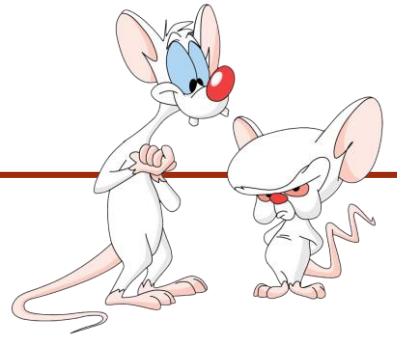
COMP 250

INTRODUCTION TO COMPUTER SCIENCE

Week 6 - 1 : Quadrat t

Giulia Alberini, Fall 2020

WHAT ARE WE GOING TO DO IN THIS VIDEO?



- How to sort a list
  - Bubble sort
  - Selection sort
  - Insertion sort

SORTING

- The process of arranging items in a ordered list following a given criterion.
- For example, sorting a list of integers in ascending order (from smallest to largest):

BEFO

SORTING ALGORITHMS

There are many techniques for sorting a list

- Selection Sort
- Bubble Sort
- Insertion Sort
- Random Sort :P
- Heap Sort
- Merge Sort
- Quick Sort

SORTING ALGORITHMS

There are many techniques for sorting a list

- Selection Sort
- Bubble Sort
- Insertion Sort

Check out how different algorithms compare:
https://www.youtube.com/watch?v=ZZuD6iUe3Pc

- Heap Sort
- Merge Sort
- Quick Sort

Later $O(N \cdot \log N)$

OBAMA KNOWS ABOUT SORTING!

https://www.youtube.com/watch?v=k4RRi_ntQc8

OBSERVATION

Today we are concerned with algorithms, not data structures.

The following algorithms are in f whether we use an array list or a linked list.

BUBBLE SORT

- Bubble sort is the simplest sorting algorithm.

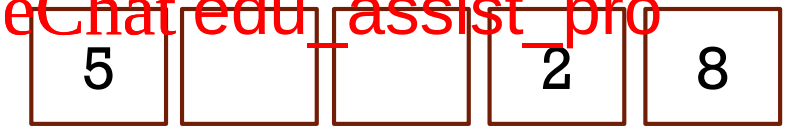
- Goal: order a list of int

- IDEA: repeatedly iterate through the list and swap adjacent elements if they are in the wrong order.

BUBBLE SORT – PSEUDOCODE

```
for i from 0 to list.length-1 {
   for j fr                    h -2 {
      if(li                          {
         swap(list[j],                ])
      }
   }
}
```

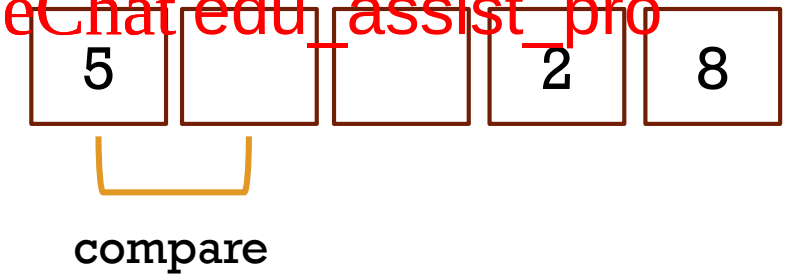
EXAMPLE – ONE ITERATION



EXAMPLE – ONE ITERATION

Iteration #1

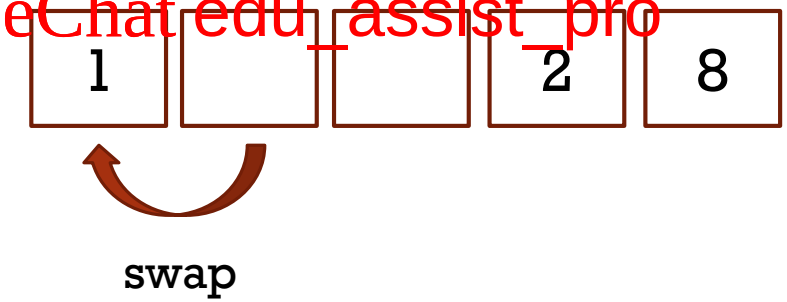
- Compare all adjacent elements.
- If needed, swap!



EXAMPLE – ONE ITERATION

Iteration #1

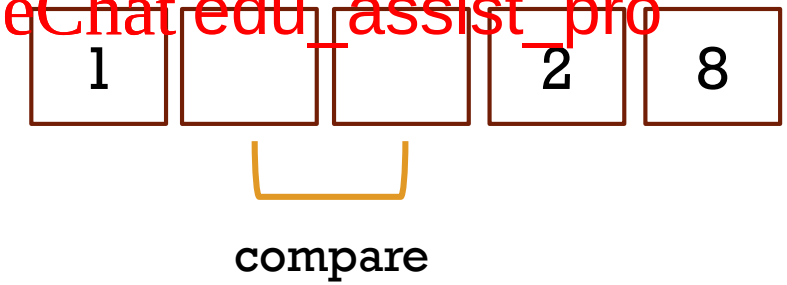
- Compare all adjacent elements.
- If needed, swap!



EXAMPLE – ONE ITERATION

Iteration #1

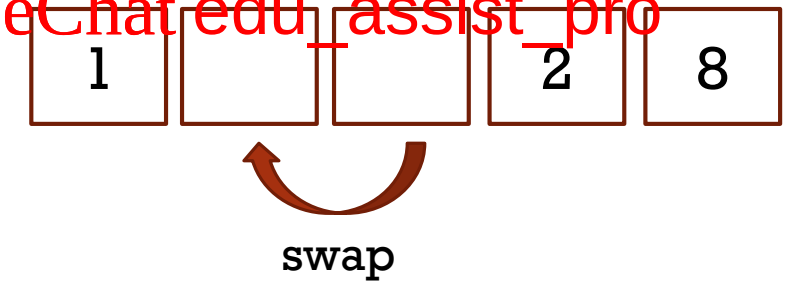
- Compare all adjacent elements.
- If needed, swap!



EXAMPLE – ONE ITERATION

Iteration #1

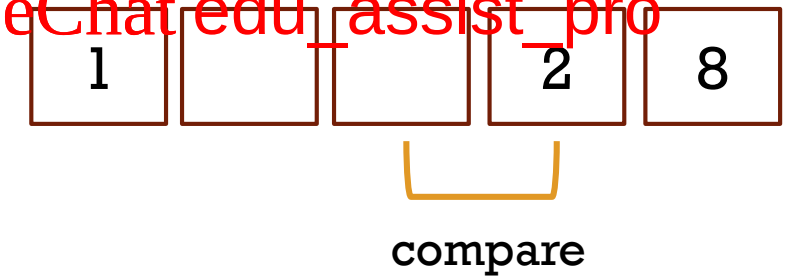
- Compare all adjacent elements.
- If needed, swap!



EXAMPLE – ONE ITERATION

Iteration #1

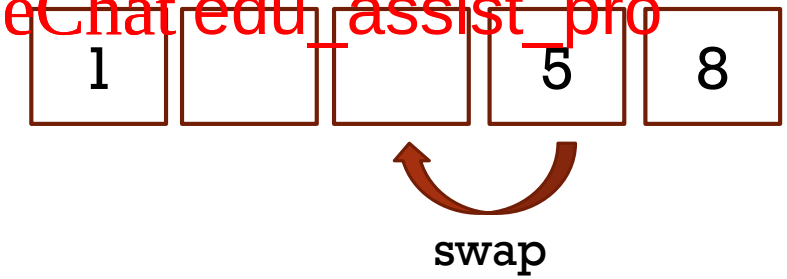
- Compare all adjacent elements.
- If needed, swap!



EXAMPLE – ONE ITERATION

Iteration #1

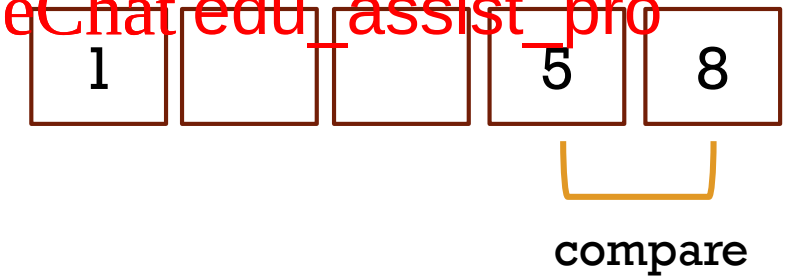
- Compare all adjacent elements.
- If needed, swap!



EXAMPLE – ONE ITERATION

Iteration #1

- Compare all adjacent elements.
- If needed, swap!



WHAT CAN WE SAY AFTER THE FIRST ITERATION?

Q: Where is the largest element ?

A:

Q: Where is the smallest element?

A:

WHAT CAN WE SAY AFTER THE FIRST ITERATION?

Q: Where is the largest element ?

A: It must be at the en

Q: Where is the smallest element?

A: Anywhere (except position N-1)

WHAT CAN WE SAY AFTER THE FIRST ITERATION?

Q: Where is the largest element ?

A: It must be at the en

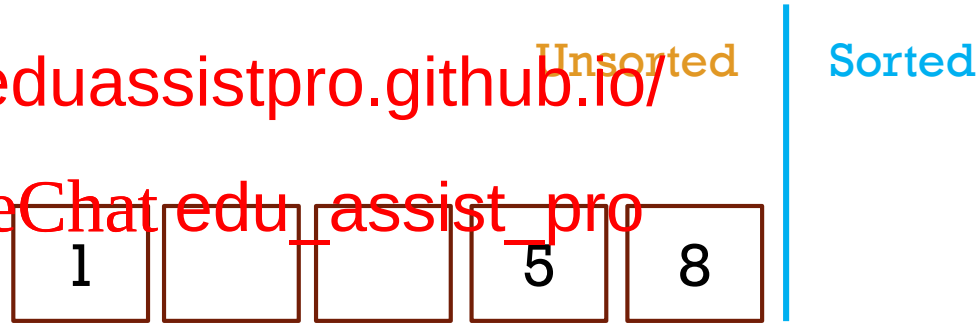
- Since each time we iterate through the list we ensure that the largest element is in the correct position. → at each iteration we can stop comparing adjacent elements one step earlier.

BUBBLE SORT – PSEUDOCODE

```
for i from 0 to list.length-1 {
   for j fr                    h - i -2 {
      if(li                          {
         swap(list[j],                 )
      }
   }
}
```

EXAMPLE

We left off at the end of Iteration #1



EXAMPLE

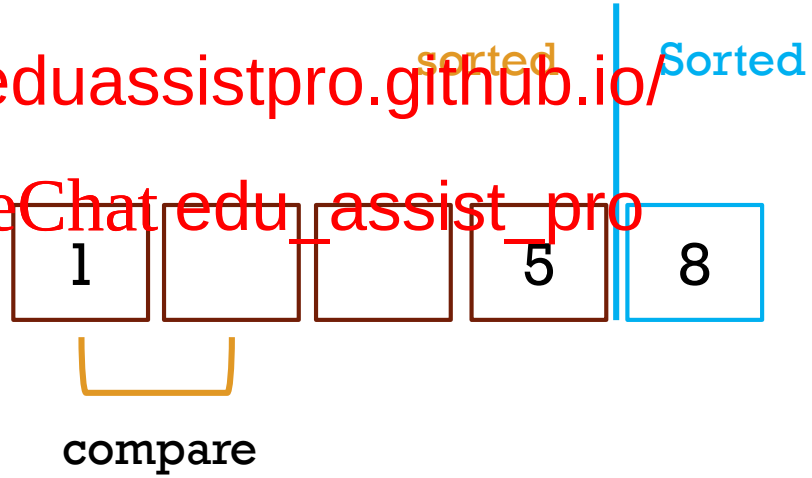
Iteration #2

- Compare all adjacent elements up to index **3**.
- If needed, swap!

EXAMPLE

Iteration #2

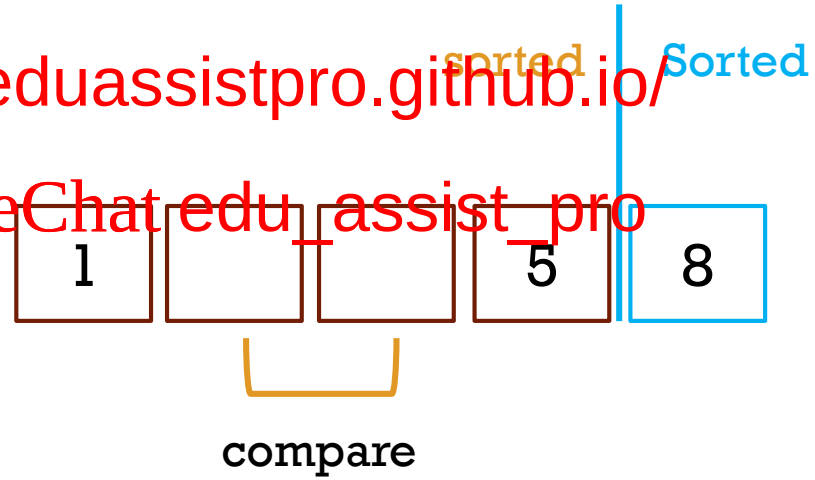
- Compare all adjacent elements up to index **3**.
- If needed, swap!



EXAMPLE

Iteration #2

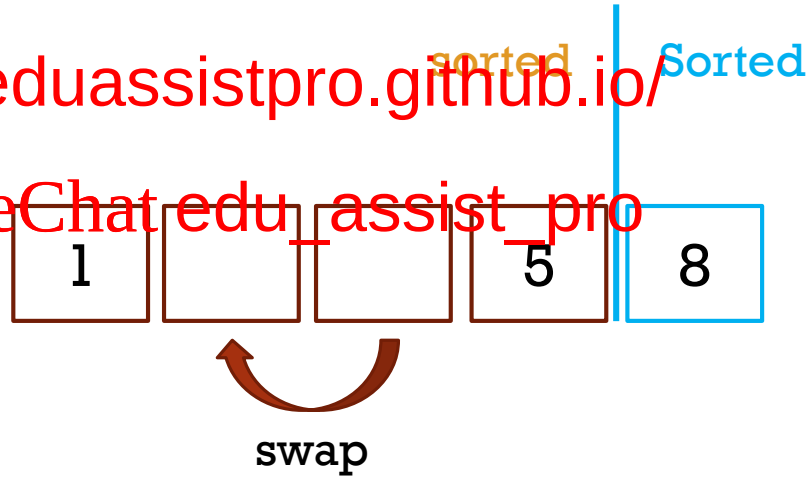
- Compare all adjacent elements up to index **3**.
- If needed, swap!



EXAMPLE

Iteration #2

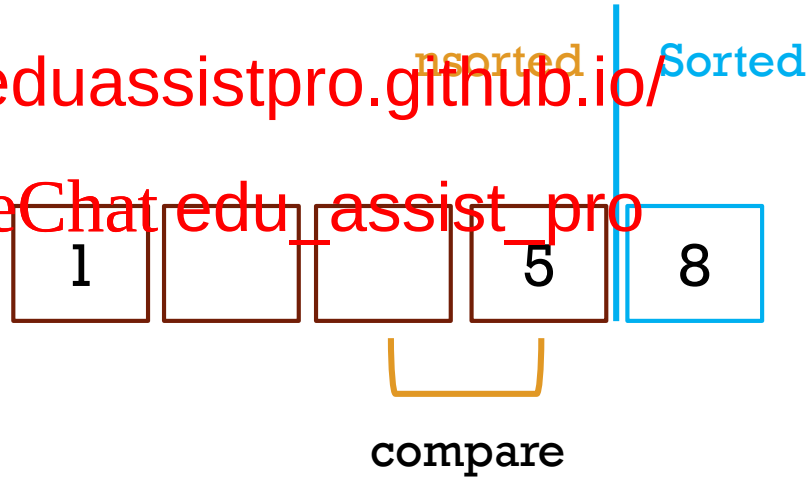
- Compare all adjacent elements up to index **3**.
- If needed, swap!



EXAMPLE

Iteration #2

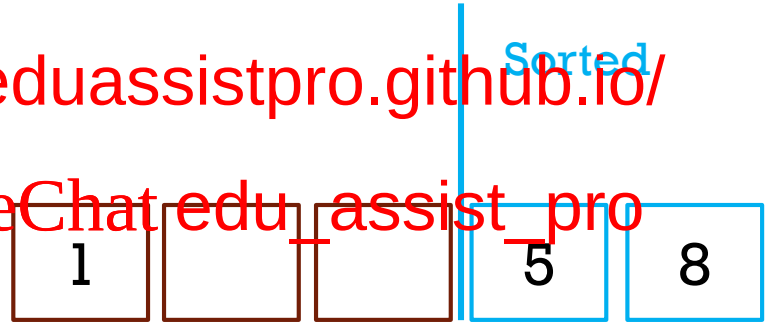
- Compare all adjacent elements up to index **3**.
- If needed, swap!



EXAMPLE

Iteration #3

- Compare all adjacent elements up to index **2**.



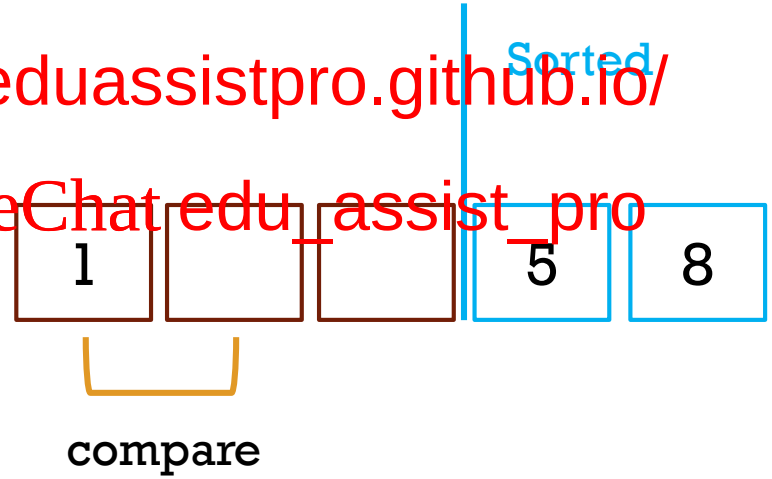
- If needed, swap!

Note: now the list is sorted, but the algorithm does not know that.
When can the algorithm infer that the list is sorted?

EXAMPLE

Iteration #3

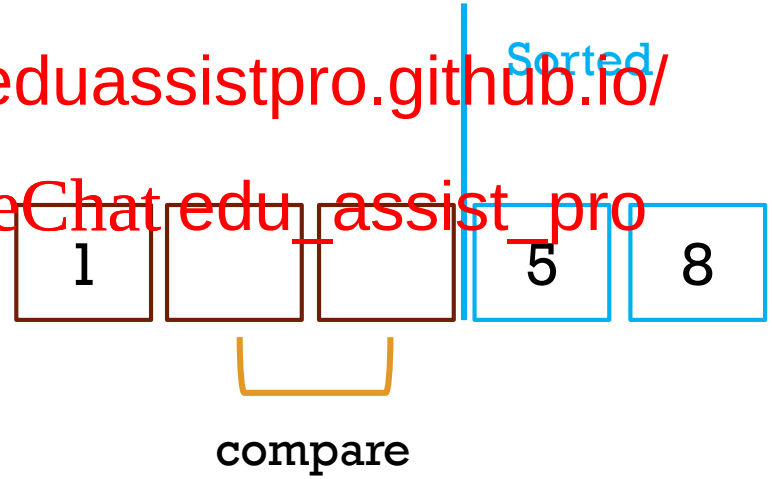
- Compare all adjacent elements up to index **2**.
- If needed, swap!



EXAMPLE

Iteration #3

- Compare all adjacent elements up to index **2**.
- If needed, swap!



No swap was needed in this iteration → the list is sorted!

EXAMPLE

No swap was needed in the last iteration. We can stop comparing. The list is sorted!

BUBBLE SORT – PSEUDOCODE

```
sorted = false
i = 0
while (!sorted) {
    sorted =
    for j fr                    - i - 2 {
        if(list[j] > list
            swap(list[j], list[j+1])
            sorted = false
        }
    }
    i++
}
```

S RT

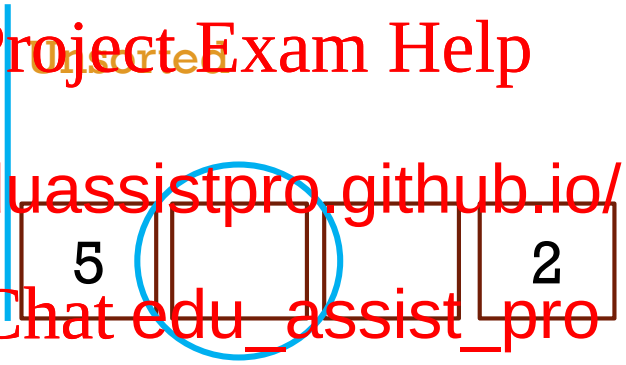
SELECTION SORT

- Goal: order a list of integers in ascending order
- Idea: consider the list as if it was divided into two parts, one sorted and the other unsort g the sorted part is empty)
- Procedure:
  - Select the smallest element in the unsorted part of the list
  - Swap that element with the element in the initial position of the unsorted array
  - Change where you divide the array from the sorted part to the unsorted part.

EXAMPLE

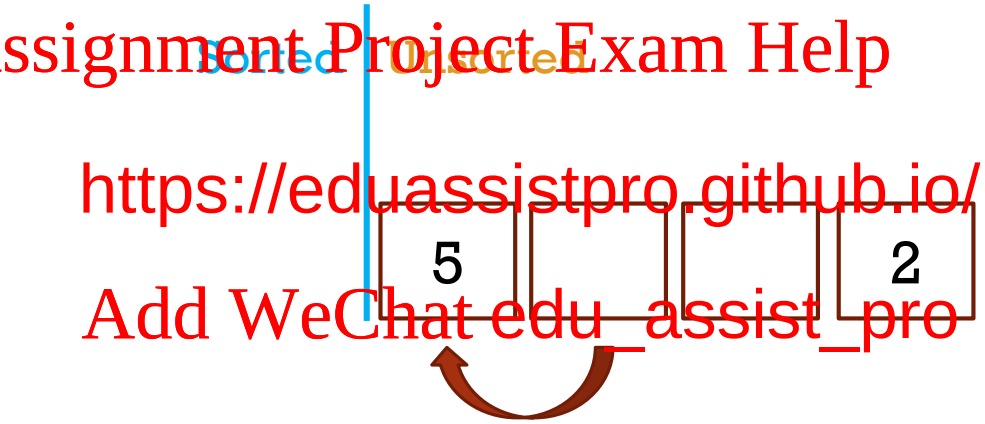
EXAMPLE

- Select



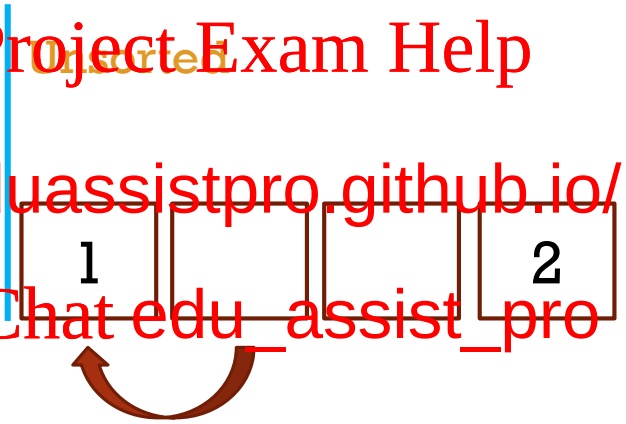
EXAMPLE

- Select
- Swap



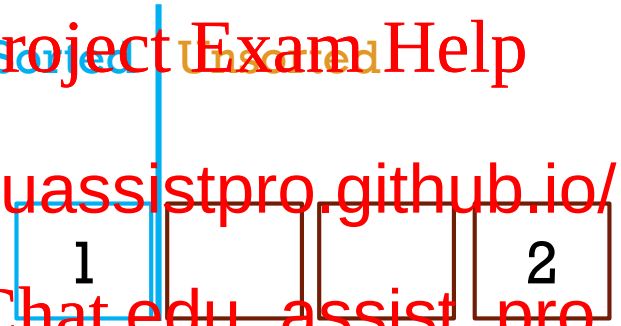
EXAMPLE

- Select
- Swap



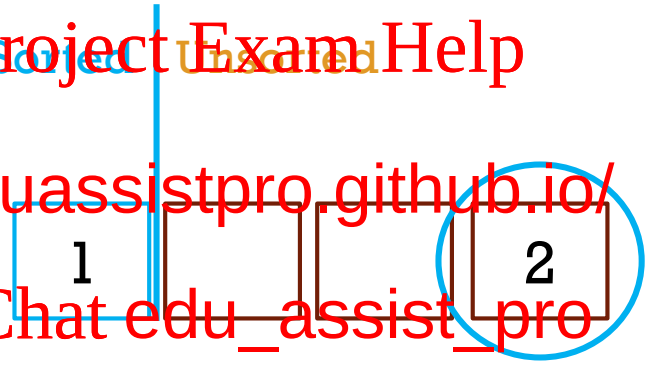
EXAMPLE

- Select
- Swap
- Update delimiter



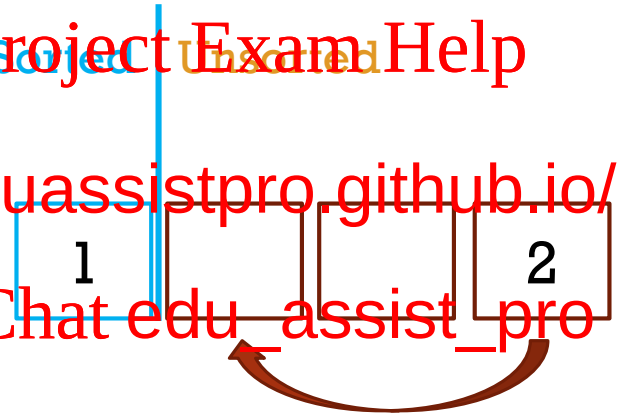
EXAMPLE

- Select



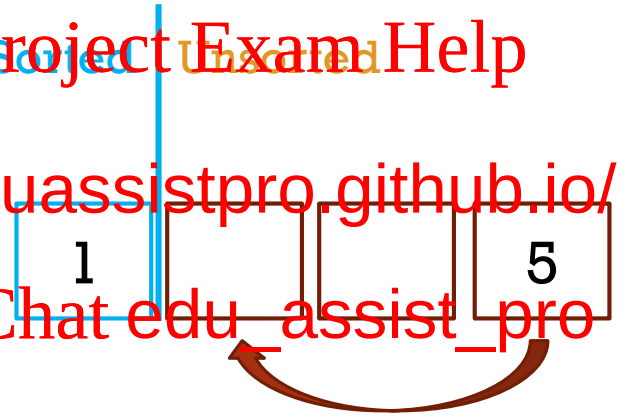
EXAMPLE

- Select
- Swap



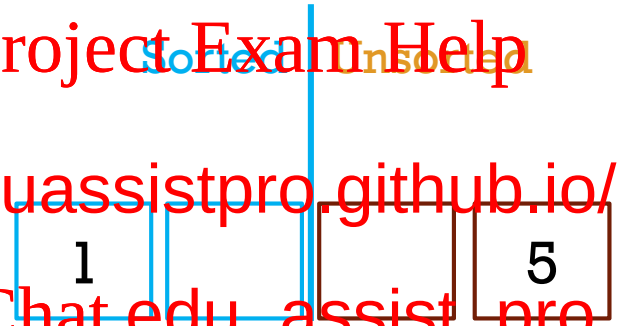
EXAMPLE

- Select
- Swap



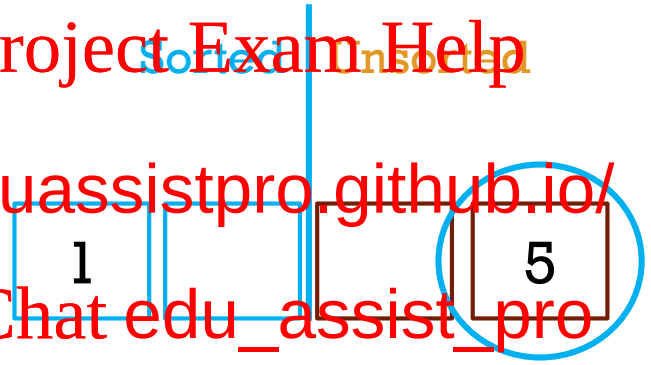
EXAMPLE

- Select
- Swap
- Update



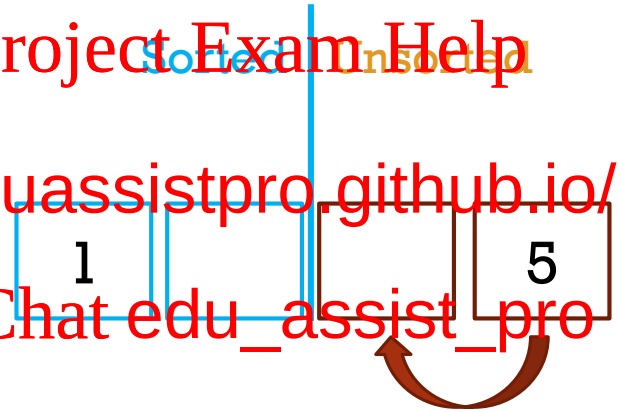
EXAMPLE

- Select



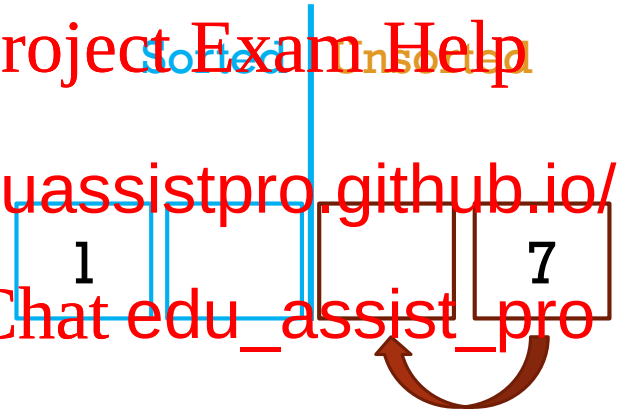
EXAMPLE

- Select
- Swap



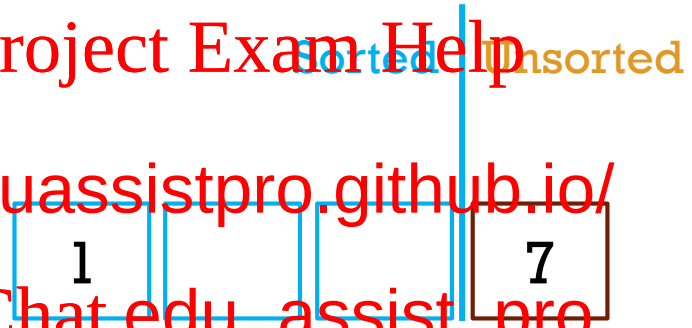
EXAMPLE

- Select
- Swap



EXAMPLE

- Select
- Swap
- Update



EXAMPLE

- Done!

SELECTION SORT – PSEUDOCODE

```
for delim from 0 to N-2 {
   min = delim
   for i from delim+1 to N-1 {
      if(list[i]
         min = i
      }
   }
   if(min != delim) {
      swap(list[min], list[delim])
   }
}
```

Repeat until list is all sorted (~N times)

Find the index of the min element in the unsorted part of the list

Swap the min element in the first position of the unsorted part of the list.

SELECTION SORT

```
for delim from 0 to N-2
    for i from delim+1 to N-1
        ...
```


- How many times does the inner lo

SELECTION SORT

```
for delim from 0 to N-2
    for i from delim+1 to N-1
        ...
```


- How many times does the inner lo
  - N-1 + N-2 + N-3 +…+ 2 + 1

SELECTION SORT

```
for delim from 0 to N-2
    for i from delim+1 to N-1
        ...
```


- How many times does the inner lo

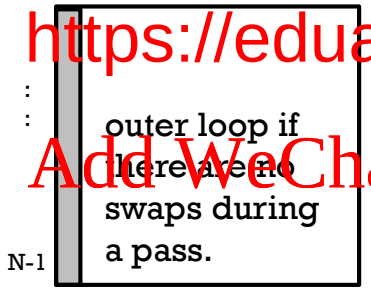
  - N-1 + N-2 + N-3 +…+ 2 + 1 = **N*(N-1)/2**

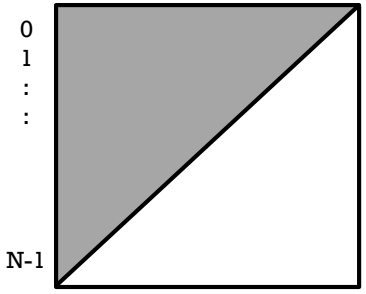
COMPARISON

Dark area denotes which elements of the list need to be examined at each iteration of the outer loop.

Bubblesort

```
while(!sorted)
    for i from 0 to N – 2 – i
```

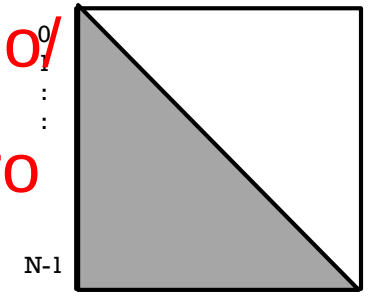





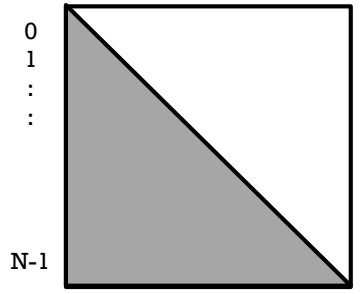
Outer loop

Selection sort

```
for  delim from 0  to N-2
    for i from delim+1  to  N-1
```



Worst case



Outer loop

I RT

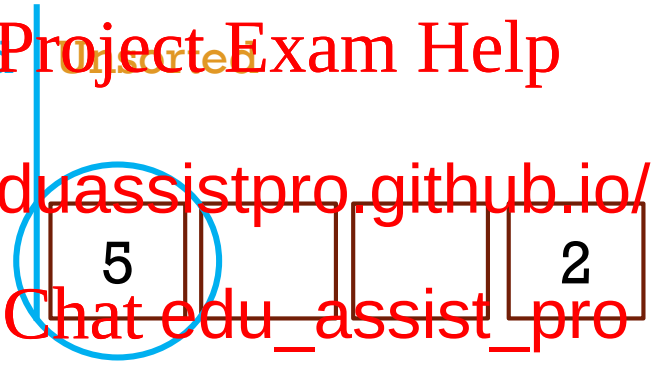
INSERTION SORT

- Goal: order a list of integers in ascending order
- Idea: consider the list as if it was divided into two parts, one sorted and the other unsort g the sorted part is empty)
- Procedure:
  - Select the first element of the unsorted part of the list
  - Insert such element into its correct position in the sorted part of the list.
  - Change where you divide the array from the sorted part to the unsorted part.

EXAMPLE

EXAMPLE

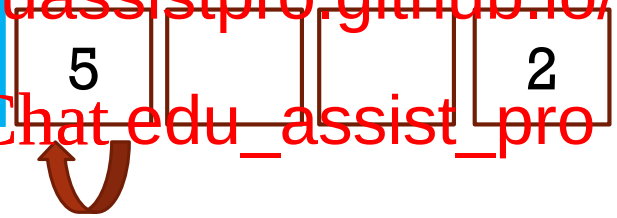
- Select



EXAMPLE

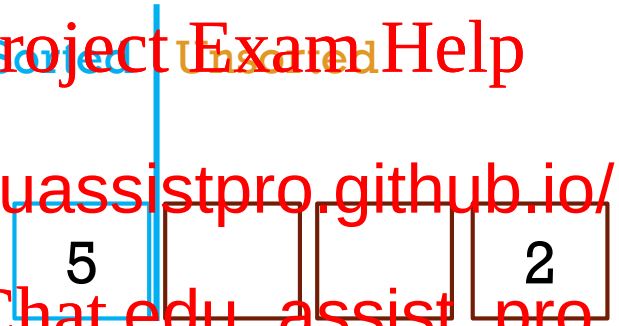
- Select
- Insert

sorted | unsorted



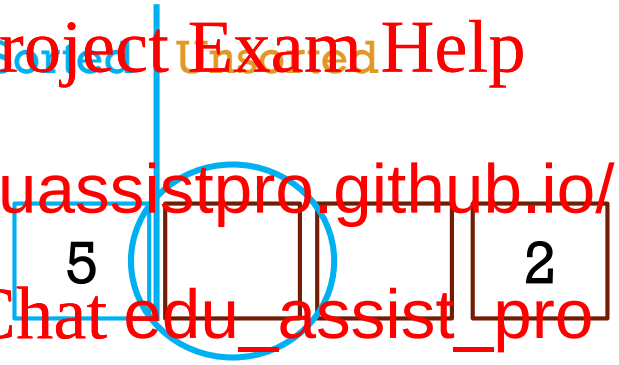
EXAMPLE

- Select
- Insert
- Update



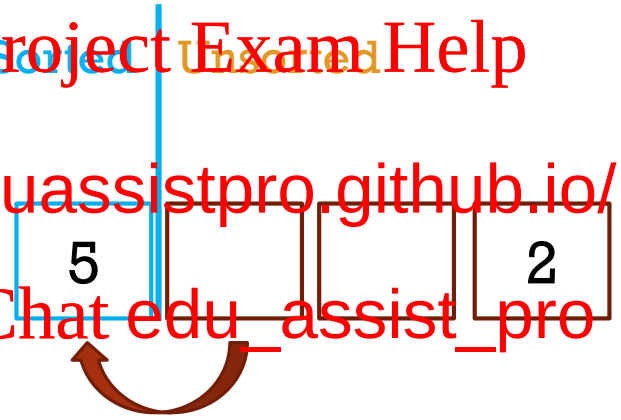
EXAMPLE

- Select



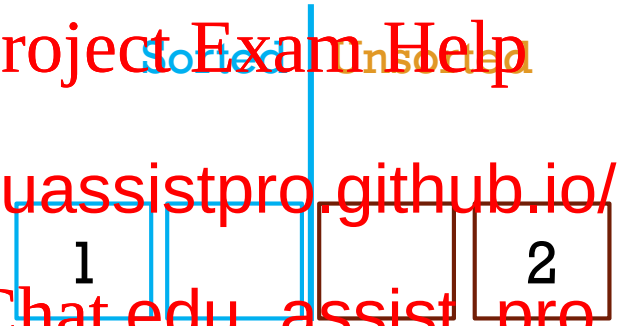
EXAMPLE

- Select
- Insert



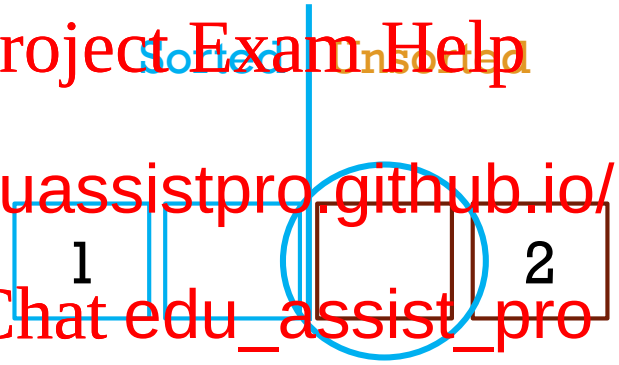
EXAMPLE

- Select
- Insert
- Update



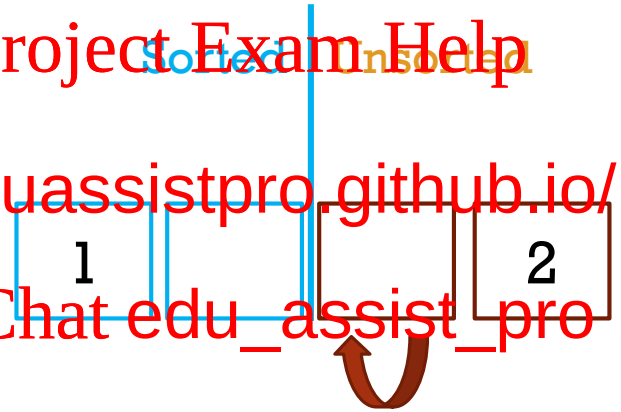
EXAMPLE

- Select



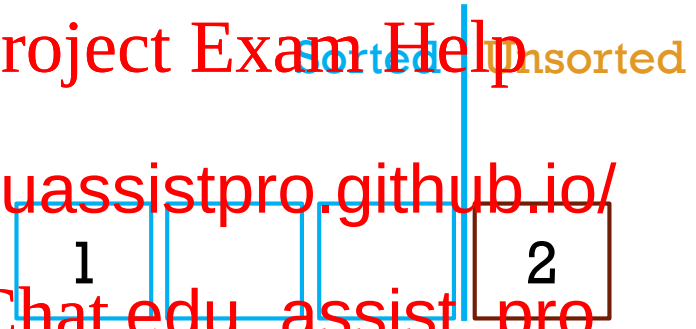
EXAMPLE

- Select
- Insert



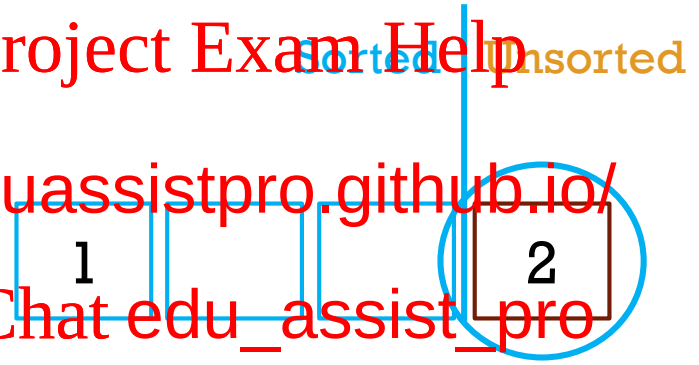
EXAMPLE

- Select
- Insert
- Update



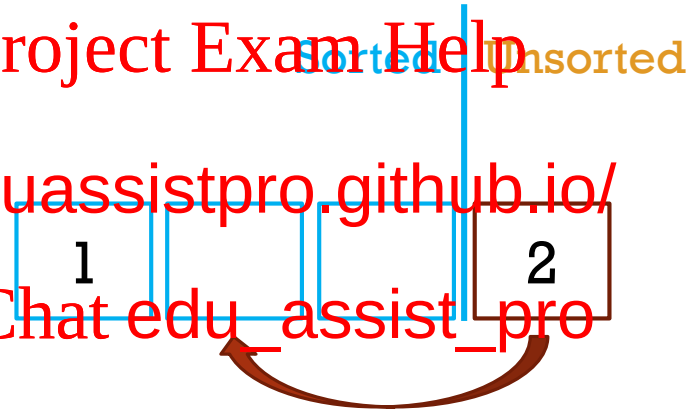
EXAMPLE

- Select



EXAMPLE

- Select
- Insert



EXAMPLE

- Done!

INSERTING

Mechanism is similar to inserting (adding) an element to an array list:

Shift all elements ahead by one make a hole, and then fill the hole.

INSERTION SORT – PSEUDOCODE

```
for i from 0 to N-1 {
    element = list[i]
    k = i
    while(k>0 && e
        list[k] = list[k-1]
        k--
    }
    list[k] = element
}
```

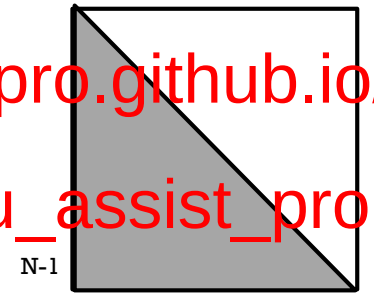
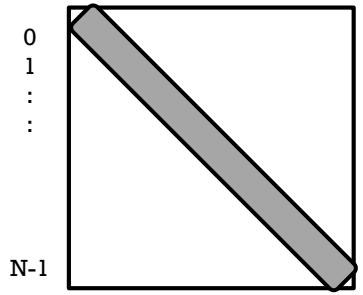
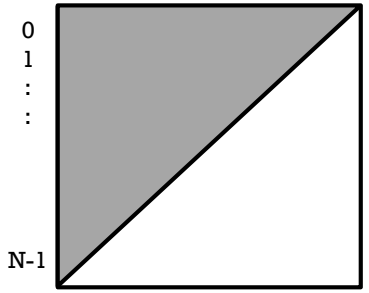
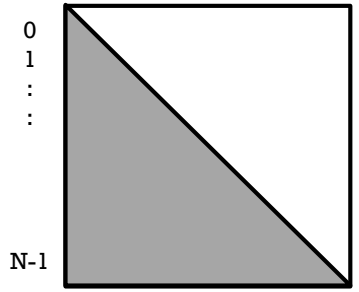
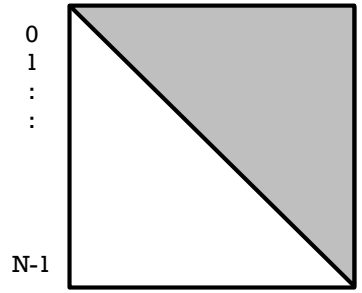

Repeat until list is all sorted (~N times)

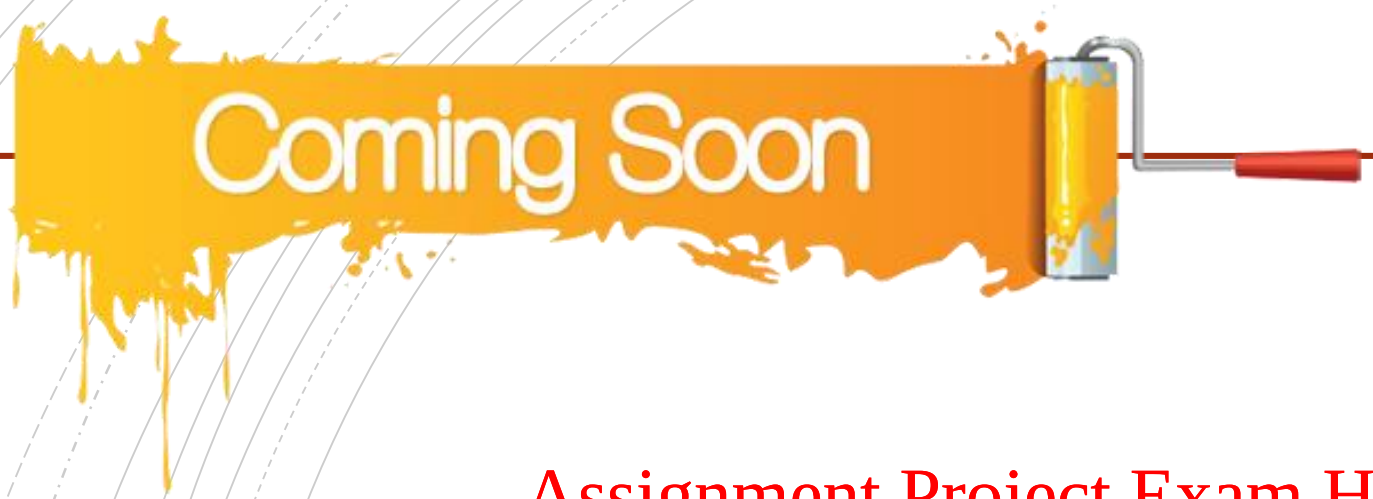
Find where the element should be inserted in the sorted part of the list + make space for it (shift all the larger elements to the right)

Insert the element in the sorted part of the list.

COMPARISON OF THE THREE ALGORITHMS

Performance depends highly on initial data. Also, it depends on implementation (array vs. linked list), e.g. what is cost of swap and 'shift'.

	Bubblesort	Selection sort	Insertion sort
	while(!sorted) for j from 0 to N – 2 – i	for delim from 0 to N-2 for i from delim+1 to N-1	for i from 0 to N-1 while ….
Best case			
Worst case			
	Outer loop	Outer loop	Outer loop



In the next

- Asympto