Data Structures and Algorithms

Sorting

CS51 – Spring 2026

So far in this unit we talked about how to store data in different structures, such as sequences and dictionaries. Quite often, we will want the data to follow a certain ordering. This can be imposed by sorting the data.

Sorting

- Process of arranging elements in a data structure in non-decreasing order.
- At the heart of many problems, often good first step.
- Exemplary algorithms that we study in CS.

By sorting, I mean the process of arranging, that is ordering, elements in a data structure in non-decreasing order. We say non-decreasing to take care of duplicates. Running the right sorting algorithm on our data is often the first step in solving problems that at first appear a lot more complicated and because of that they are recurring subject of study both theoretically and more practically in our curriculum.

Assumptions for sorting data

- We are working with lists of elements of the same type.
- We have a way of comparing elements with each other.
- We can exchange elements.
- Many different sorting algorithms and most use these assumptions.

3

Let's establish some rules for sorting. For simplicity, we are assuming that we are working with lists only. We will assume that the list we want to sort contains elements of the same type and that there is a way of comparing elements with each other. Since lists are mutable, we know that we will be able to exchange elements. There are a lot of different sorting algorithms and most use these assumptions.

Ingredient 1: Compare two elements

- We want to be able to determine whether two elements are equal or which one is "smaller".
- Smaller can mean different things based on the type of data.
  - For example, numbers are sorted in non-decreasing order and strings are sorted in lexicographic order.
- We will assume we work with elements we can compare using the <, >, and == operators.

4

Let's start with how we compare two elements. We want to be able to determine whether two elements are equal or which one is smaller so that we can put it first. Smaller can of course mean different things based on what data type we are working with. E.g., for ints we talk about non-decreasing order, for strings about lexicographic order. Regardless of the data type that our list holds, we will assume we can compare two elements with the <, >, and == operators.

Ingredient 2: Exchange two elements

- We want to be able to exchange two elements in the list.
- Define a function `exchange` that takes three parameters, a list and two indices which it uses to exchange the elements in these positions.
- For example, if `lst = [-12, 3, 7, -1]` calling `exchange(lst, 0, 3)` would result in `lst = [-1, 3, 7, -12]`
-
```
def exchange(lst, i, j):
    temp = lst[i]
    lst[i] = lst[j]
    lst[j] = temp
```

5

the second thing we will need is a way to exchange two elements in the list as our ultimate goal is to bring them to the right position. It's your turn now to define a function called exchange that takes three inputs, a list, and two indices for which it uses to exchange the elements in these positions. We will do so with using a temporary local variable to store the element at one of the indices.

Bubble sort

6

The first algorithm that we will see is called bubblesort. it's

Bubble sort

```
def bubblesort (lst):
    n = len(lst)
    for i in range(n):
        for j in range(n-i-1):
            if lst[j]>lst[j+1]:
                exchange(lst, j, j+1)
```

- Repeatedly exchange adjacent elements that are out of order.
- At the end of first pass of the external loop, the maximum element goes to the last index.
- At the end of the second pass, the second maximum element goes to the second to last index, and so on.

7

Bubble sort is a very simple algorithm: it repeatedly exchanges adjacent elements that are out of order. At the end of the first pass, the maximum element of the list will be at the last index. At the end of the second pass, the second maximum element will be at the penultimate index, and so on

Bubble sort example – 1st pass

```
def bubblesort (lst):
    n = len(lst)
for i in range(n):
    for j in range(n-i-1):
        if lst[j]>lst[j+1]:
            exchange(lst, j, j+1)
```

0	1	2	3	4
3	1	5	6	2

- i=0
- j=0
- Is lst[0]>lst[1], that is 3>1?
- Yes, then exchange them

0	1	2	3	4
1	**3**	5	6	2

8

Bubble sort example – 1st pass

```
def bubblesort (lst):
    n = len(lst)
for i in range(n):
    for j in range(n-i-1):
        if lst[j]>lst[j+1]:
            exchange(lst, j, j+1)
```

0	1	2	3	4
1	3	5	6	2

- i=0
- j=1
- Is lst[1]>lst[2], that is 3>5?
- No, don't do anything

9

Bubble sort example – 1st pass

```
def bubblesort (lst):
    n = len(lst)
for i in range(n):
    for j in range(n-i-1):
        if lst[j]>lst[j+1]:
            exchange(lst, j, j+1)
```

0	1	2	3	4
1	3	5	6	2

- i=0
- j=2
- Is lst[2]>lst[3], that is 5>6?
- No, don't do anything

10

Bubble sort example – 1st pass

```
def bubblesort (lst):
    n = len(lst)
for i in range(n):
    for j in range(n-i-1):
        if lst[j]>lst[j+1]:
            exchange(lst, j, j+1)
```

0	1	2	3	4
1	3	5	6	2

- i=0
- j=3
- Is lst[3]>lst[4], that is 6>2?
- Yes, then exchange them

0	1	2	3	4
1	3	5	**2**	**6**

11

Bubble sort example – 1st pass

```
def bubblesort (lst):
    n = len(lst)
for i in range(n):
    for j in range(n-i-1):
        if lst[j]>lst[j+1]:
            exchange(lst, j, j+1)
```

0	1	2	3	4
1	3	5	2	6

- End of first pass and the largest element, 6, is at the last index

12

Bubble sort example – 2nd pass

```
def bubblesort (lst):
    n = len(lst)
    for i in range(n):
        for j in range(n-i-1):
            if lst[j]>lst[j+1]:
                exchange(lst, j, j+1)
```

0	1	2	3	4
1	3	5	2	6

- i=1
- j=0
- Is lst[0]>lst[1], that is 1>3?
- No, don't do anything

13

Bubble sort example – 2nd pass

```
def bubblesort (lst):
    n = len(lst)
    for i in range(n):
        for j in range(n-i-1):
            if lst[j]>lst[j+1]:
                exchange(lst, j, j+1)
```

0	1	2	3	4
1	3	5	2	6

- i=1
- j=1
- Is lst[1]>lst[2], that is 3>5?
- No, don't do anything

14

Bubble sort example – 2nd pass

```
def bubblesort (lst):
    n = len(lst)
    for i in range(n):
        for j in range(n-i-1):
            if lst[j]>lst[j+1]:
                exchange(lst, j, j+1)
```

0	1	2	3	4
1	3	5	2	6

- i=1
- j=2
- Is lst[2]>lst[3], that is 5>2?
- Yes, then exchange them

0	1	2	3	4
1	3	**2**	**5**	6

15

Bubble sort example – 2nd pass

```
def bubblesort (lst):
    n = len(lst)
    for i in range(n):
        for j in range(n-i-1):
            if lst[j]>lst[j+1]:
                exchange(lst, j, j+1)
```

0	1	2	3	4
1	3	2	5	6

- End of second pass and the second largest element, 5, is at the second to last index

16

Bubble sort example – 3rd pass

```
def bubblesort (lst):
    n = len(lst)
    for i in range(n):
        for j in range(n-i-1):
            if lst[j]>lst[j+1]:
                exchange(lst, j, j+1)
```

0	1	2	3	4
1	3	2	5	6

- i=2
- j=0
- Is lst[0]>lst[1], that is 1>3?
- No don't do anything

17

Bubble sort example – 3rd pass

```
def bubblesort (lst):
    n = len(lst)
    for i in range(n):
        for j in range(n-i-1):
            if lst[j]>lst[j+1]:
                exchange(lst, j, j+1)
```

0	1	2	3	4
1	3	2	5	6

- i=2
- j=1
- Is lst[1]>lst[2], that is 3>2?
- Yes, then exchange them

0	1	2	3	4
1	**2**	**3**	5	6

18

Bubble sort example – 3rd pass

```
def bubblesort (lst):
    n = len(lst)
    for i in range(n):
        for j in range(n-i-1):
            if lst[j]>lst[j+1]:
                exchange(lst, j, j+1)
```

0	1	2	3	4
1	**2**	**3**	5	6

- End of third pass and the third largest element, 3, is at the third to last index

19

Bubble sort example – 4th pass

```
def bubblesort (lst):
    n = len(lst)
    for i in range(n):
        for j in range(n-i-1):
            if lst[j]>lst[j+1]:
                exchange(lst, j, j+1)
```

0	1	2	3	4
1	2	3	5	6

- i=3
- j=0
- Is lst[0]>lst[1], that is 1>2?
- No, don't do anything

20

Bubble sort example – 4th pass

```
def bubblesort (lst):
    n = len(lst)
    for i in range(n):
        for j in range(n-i-1):
            if lst[j]>lst[j+1]:
                exchange(lst, j, j+1)
```

0	1	2	3	4
1	2	3	5	6

- End of fourth pass and the fourth largest element, 2, is at the fourth to last index

21

Bubble sort loop invariant

```
def bubblesort (lst):
    n = len(lst)
    for i in range(n):
        for j in range(n-i-1):
            if lst[j]>lst[j+1]:
                exchange(lst, j, j+1)
```

- **Practice time:** What would be good loop invariant for the outer loop?

22

Consider again the code for bubblesort. If we were to think of the outer loop, what would a good loop invariant for it be?

Bubble sort loop invariant

```
def bubblesort (lst):
    n = len(lst)
    for i in range(n):
        for j in range(n-i-1):
            if lst[j]>lst[j+1]:
                exchange(lst, j, j+1)
```

- **Outer loop invariant:** At the start of the i-th iteration, the largest i elements in the list are in the last i indices in non-descending order (i.e. in indices lst[n-i:])

23

For the outer loop, we have to consider that after the first iteration the largest element is in the last index and after the second iteration the second largest element is in the second index from the end and so on. So a good loop invariant would be that at the start of the i-th iteration, the largest i elements in the list are in the last i indices in sorted order. If we were to prove this, we would actually need the idea of the inner loop invariant which we will see next.

Bubble sort - optimization

```
def bubblesort (lst):
    n = len(lst)
    for i in range(n):
        exchanged = False
        for j in range(n-i-1):
            if lst[j]>lst[j+1]:
                exchange(lst, j, j+1)
                exchanged = True
        if not exchanged:
            break
```

25

Finally, you might have noticed that we did some unnecessary work at the fourth pass and we could have stopped earlier since every element was at its final place. A small optimization we could do is to add a bool variable that allows us to check if a pass did not change anything. If that's the case, we can break the for loop and consider our work complete.

Practice Time

- Following the same steps we just did, run Bubble sort on the following list [37, 6, 33, 25, 36, 27, 14, 7, 40, 23].

```
def bubblesort (lst):
    n = len(lst)
    for i in range(n):
        for j in range(n-i-1):
            if lst[j]>lst[j+1]:
                exchange(lst, j, j+1)
```

26

Selection sort

27

The second algorithm that we will see is called selection sort.

Selection sort

```
def selectionsort (lst):
    n = len(lst)
    for i in range(n-1):
        min_index = i
        for j in range(i+1, n):
            if lst[j]<lst[min_index]:
                min_index = j
        exchange(lst, i, min_index)
```

- Minimizes number of exchanges.
- At every pass, identifies the minimum element and moves it to the left.

28

In contrast to bubblesort, selection sort exchanges data far less frequently. It contains again two for loops. The outer for loop identifies the minimum element and moves it to the left. That means that after the first pass, the minimum element will be in the first index, after the second pass, the second smallest element will be in the second index and so on.

Selection sort example – 1st pass

```
def selectionsort (lst):
    n = len(lst)
    for i in range(n-1):
        min_index = i
        for j in range(i+1, n):
            if lst[j]<lst[min_index]:
                min_index = j
        exchange(lst, i, min_index)
```

0	1	2	3	4
3	1	5	6	2

- i=0
- j=1
- min_index = 0
- Is lst[1]<lst[0], that is 1<3?
- Yes, min_index = 1

0	1	2	3	4
3	**1**	5	6	2

29

Here's an example to see how selection sort works.

Selection sort example – 1st pass

```
def selectionsort (lst):
    n = len(lst)
    for i in range(n-1):
        min_index = i
        for j in range(i+1, n):
            if lst[j]<lst[min_index]:
                min_index = j
        exchange(lst, i, min_index)
```

0	1	2	3	4
3	**1**	5	6	2

- i=0
- j=2
- min_index = 1
- Is lst[2]<lst[1], that is 5>1?
- No, don't do anything

30

Selection sort example – 1st pass

```
def selectionsort (lst):
    n = len(lst)
    for i in range(n-1):
        min_index = i
        for j in range(i+1, n):
            if lst[j]<lst[min_index]:
                min_index = j
        exchange(lst, i, min_index)
```

0	1	2	3	4
3	**1**	5	6	2

- i=0
- j=3
- min_index = 1
- Is lst[3]<lst[1], that is 6<1?
- No, don't do anything

31

Selection sort example – 1st pass

```
def selectionsort (lst):
    n = len(lst)
    for i in range(n-1):
        min_index = i
        for j in range(i+1, n):
            if lst[j]<lst[min_index]:
                min_index = j
        exchange(lst, i, min_index)
```

0	1	2	3	4
3	**1**	5	6	2

- i=0
- j=4
- min_index = 1
- Is lst[4]<lst[1], that is 2<1?
- No, don't do anything

32

Selection sort example – 1st pass

```
def selectionsort (lst):
    n = len(lst)
    for i in range(n-1):
        min_index = i
        for j in range(i+1, n):
            if lst[j]<lst[min_index]:
                min_index = j
        exchange(lst, i, min_index)
```

0	1	2	3	4
1	3	5	6	2

- Exchange element at index i=0 with element at min_index=1
- End of first pass and the smallest element, 1, is at the first index

33

Selection sort example – 2nd pass

```
def selectionsort (lst):
    n = len(lst)
    for i in range(n-1):
        min_index = i
        for j in range(i+1, n):
            if lst[j]<lst[min_index]:
                min_index = j
        exchange(lst, i, min_index)
```

0	1	2	3	4
1	**3**	5	6	2

- i=1
- j=2
- min_index = 1
- Is lst[2]<lst[1], that is 5<3?
- No, don't do anything

34

Selection sort example – 2nd pass

```
def selectionsort (lst):
    n = len(lst)
    for i in range(n-1):
        min_index = i
        for j in range(i+1, n):
            if lst[j]<lst[min_index]:
                min_index = j
        exchange(lst, i, min_index)
```

0	1	2	3	4
1	**3**	5	6	2

- i=1
- j=3
- min_index = 1
- Is lst[3]<lst[1], that is 6<3?
- No, don't do anything

35

Selection sort example – 2nd pass

```
def selectionsort (lst):
    n = len(lst)
    for i in range(n-1):
        min_index = i
        for j in range(i+1, n):
            if lst[j]<lst[min_index]:
                min_index = j
        exchange(lst, i, min_index)
```

0	1	2	3	4
1	**3**	5	6	2

- i=1
- j=4
- min_index = 1
- Is lst[4]<lst[1], that is 2<3?
- Yes, min_index = 4

0	1	2	3	4
1	3	5	6	**2**

36

Selection sort example – 2nd pass

```
def selectionsort (lst):
    n = len(lst)
    for i in range(n-1):
        min_index = i
        for j in range(i+1, n):
            if lst[j]<lst[min_index]:
                min_index = j
        exchange(lst, i, min_index)
```

0	1	2	3	4
1	2	5	6	3

- Exchange element at index i=1 with element at min_index=4
- End of second pass and the second smallest element, 2, is at the second index

37

Selection sort example – 3rd pass

```
def selectionsort (lst):
    n = len(lst)
    for i in range(n-1):
        min_index = i
        for j in range(i+1, n):
            if lst[j]<lst[min_index]:
                min_index = j
        exchange(lst, i, min_index)
```

0	1	2	3	4
1	2	**5**	6	3

- i=2
- j=3
- min_index = 2
- Is lst[3]<lst[2], that is 6<5?
- No, don't do anything

38

Selection sort example – 3rd pass

```
def selectionsort (lst):
    n = len(lst)
    for i in range(n-1):
        min_index = i
        for j in range(i+1, n):
            if lst[j]<lst[min_index]:
                min_index = j
        exchange(lst, i, min_index)
```

0	1	2	3	4
1	2	**5**	6	3

- i=2
- j=4
- min_index = 2
- Is lst[4]<lst[2], that is 3<5?
- Yes, min_index = 4

0	1	2	3	4
1	2	5	**6**	**3**

39

Selection sort example – 3rd pass

```
def selectionsort (lst):
    n = len(lst)
    for i in range(n-1):
        min_index = i
        for j in range(i+1, n):
            if lst[j]<lst[min_index]:
                min_index = j
        exchange(lst, i, min_index)
```

0	1	2	3	4
1	2	3	6	**5**

- Exchange element at index i=2 with element at min_index=4
- End of third pass and the third smallest element, 3, is at the third index

40

Selection sort example – 4th pass

```
def selectionsort (lst):
    n = len(lst)
    for i in range(n-1):
        min_index = i
        for j in range(i+1, n):
            if lst[j]<lst[min_index]:
                min_index = j
        exchange(lst, i, min_index)
```

0	1	2	3	4
1	2	3	**6**	5

- i=3
- j=4
- min_index = 3
- Is lst[4]<lst[3], that is 5<6?
- Yes, min_index = 4

0	1	2	3	4
1	2	3	6	**5**

41

Selection sort example – 4th pass

```
def selectionsort (lst):
    n = len(lst)
    for i in range(n-1):
        min_index = i
        for j in range(i+1, n):
            if lst[j]<lst[min_index]:
                min_index = j
        exchange(lst, i, min_index)
```

0	1	2	3	4
1	2	3	5	6

- Exchange element at index i=3 with element at min_index=4
- End of third pass and the third smallest element, 3, is at the third index

42

Selection sort loop invariant

```
def selectionsort (lst):
    n = len(lst)
    for i in range(n-1):
        min_index = i
        for j in range(i+1, n):
            if lst[j]<lst[min_index]:
                min_index = j
        exchange(lst, i, min_index)
```

- **Practice time:** Loop invariants for outer loop?

43

Selection sort loop invariant

```
def selectionsort (lst):
    n = len(lst)
    for i in range(n-1):
        min_index = i
        for j in range(i+1, n):
            if lst[j]<lst[min_index]:
                min_index = j
        exchange(lst, i, min_index)
```

- **Outer loop:** At the start of the i-th loop, the smallest i elements are in non-descending order in lst[0:i]

45

Practice Time

- Following the same steps we just did, run selection sort on the following list [37, 6, 33, 25, 36, 27, 14, 7, 40, 23].

```
def selectionsort (lst):
    n = len(lst)
    for i in range(n-1):
        min_index = i
        for j in range(i+1, n):
            if lst[j]<lst[min_index]:
                min_index = j
        exchange(lst, i, min_index)
```

46

Insertion sort

47

The last sorting algorithm that we will see is called insertion sort.

Insertion sort

```
def insertionsort (lst):
    n = len(lst)
    for i in range(1, n):
        for j in range(i, 0, -1):
            if lst[j]<lst[j-1]:
                exchange(lst, j, j-1)
            else:
                break
```

- Intuitive algorithm that is often the first one that people think of.
- Splits the elements into two groups, a partially sorted on the left and an unsorted on the right. Then it takes an element from the unsorted group and finds its right place in the sorted group.

48

Insertion sort is probably the one algorithm that you are intuitively using if you hold in one hand a small number of cards and try to sort them. Insertion sort splits the elements into two groups, a partially sorted on the left and an unsorted on the right. The it repeatedly takes an element from the unsorted group and finds its right place in the partially sorted one.

Insertion sort example – 1st pass

```
def insertionsort (lst):
    n = len(lst)
    for i in range(1, n):
        for j in range(i, 0, -1):
            if lst[j]<lst[j-1]:
                exchange(lst, j, j-1)
            else:
                break
```

0	1	2	3	4
3	1	5	6	2

- i=1
- j=1
- Is lst[1]<lst[0], that is 1<3?
- Yes, exchange them

0	1	2	3	4
1	3	5	6	2

49

Here's an example to see how insertion sort works.

Insertion sort example – 2nd pass

```
def insertionsort (lst):
    n = len(lst)
    for i in range(1, n):
        for j in range(i, 0, -1):
            if lst[j]<lst[j-1]:
                exchange(lst, j, j-1)
            else:
                break
```

0	1	2	3	4
1	3	5	6	2

- i=2
- j=2
- Is lst[2]<lst[1], that is 5<3?
- No, break inner for loop

0	1	2	3	4
1	3	5	6	2

50

Insertion sort example – 3rd pass

```
def insertionsort (lst):
    n = len(lst)
    for i in range(1, n):
        for j in range(i, 0, -1):
            if lst[j]<lst[j-1]:
                exchange(lst, j, j-1)
            else:
                break
```

0	1	2	3	4
1	3	5	6	2

- i=3
- j=3
- Is lst[3]<lst[2], that is 6<5?
- No, break inner for loop

0	1	2	3	4
1	3	5	6	2

51

Insertion sort example – 4th pass

```
def insertionsort (lst):
    n = len(lst)
    for i in range(1, n):
        for j in range(i, 0, -1):
            if lst[j]<lst[j-1]:
                exchange(lst, j, j-1)
            else:
                break
```

0	1	2	3	4
1	3	5	6	2

- i=4
- j=4
- Is lst[4]<lst[3], that is 2<6?
- Yes, exchange them

0	1	2	3	4
1	3	5	2	6

52

Insertion sort example – 4th pass

```
def insertionsort (lst):
    n = len(lst)
    for i in range(1, n):
        for j in range(i, 0, -1):
            if lst[j]<lst[j-1]:
                exchange(lst, j, j-1)
            else:
                break
```

0	1	2	3	4
1	3	5	2	6

- i=4
- j=3
- Is lst[3]<lst[2], that is 2<5?
- Yes, exchange them

0	1	2	3	4
1	3	2	5	6

53

Insertion sort example – 4th pass

```
def insertionsort (lst):
    n = len(lst)
    for i in range(1, n):
        for j in range(i, 0, -1):
            if lst[j]<lst[j-1]:
                exchange(lst, j, j-1)
            else:
                break
```

0	1	2	3	4
1	3	2	5	6

- i=4
- j=2
- Is lst[2]<lst[1], that is 2<3?
- Yes, exchange them

0	1	2	3	4
1	2	3	5	6

54

Insertion sort example – 4th pass

```
def insertionsort (lst):
    n = len(lst)
    for i in range(1, n):
        for j in range(i, 0, -1):
            if lst[j]<lst[j-1]:
                exchange(lst, j, j-1)
            else:
                break
```

0	1	2	3	4
1	2	3	5	6

- i=4
- j=1
- Is lst[1]<lst[0], that is 2<3?
- No, break inner for loop

0	1	2	3	4
1	2	3	5	6

55

Insertion sort loop invariant

```
def insertionsort (lst):
    n = len(lst)
    for i in range(1, n):
        for j in range(i, 0, -1):
            if lst[j]<lst[j-1]:
                exchange(lst, j, j-1)
            else:
                break
```

- **Practice time:** Loop invariants for outer loop?

56

What loop invariants can we come up with?

Insertion sort loop invariant

```
def insertionsort (lst):
    n = len(lst)
    for i in range(1, n):
        for j in range(i, 0, -1):
            if lst[j]<lst[j-1]:
                exchange(lst, j, j-1)
            else:
                break
```

Outer loop: At the start of the i-th iteration the lst[0:i] slice consists of the elements originally in lst[0:i] but in sorted order (but not necessarily in final position)

Practice Time

- Following the same steps we just did, run insertion sort on the following list [37, 6, 33, 25, 36, 27, 14, 7, 40, 23].