

Examples:

- stack of exams
- array of numbers
- tasks by deadline

Each may support different operations & require different techniques to solve efficiently

Going Forward

Spend ~2 weeks on sorting

- Elementary algorithms — SelectionSort, InsertionSort, BubbleSort
  - argue correctness
    - mathematical induction
  - argue running time
    - big O notation
- Divide-and-conquer algorithms
  - algorithms: MergeSort, QuickSort, RadixSort
  - argue running time
    - "master method"

Sorting Arrays

Representation:

- a an array of size n
- $a[1], a[2], \dots, a[n]$

Supported Operations

- compare(a, i, j)
  - return true if $a[i] > a[j]$ and false otherwise
- swap(a, i, j)
  - before $a[i] = x$ and $a[j] = y$
  - after $a[i] = y$ and $a[j] = x$

Example

1 2 3 4 5 6 7 8

$a = [17, 7, 5, 2, 3, 19, 5, 13]$

- compare$(a, 2, 6)$? → false
- swap$(a, 2, 5)$? $a \rightarrow [17, 3, 5, 2, 7, 19, 5, 13]$

Central Tenet

Break a large task into smaller subtasks.

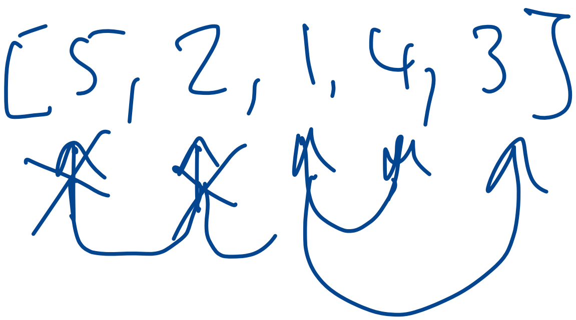
Lecture Ticket

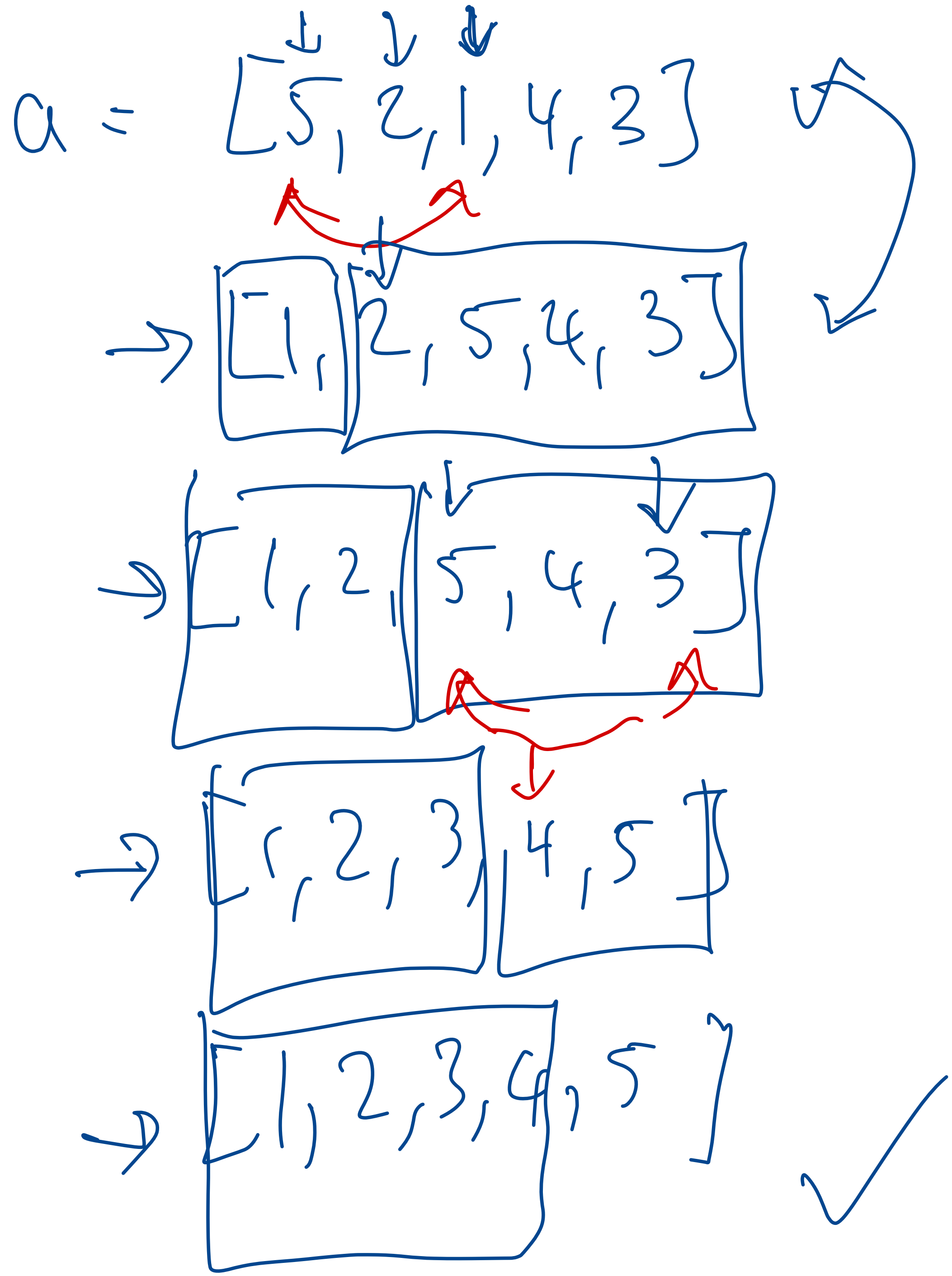
Express "selection sort" in pseudocode

- find smallest element and put it at index 1
- find second smallest element and put it at index 2
- find third smallest element and put it at index 3
- ...

Example

- Sorting a small array:





SelectionSort in Pseudocode

```
SelectionSort(a):
  n <- size(a)
  for j = 1 to n - 1 do
    min <- j
    for i = j+1 to n do
      if compare(a, min, i)
        min <- i
      endif
    endfor
    swap(a, j, min)
  endfor
```

true if a[min] > a[i]

min is index of min value in a[j..n]

Think about: does work for duplicate values?

Prove correctness mathematically?

Why does `SelectionSort` Work?

- find min value not yet sorted and puts it in right place

Each step succeeds <u>because</u> all previous steps succeeded

Arguing Correctness

Goal. Logically deduce that algorithm succeeds on all inputs.

To do:

- specify task
- specify allowed operations and effects
- specify algorithm
- demonstrate that on all possible inputs, algorithm output satisfies task specification

A Remark

It may be “obvious” to you that `SelectionSort` works.

- give *formal* analysis of algorithm here
- introduce tools that will help when things become less obvious

Specifying the Sorting Task

Input. Array a of numbers

Output. Sorted array s:

1. s contains the same elements as a
2. s is sorted: $s[1] \leq s[2] \leq \cdots \leq s[n]$
   - for every index $i < n$, $s[i] \leq s[i+1]$

always holds if all manipulations are swaps

Allowed Operations

- compare(a, i, j): return true if $a[i] > a[j]$
- swap(a, i, j):
  - before swap have $a[i] = x$ and $a[j] = y$
  - after swap have $a[i] = y$ and $a[j] = x$

Allowed Operations

- compare(a, i, j): return true if $a[i] > a[j]$
- swap(a, i, j):
  - before swap have $a[i] = x$ and $a[j] = y$
  - after swap have $a[i] = y$ and $a[j] = x$

Observation. If s is array formed from a by any sequence of swap operations, then s and a contain the same elements.

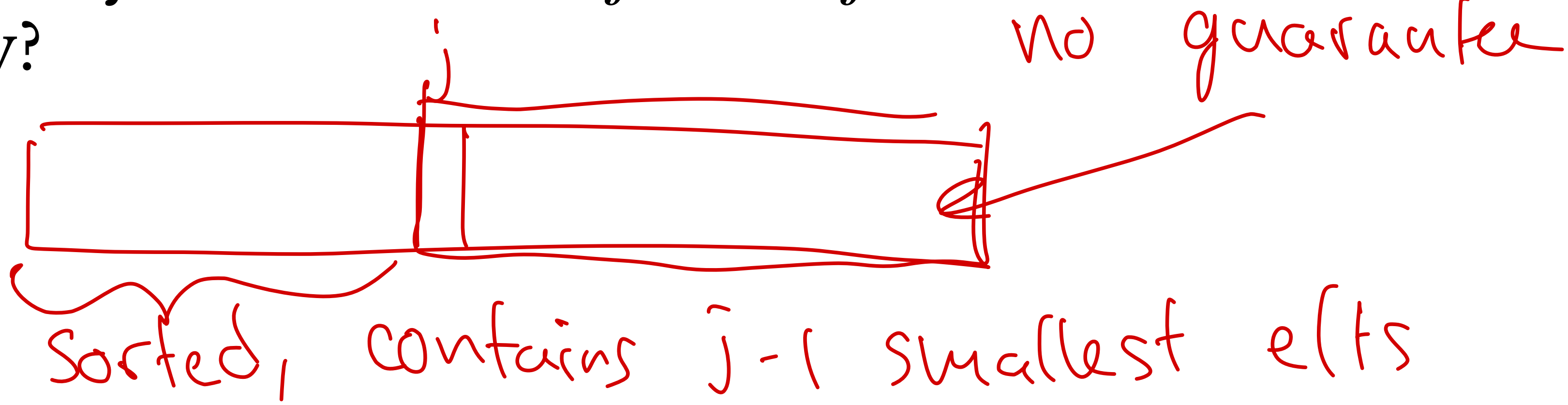
- Item (1) from sorting task is satisfied for any procedure that only modifies the array with swaps

Next Step

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

```
SelectionSort(a):
  n <- size(a)
  for j = 1 to n - 1 do
    min <- j
    for i = j+1 to n do
      if compare(a, min, i)
        min <- i
      endif
    endfor
    swap(a, j, min)
  endfor
end
```

Question. Why does iteration *j* select *j*th smallest element in the array?



Inductive Reasoning

Question. Why does iteration *j* select *j*th smallest element in the array?

```
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)
```

Reason. (informal)

1. Loop in lines 5-9 selects smallest value in `a[j..n]`
2. Previous steps moved smaller values to `a[1..j−1]`

Moral. Step `j` succeeds *because* steps `1, 2,...,j−1` succeeded

- *inductive reasoning*