

APS 105

Winter 2012

Jonathan Deber

jdeber -at- cs -dot- toronto -dot- edu

Lecture 33
April 9, 2012

Today

- More Sorting
- Course Evaluations

Sorting

- Two approaches:
 - Keep your list sorted as you add items
 - Take an unsorted list and sort it
- Depends on how often you're inserting items

Sorting an Existing List

- Simpler but inefficient algorithms
 - e.g., bubble sort, selection sort, insertion sort
- Complicated but faster algorithms
 - e.g., merge sort, quicksort
- In the real world, you rarely write these yourself

Bubble Sort

- Simplest of the “exchange sorts”

```
while (the list is unsorted)
```

```
{
```

```
    Go through the list, and compare pairs of items.
```

```
    if (they're out of order)
```

```
    {
```

```
        Swap them.
```

```
    }
```

```
}
```



23

49

10

4

28

88

64

37

↓ ↓

23 49 10 4 28 88 64 37

↓ ↓

23 10 49 4 28 88 64 37

↓ ↓

23 10 49 4 28 88 64 37

↓ ↓

23 10 4 49 28 88 64 37

23 10 4 49 28 88 64 37



23 10 4 28 49 88 64 37



23 10 4 28 49 88 64 37



23 10 4 28 49 88 64 37



23 10 4 28 49 64 88 37



23 10 4 28 49 64 88 37



23 10 4 28 49 64 37 88



23 10 4 28 49 64 37 88



23

10

4

28

49

64

37

88

↓ ↓

10 23 4 28 49 64 37 88

↓ ↓

10 23 4 28 49 64 37 88

↓ ↓

10 4 23 28 49 64 37 88

↓ ↓

10 4 23 28 49 64 37 88

↓ ↓

10 4 23 28 49 64 37 88

10 4 23 28 49 64 37 88



10 4 23 28 49 64 37 88



10 4 23 28 49 37 64 88



10 4 23 28 49 37 64 88



10 4 23 28 49 37 64 88

↓ ↓

10 4 23 28 49 37 64 88

↓ ↓

4 10 23 28 49 37 64 88

↓ ↓

4 10 23 28 49 37 64 88

↓ ↓

4 10 23 28 49 37 64 88

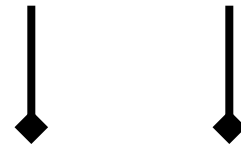
4 10 23 28 49 37 64 88



4 10 23 28 49 37 64 88



4 10 23 28 37 49 64 88



4 10 23 28 37 49 64 88



4 10 23 28 37 49 64 88



4 10 23 28 37 49 64 88

↓ ↓

4 10 23 28 37 49 64 88

↓ ↓

4 10 23 28 37 49 64 88

↓ ↓

4 10 23 28 37 49 64 88

↓ ↓

4 10 23 28 37 49 64 88

4 10 23 28 37 49 64 88



4 10 23 28 37 49 64 88



4 10 23 28 37 49 64 88



4 10 23 28 37 49 64 88

```
void bubbleSort(int list[], int n)
{
    bool isSorted = false;

    while (!isSorted)
    {
        isSorted = true;
        for (int i = 0; i < n - 1; i++)
        {
            if (list[i] > list[i + 1])
            {
                swap(&list[i], &list[i + 1]);
                isSorted = false;
            }
        }
    }
}
```



23

49

10

4

28

88

64

37

↓ ↓

23 49 10 4 28 88 64 37

↓ ↓

23 10 49 4 28 88 64 37

↓ ↓

23 10 49 4 28 88 64 37

↓ ↓

23 10 4 49 28 88 64 37

23 10 4 49 28 88 64 37



23 10 4 28 49 88 64 37



23 10 4 28 49 88 64 37



23 10 4 28 49 88 64 37



23 10 4 28 49 64 88 37



23 10 4 28 49 64 88 37



23 10 4 28 49 64 37 88

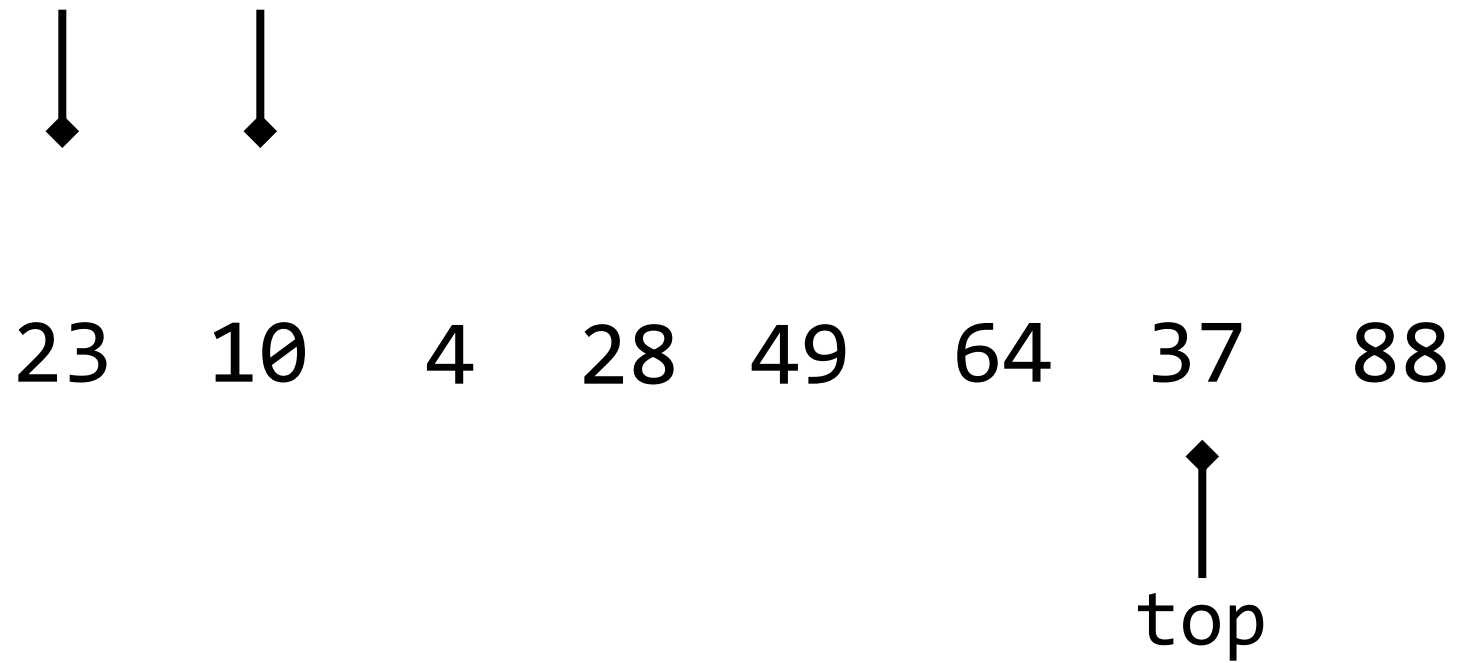


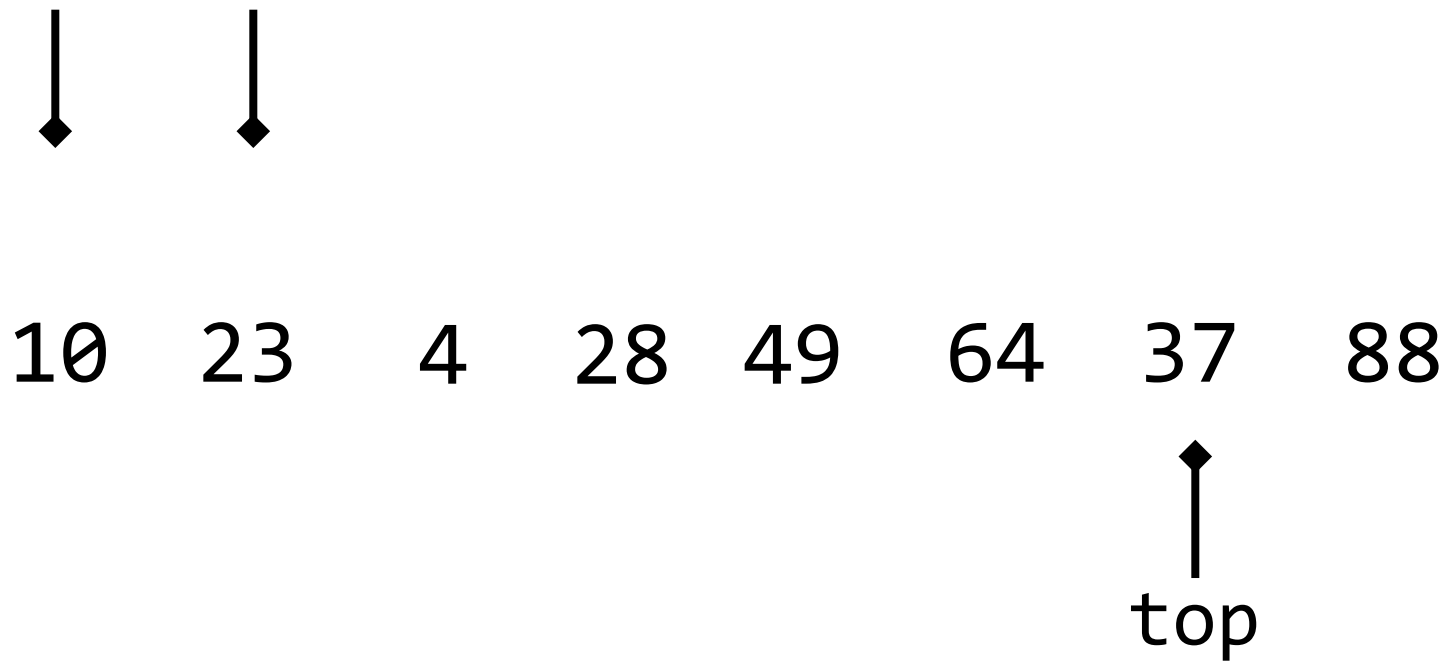
23 10 4 28 49 64 37 88

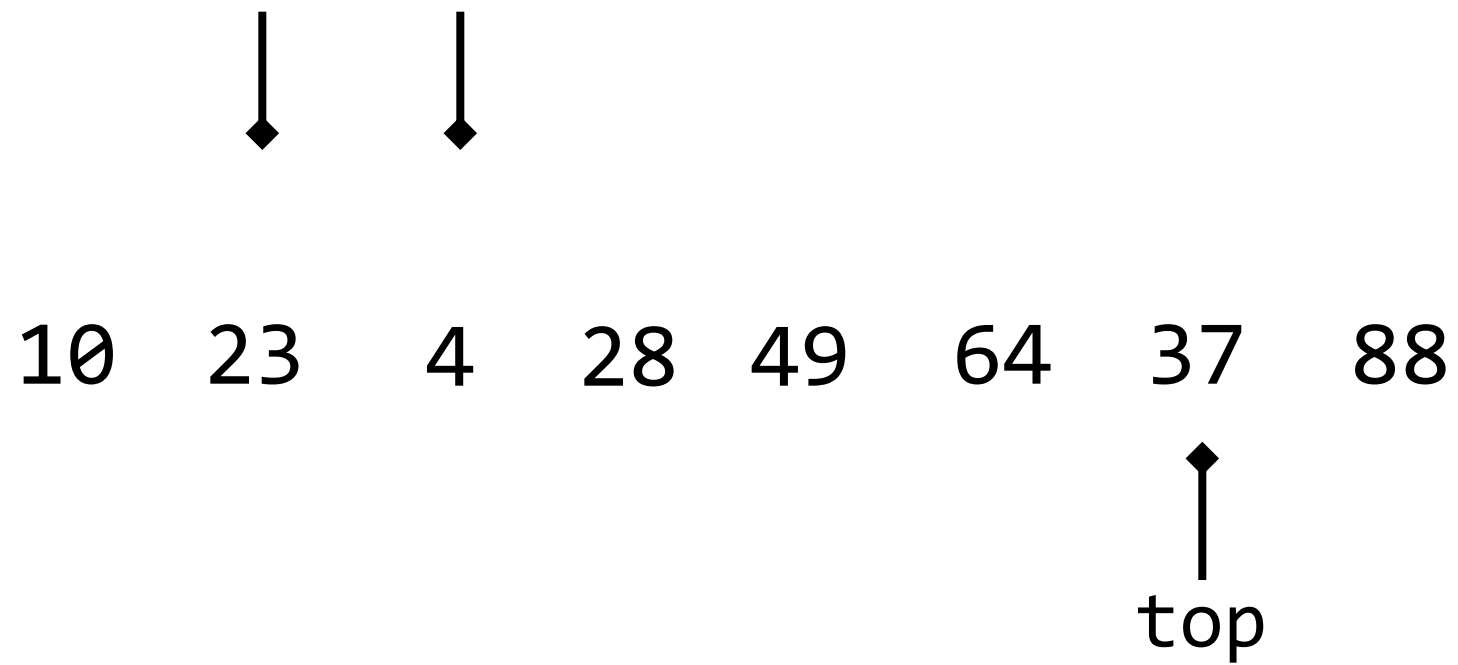
↑
top

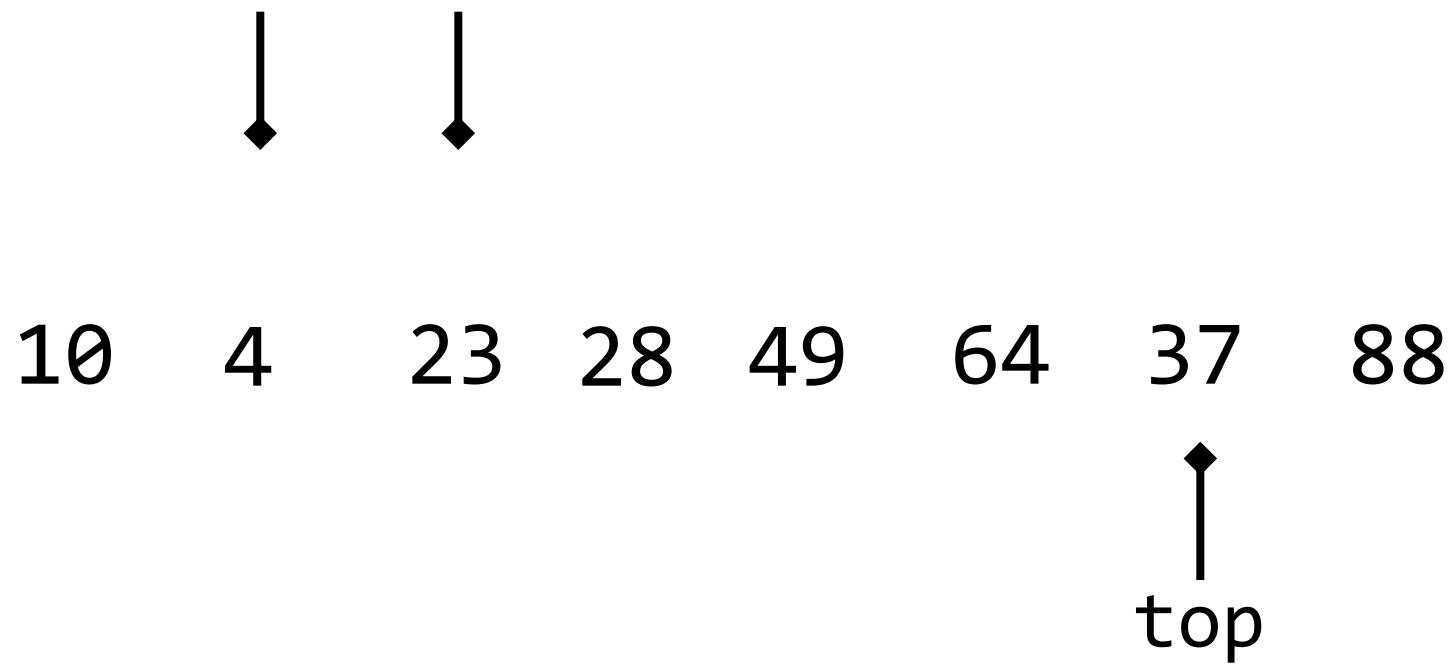
23 10 4 28 49 64 37 88

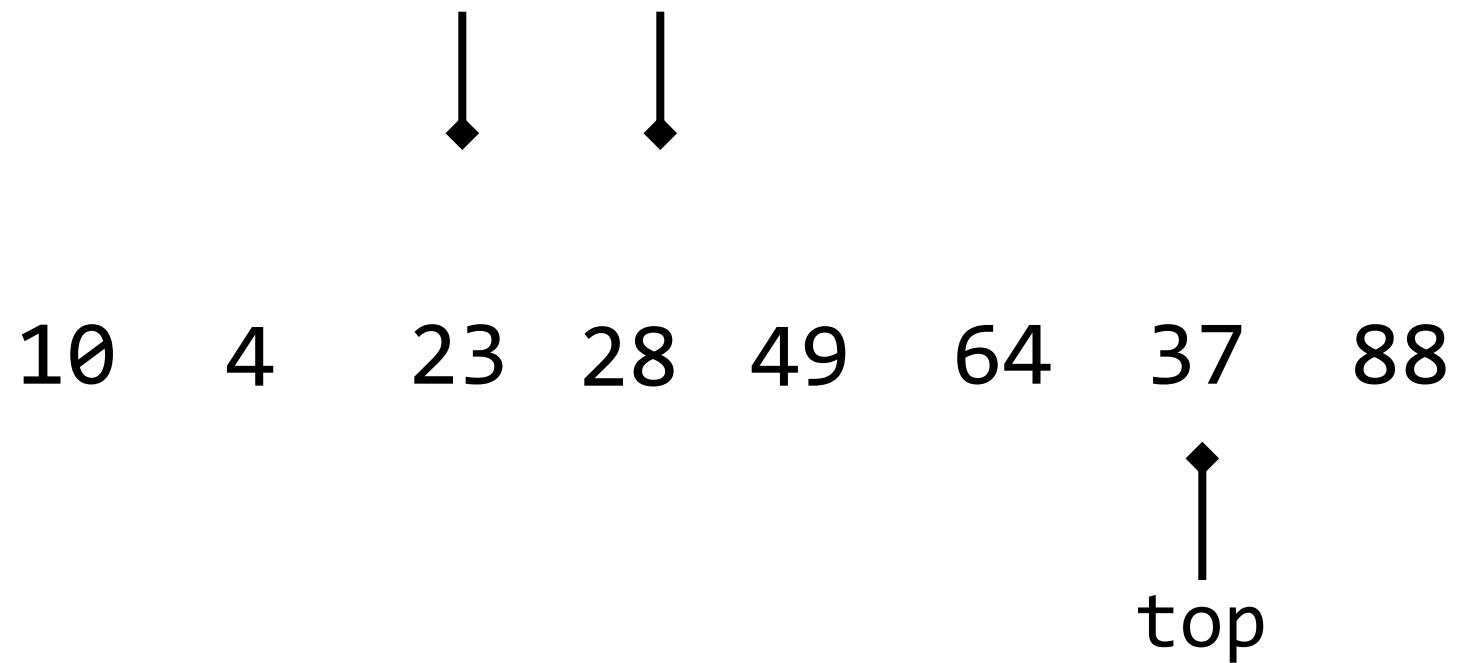
↑
top

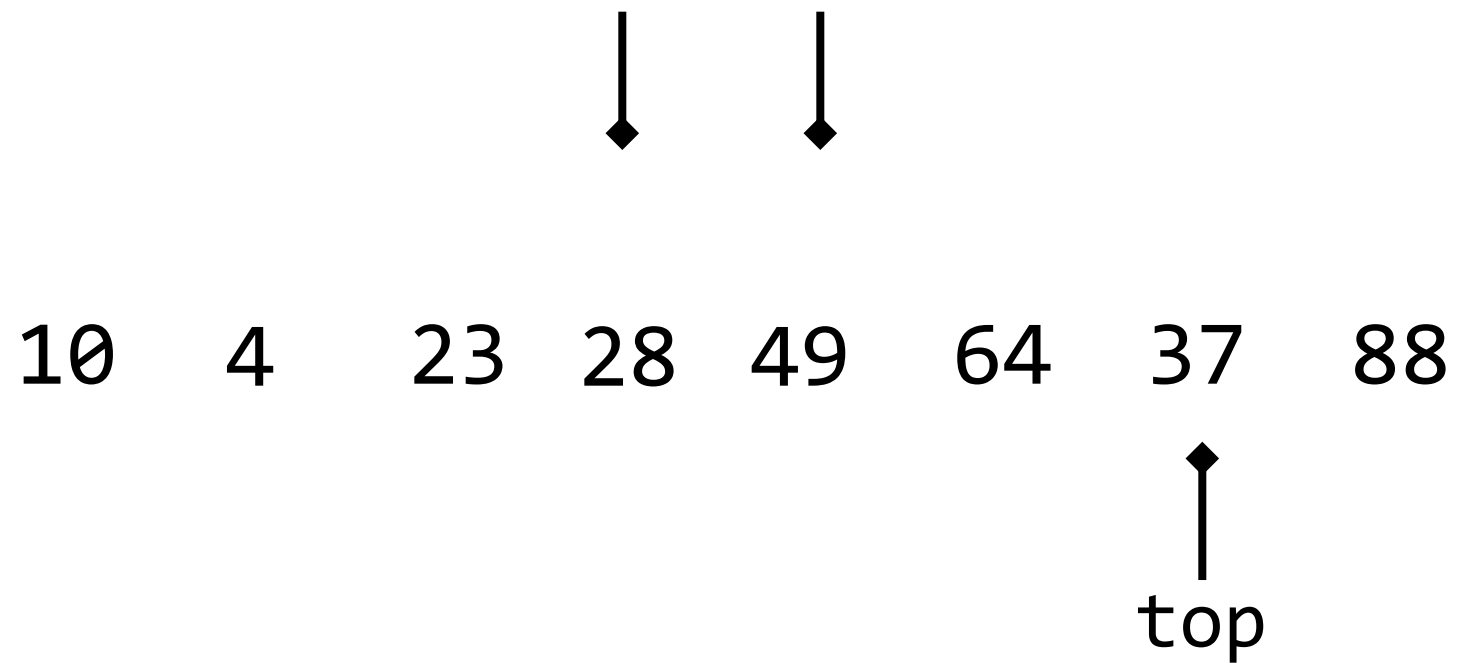


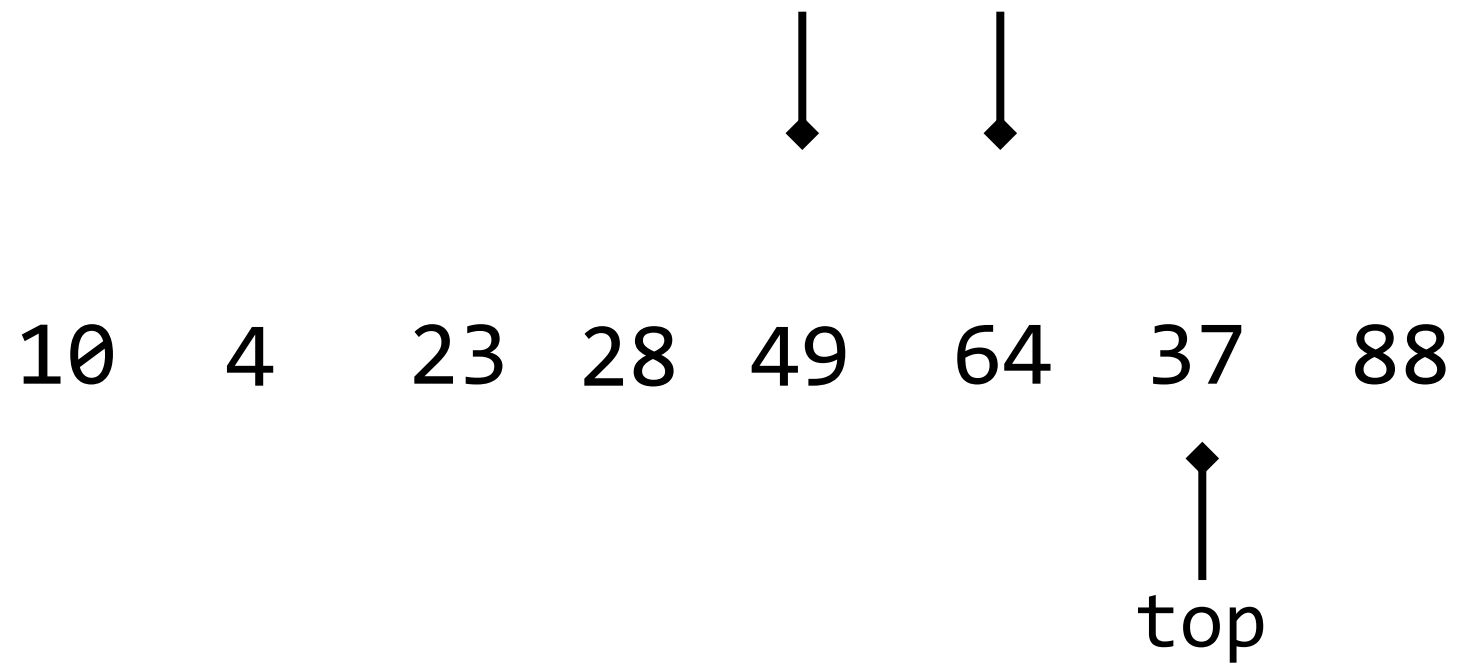


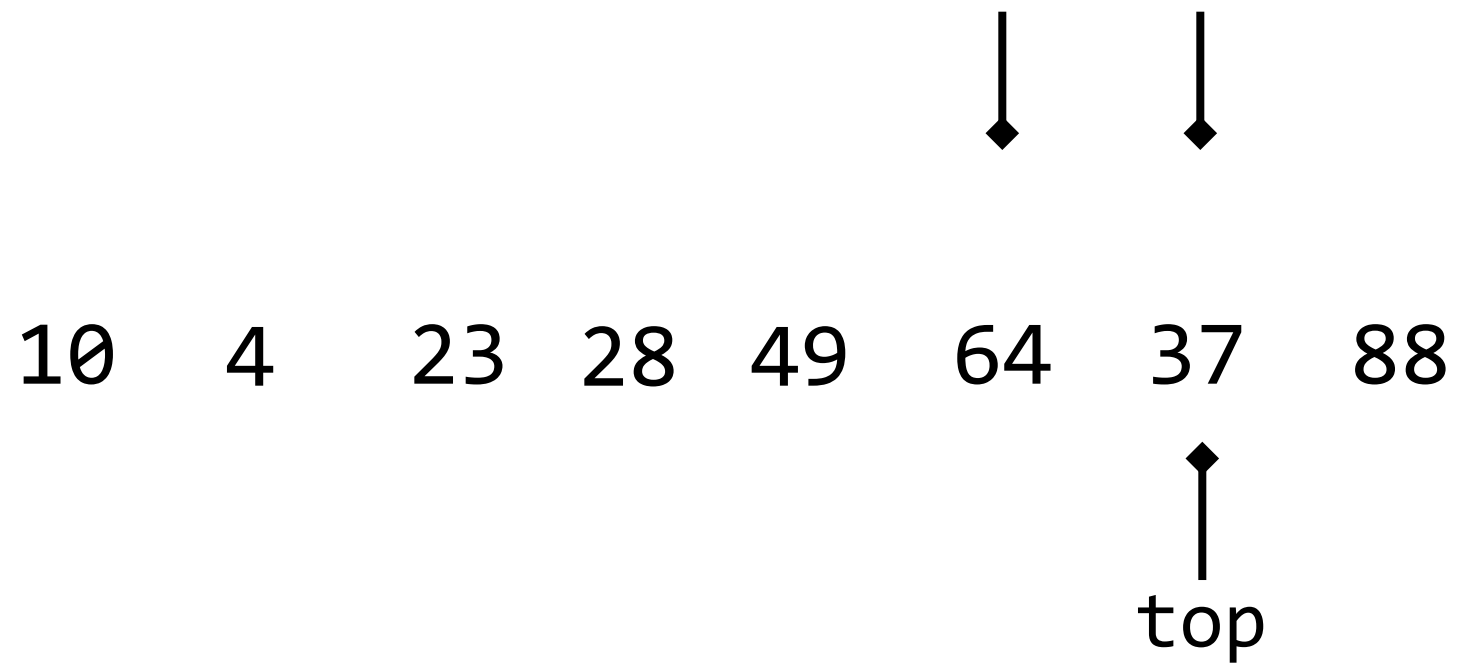


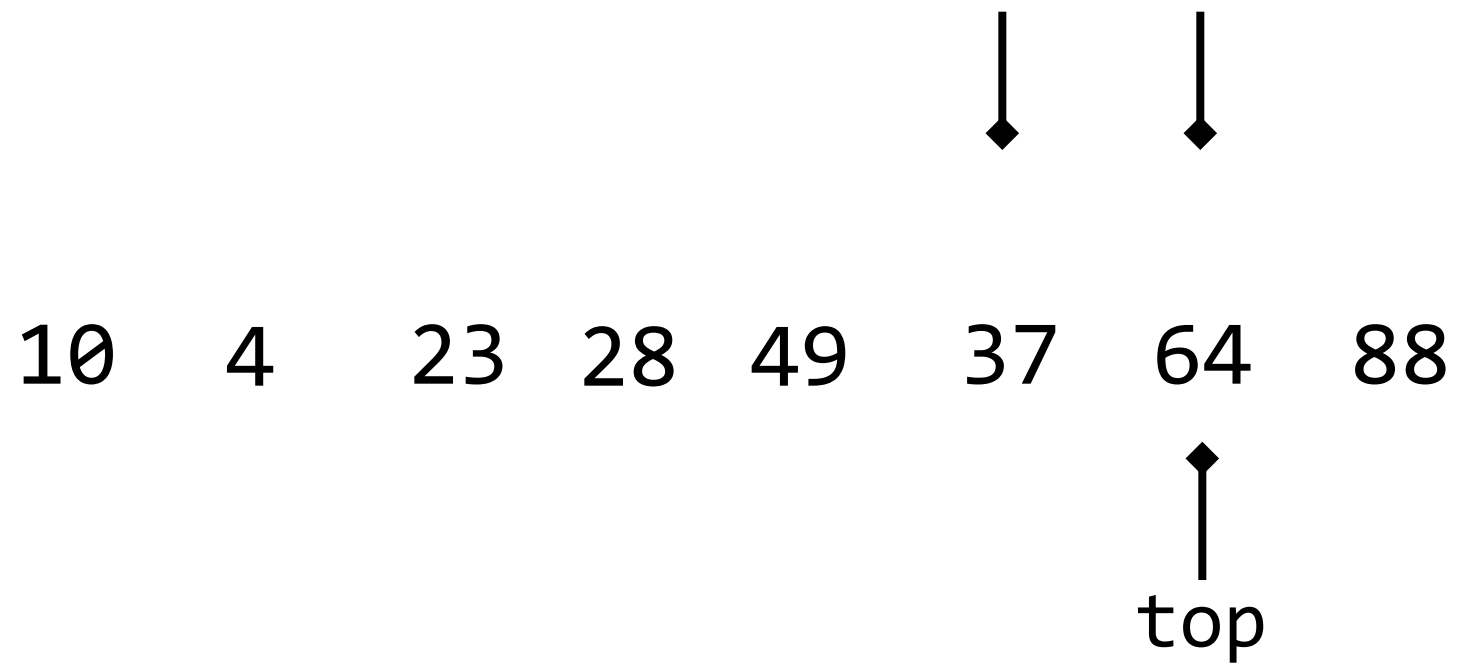










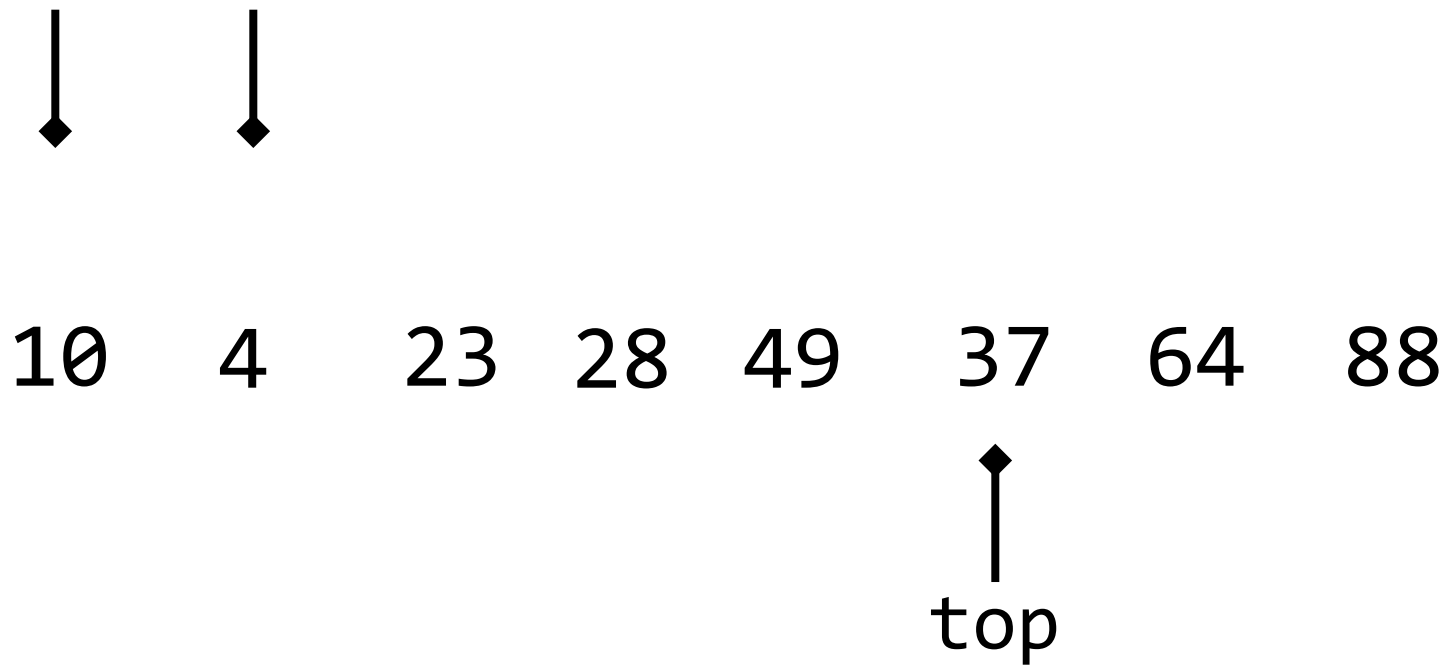


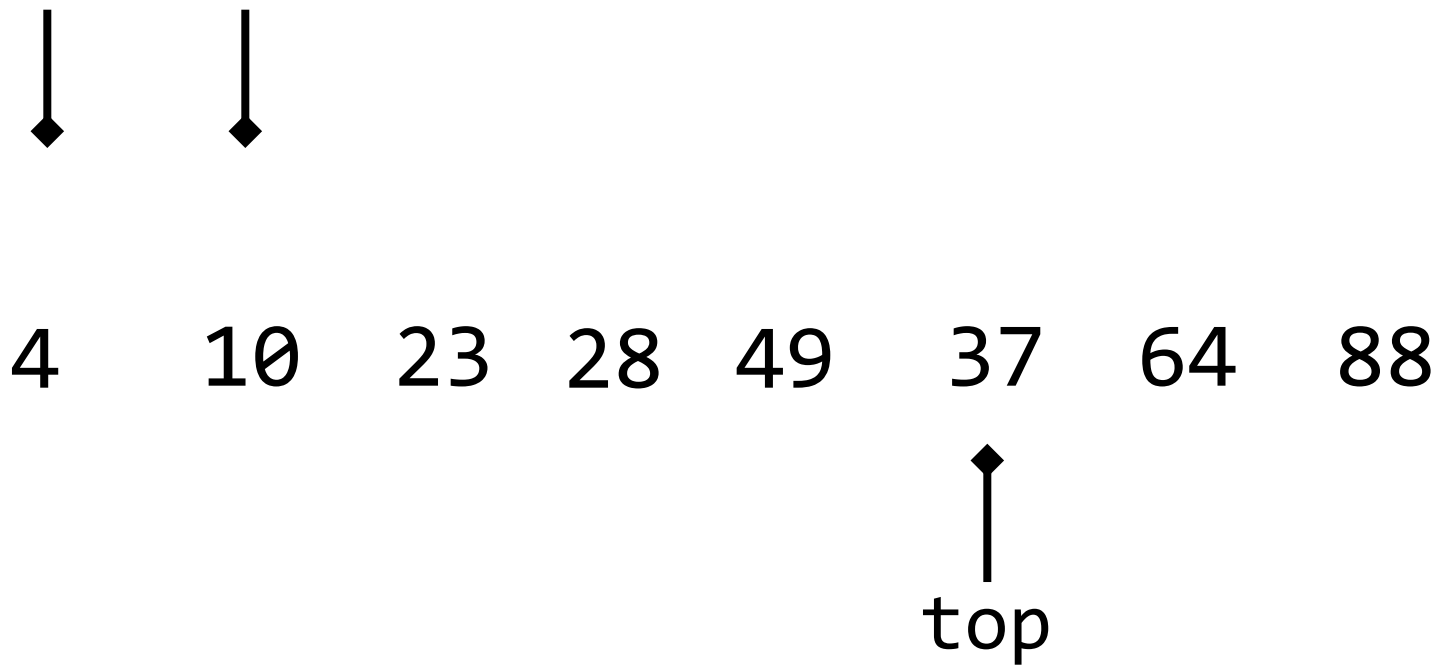
10 4 23 28 49 37 64 88

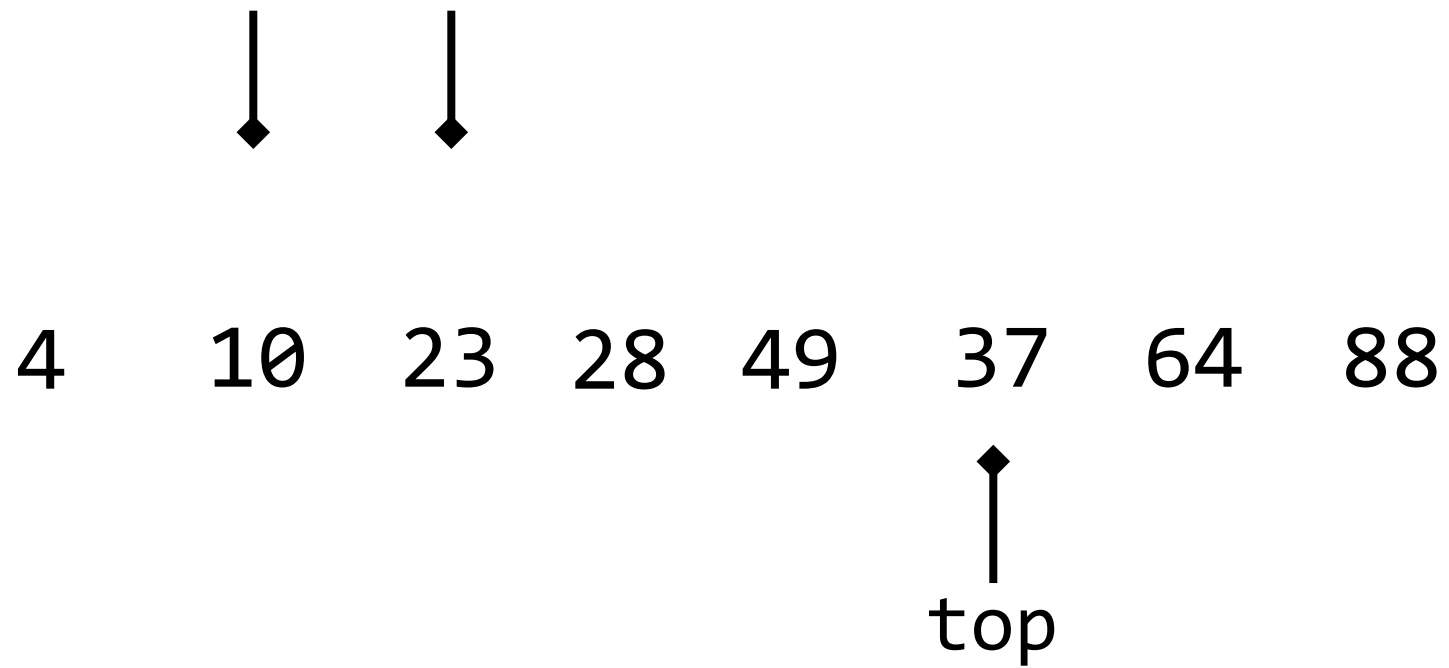
↑
top

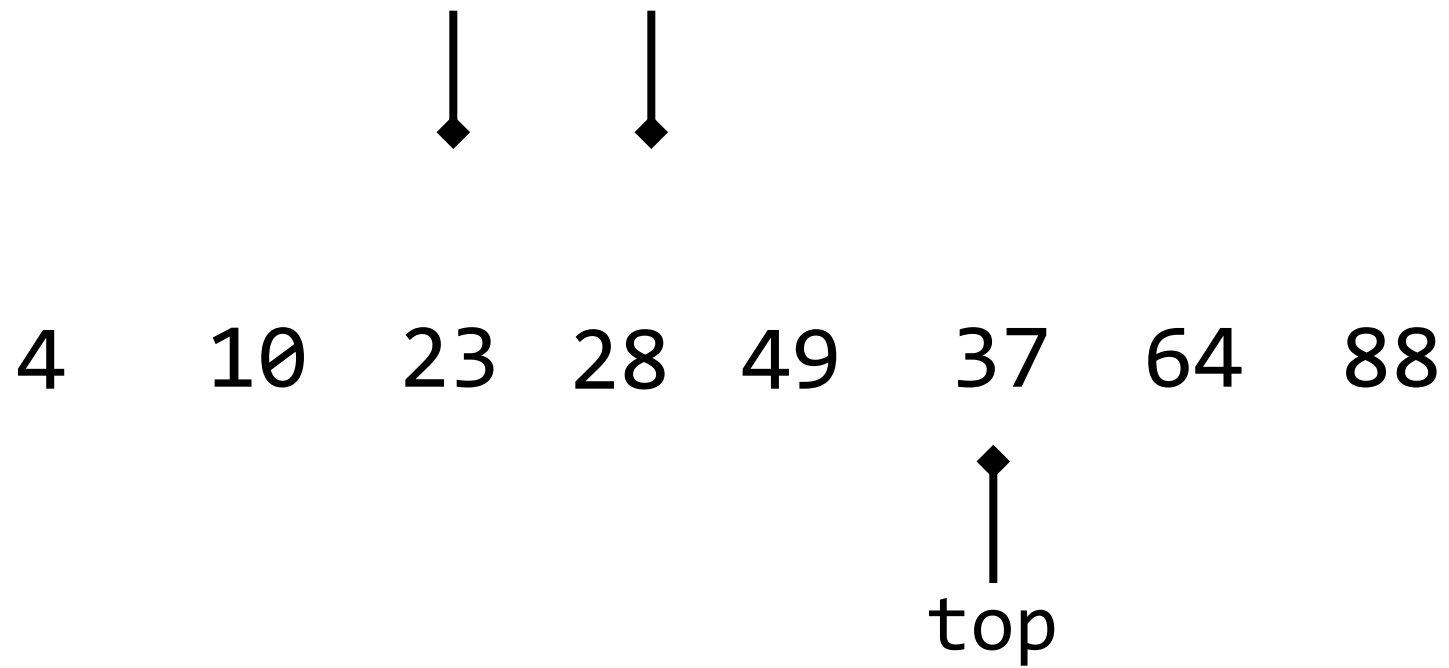
10 4 23 28 49 37 64 88

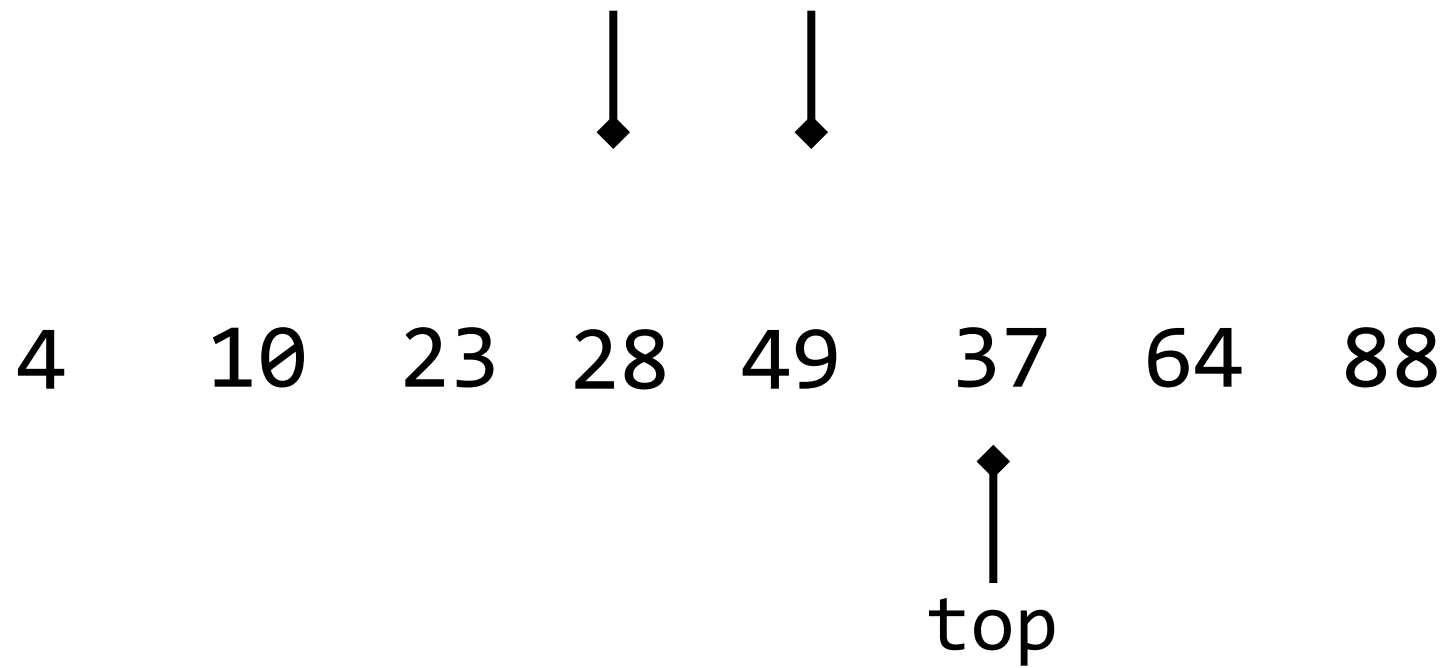
↑
top

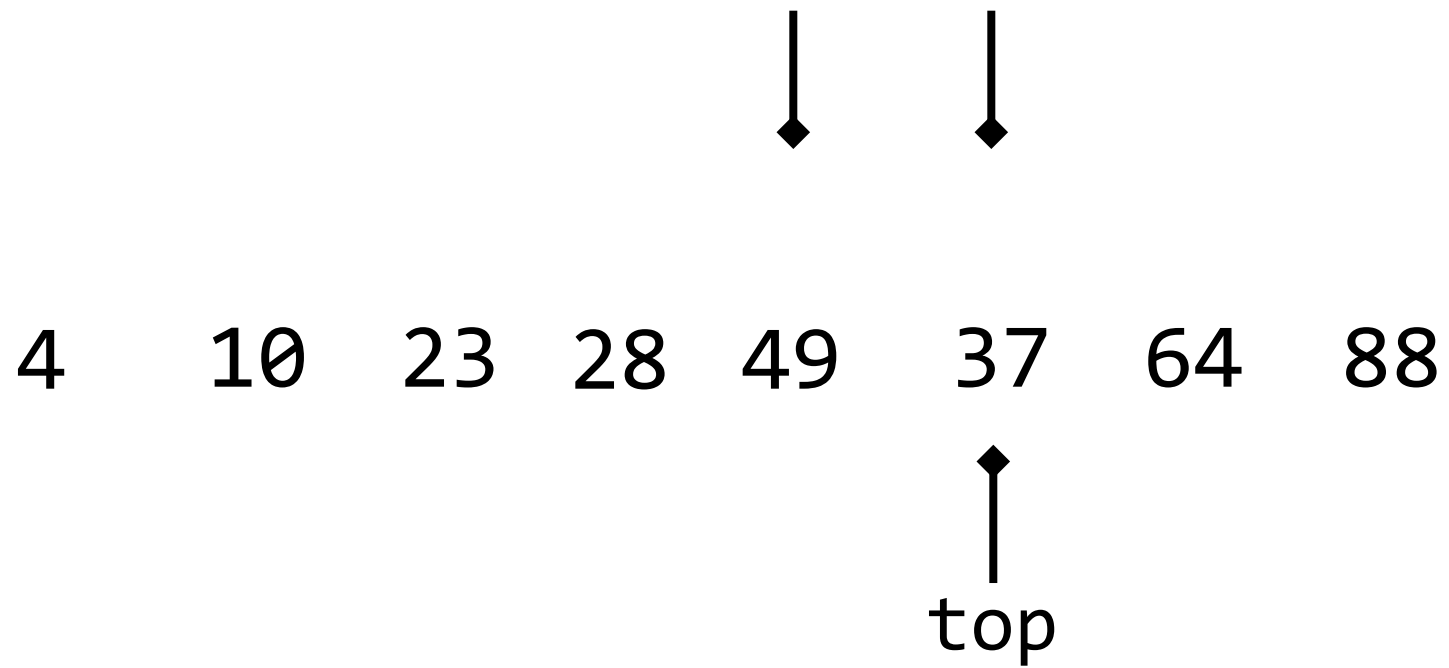


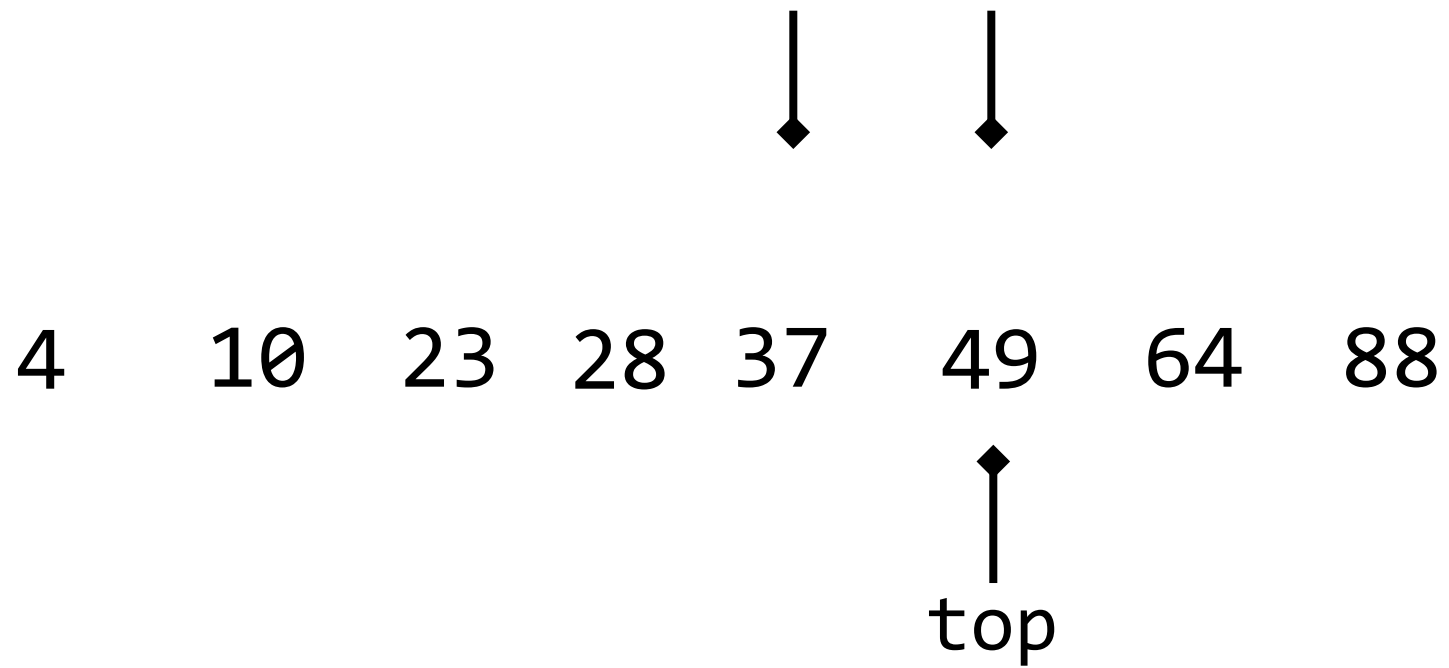










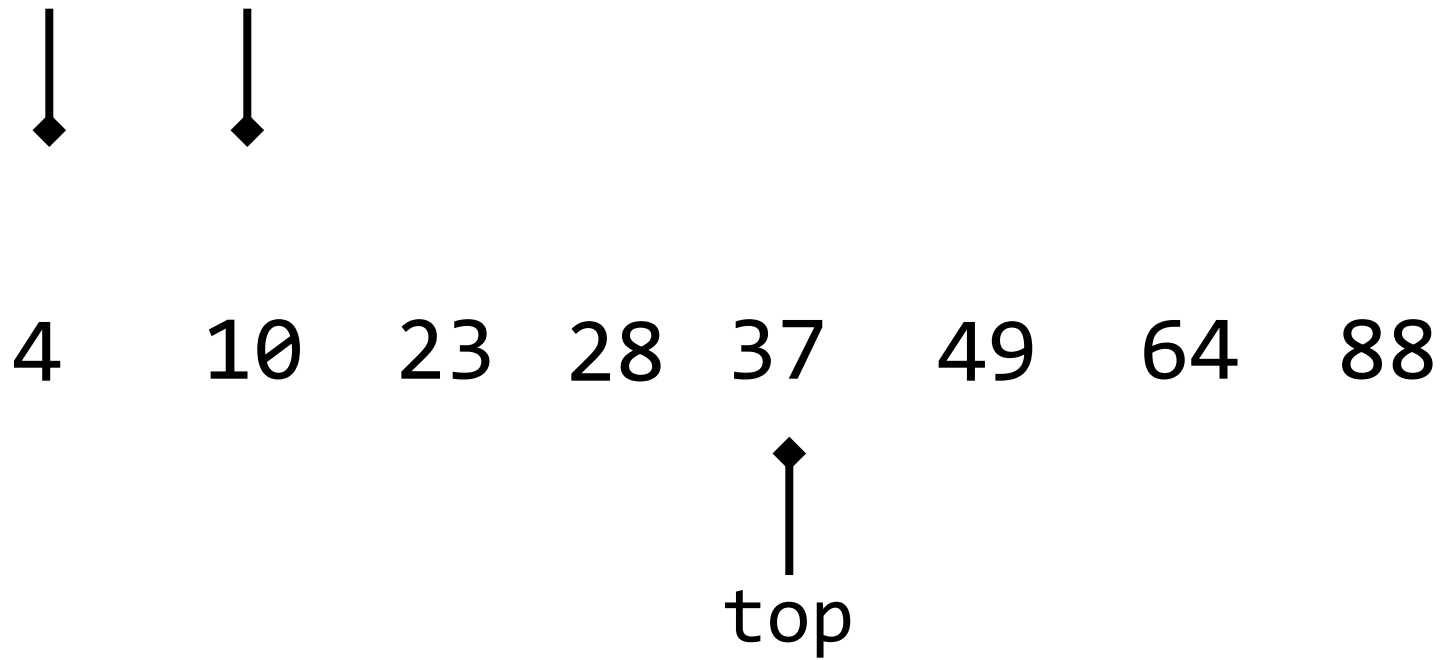


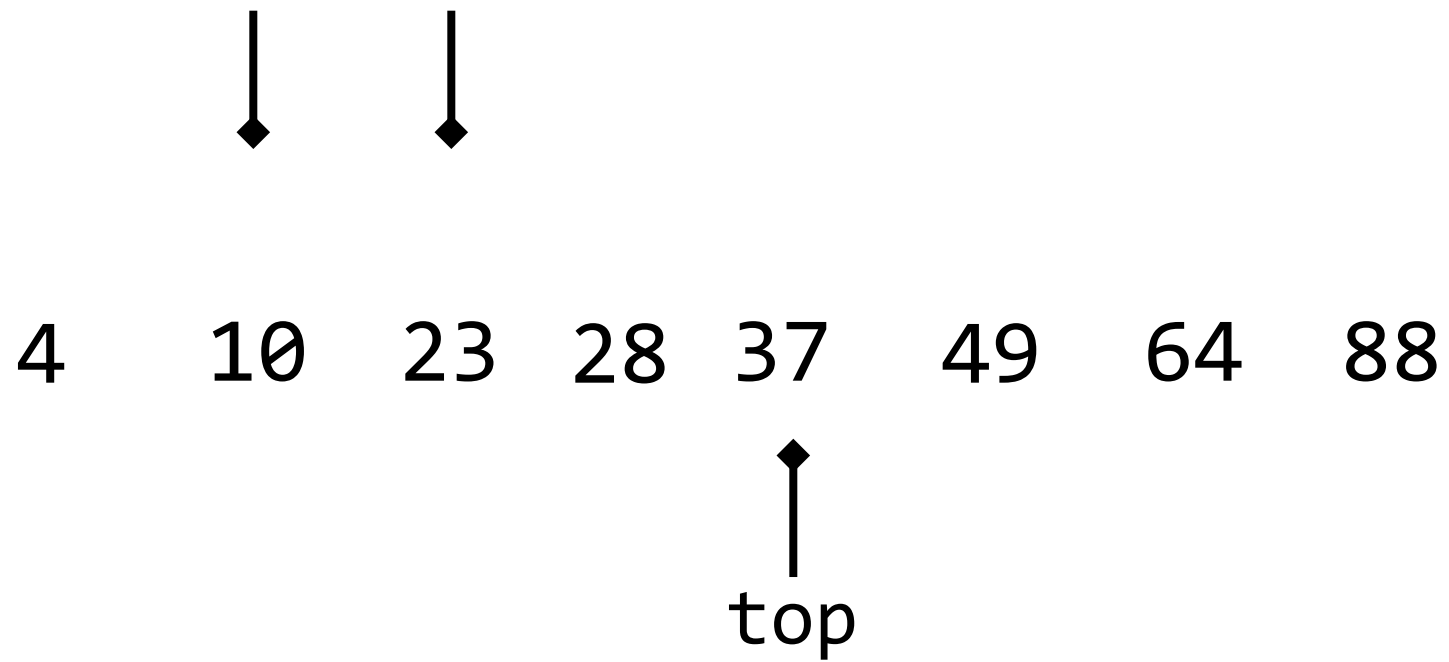
4 10 23 28 37 49 64 88

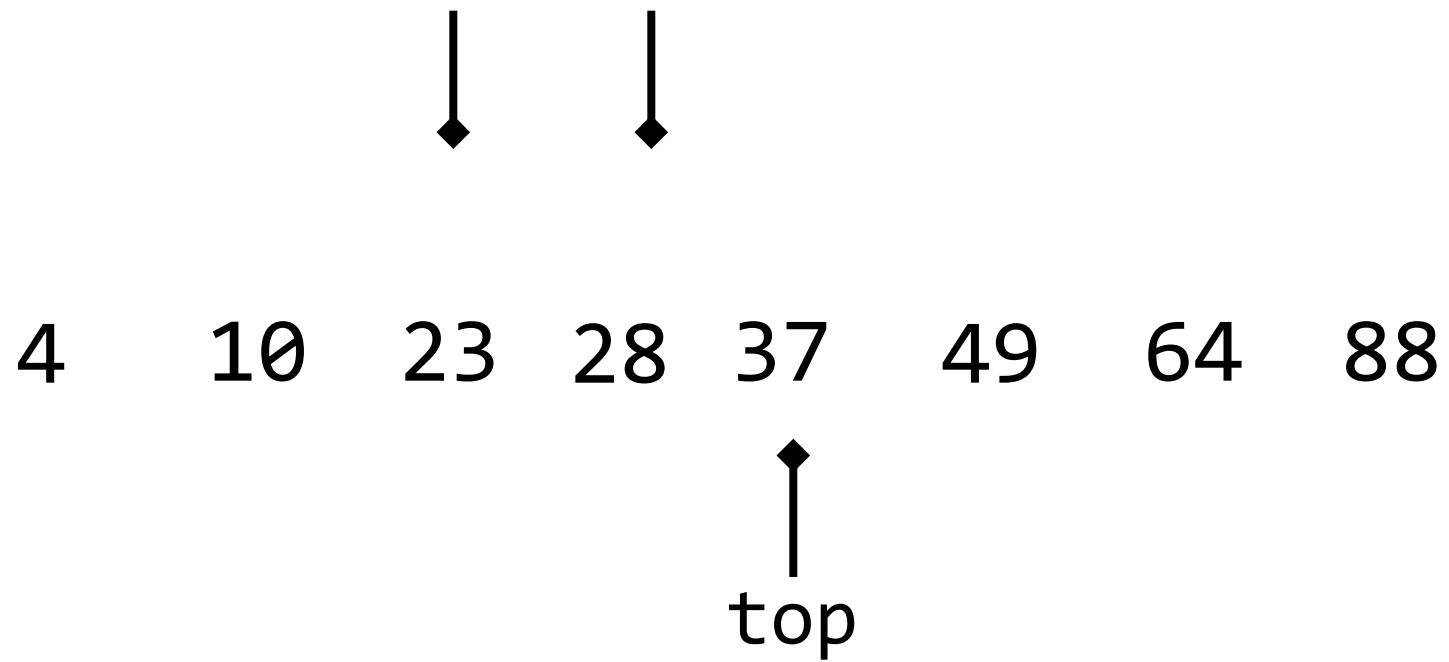
↑
top

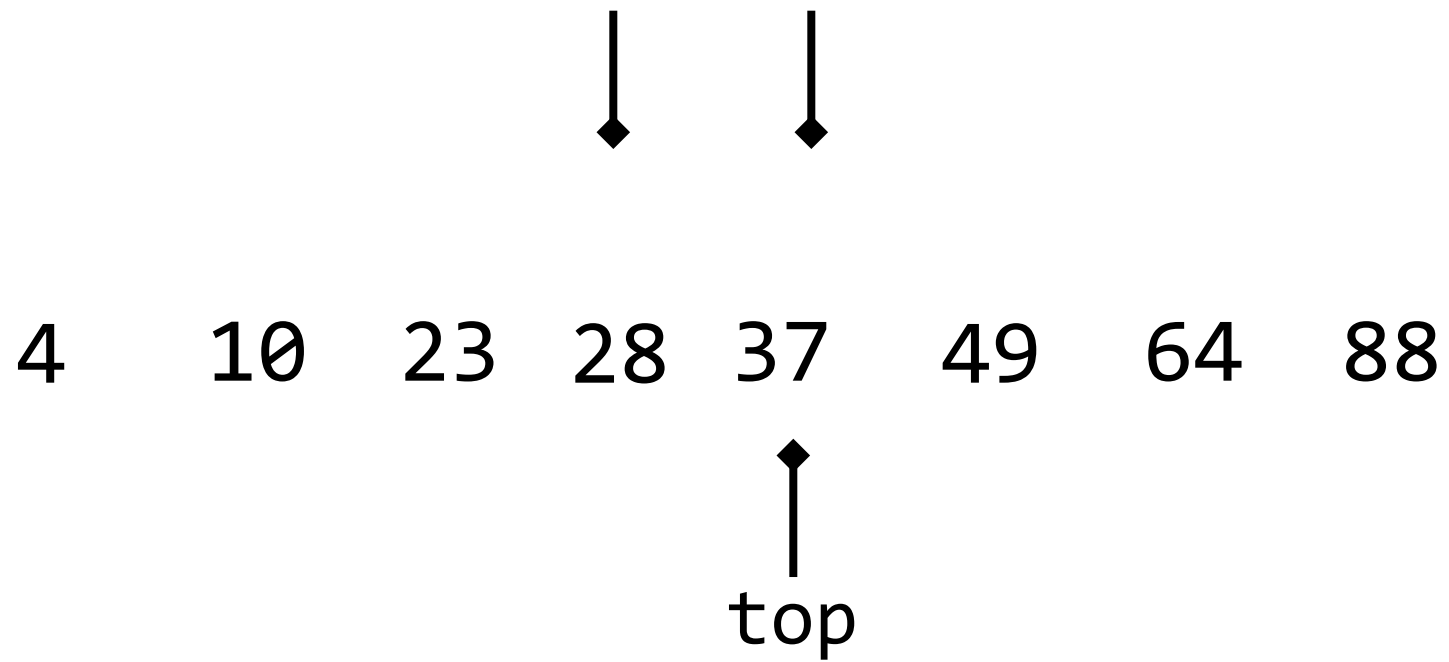
4 10 23 28 37 49 64 88

↑
top









4 10 23 28 37 49 64 88

↑
top

```
void bubbleSort(int list[], int n)
{
    bool isSorted = false;

    while (!isSorted)
    {
        isSorted = true;
        for (int i = 0; i < n - 1; i++)
        {
            if (list[i] > list[i + 1])
            {
                swap(&list[i], &list[i + 1]);
                isSorted = false;
            }
        }
    }
}
```

```

void bubbleSort(int list[], int n)
{
    bool isSorted = false;

    for (int top = n - 1; top > 0 && !isSorted; top--)
    {
        isSorted = true;
        for (int i = 0; i < top ; i++)
        {
            if (list[i] > list[i + 1])
            {
                swap(&list[i], &list[i + 1]);
                isSorted = false;
            }
        }
    }
}

```

Bubble Sort Performance

- Goes through the entire list, compares each item to every other item
- So it's approximately $n * n = n^2$ comparisons
- It does handle “nearly sorted” pretty well

Selection Sort

- Simplest of the “selection sorts”

for (each position in the list)

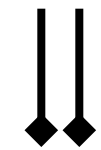
{

Starting at that position, search through
the remaining list and find the smallest item.

Swap that item to the current position.

}

Smallest: 23



23 49 10 4 28 88 64 37

Smallest: 23

↓ ↓
23 49 10 4 28 88 64 37

Smallest: 10

↓ ↓
23 49 10 4 28 88 64 37

Smallest: 4

↓ ↓
23 49 10 4 28 88 64 37

Smallest: 4

↓ ↓
23 49 10 4 28 88 64 37

Smallest: 4

↓
23 49 10 4 28 ↓ 88 64 37

Smallest: 4

↓
23 49 10 4 28 88 64 37

Smallest: 4

↓
23 49 10 4 28 88 64 37
↓

Smallest: 4



4 49 10 23 28 88 64 37

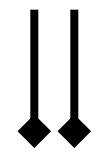
Smallest:

4 49 10 23 28 88 64 37

A black arrow points downwards from above the number 49 in the sequence.

Smallest: 49

4 49 10 23 28 88 64 37

Two vertical arrows pointing downwards, one on the left and one on the right of the number 49, indicating it is the smallest value in the sequence.

Smallest: 10

4 ↓ ↓ 23 28 88 64 37
49 10

Smallest: 10

4 ↓ 49 10 ↓ 23 28 88 64 37

Smallest: 10

4 ↓ 49 10 23 ↓ 28 88 64 37

Smallest: 10

4 ↓ 49 10 23 28 ↓ 88 64 37

Smallest: 10

4 ↓ 49 10 23 28 88 ↓ 64 37

Smallest: 10

4 ↓ 49 10 23 28 88 64 ↓ 37

Smallest: 10

4 49 10 23 28 88 64 37

A black arrow points downwards from the top center of the slide towards the number 49 in the sequence of numbers.

Smallest: 10

4 10 49 23 28 88 64 37

A black arrow pointing downwards, positioned directly above the number 10 in the sequence of numbers.

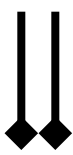
Smallest:

4 10 49 23 28 88 64 37



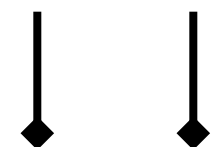
Smallest: 49

4 10 49 23 28 88 64 37

Two vertical arrows pointing downwards towards the number 49 in the sequence.

Smallest: 23

4 10 49 23 28 88 64 37



The diagram shows a sequence of eight numbers: 4, 10, 49, 23, 28, 88, 64, and 37. Two vertical arrows point downwards from above the numbers to the numbers 49 and 23, indicating they are the focus of the operation.

Smallest: 23

4 10 49 23 28 88 64 37



The diagram shows a sequence of eight numbers: 4, 10, 49, 23, 28, 88, 64, and 37. Two vertical arrows point downwards to the numbers 49 and 28, indicating a selection or comparison step in a sorting process.

Smallest: 23

4 10 49 23 28 88 64 37



The diagram shows a horizontal sequence of eight numbers: 4, 10, 49, 23, 28, 88, 64, and 37. Above the number 49, there is a vertical line with a diamond-shaped arrowhead pointing down to the number. Similarly, above the number 88, there is a vertical line with a diamond-shaped arrowhead pointing down to the number.

Smallest: 23

4 10 49 23 28 88 64 37



The diagram shows a horizontal sequence of eight numbers: 4, 10, 49, 23, 28, 88, 64, and 37. Above the number 49, there is a vertical line with a diamond-shaped arrowhead pointing down to the number. Similarly, above the number 64, there is a vertical line with a diamond-shaped arrowhead pointing down to the number.

Smallest: 23

4 10 49 23 28 88 64 37



The diagram shows a horizontal sequence of eight numbers: 4, 10, 49, 23, 28, 88, 64, and 37. Above the number 49, there is a vertical line with a diamond-shaped arrowhead pointing down to the number. Similarly, above the number 37, there is a vertical line with a diamond-shaped arrowhead pointing down to the number.

Smallest: 23

4 10 49 23 28 88 64 37



Smallest: 23

4 10 23 49 28 88 64 37



Smallest:

4 10 23 49 28 88 64 37



Smallest: 49

4 10 23 49 28 88 64 37



The image shows a sequence of numbers: 4, 10, 23, 49, 28, 88, 64, and 37. Two vertical arrows point down to the number 49, indicating it is the smallest value in the sequence.

Smallest: 28

4 10 23 49 28 88 64 37



The diagram shows a horizontal sequence of eight numbers: 4, 10, 23, 49, 28, 88, 64, and 37. Above the numbers 49 and 28, there are two vertical arrows pointing downwards. Each arrow consists of a thin vertical line and a small diamond-shaped arrowhead at the bottom, pointing directly at the number below it.

Smallest: 28

4 10 23 49 28 88 64 37



The diagram shows a horizontal sequence of eight numbers: 4, 10, 23, 49, 28, 88, 64, and 37. Above the number 49, there is a vertical line with a diamond-shaped arrowhead pointing down to the number. Similarly, above the number 88, there is a vertical line with a diamond-shaped arrowhead pointing down to the number.

Smallest: 28

4 10 23 49 28 88 64 37



The diagram shows a horizontal sequence of eight numbers: 4, 10, 23, 49, 28, 88, 64, and 37. Above the number 49, there is a vertical line with a diamond-shaped arrowhead pointing down to the number. Similarly, above the number 64, there is a vertical line with a diamond-shaped arrowhead pointing down to the number.

Smallest: 28

4 10 23 49 28 88 64 37



The diagram shows a horizontal sequence of eight numbers: 4, 10, 23, 49, 28, 88, 64, and 37. Above the number 49, there is a vertical line with a diamond-shaped arrowhead pointing down to the number. Similarly, above the number 37, there is a vertical line with a diamond-shaped arrowhead pointing down to the number.

Smallest: 28

4 10 23 49 28 88 64 37



Smallest: 28

4 10 23 28 49 88 64 37



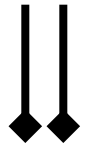
Smallest:

4 10 23 28 49 88 64 37

A black arrow pointing downwards from the top center of the slide to the number 49 in the sequence.

Smallest: 49

4 10 23 28 49 88 64 37

Two vertical arrows pointing downwards, one on the left and one on the right of the number 49, highlighting it as the smallest value in the sequence.

Smallest: 49

4 10 23 28 49 88 64 37



The diagram shows a horizontal sequence of eight numbers: 4, 10, 23, 28, 49, 88, 64, and 37. Above the number 49, there is a vertical line with a diamond-shaped arrowhead pointing down to it. Similarly, above the number 88, there is a vertical line with a diamond-shaped arrowhead pointing down to it.

Smallest: 49

4 10 23 28 49 88 64 37



The diagram shows a horizontal sequence of eight numbers: 4, 10, 23, 28, 49, 88, 64, and 37. Above the number 49, there is a vertical line with a diamond-shaped arrowhead pointing down to the number. Similarly, above the number 64, there is a vertical line with a diamond-shaped arrowhead pointing down to the number.

Smallest: 37

4 10 23 28 49 88 64 37



The diagram shows a horizontal sequence of eight numbers: 4, 10, 23, 28, 49, 88, 64, and 37. Above the number 49, there is a vertical line with a diamond-shaped arrowhead pointing down to the number. Similarly, above the number 37, there is a vertical line with a diamond-shaped arrowhead pointing down to the number.

Smallest: 37

4 10 23 28 49 88 64 37



Smallest: 37

4 10 23 28 37 88 64 49

A black arrow points vertically downwards from the space above the number 37 to the number 37 itself.

Smallest:

4 10 23 28 37 88 64 49

A black arrow pointing downwards from the top of the slide towards the number 88 in the sequence of numbers.

Smallest: 88

4 10 23 28 37 88 64 49



The image shows a sequence of numbers: 4, 10, 23, 28, 37, 88, 64, 49. Two vertical arrows point down to the number 88, which is the smallest number in the sequence.

Smallest: 64

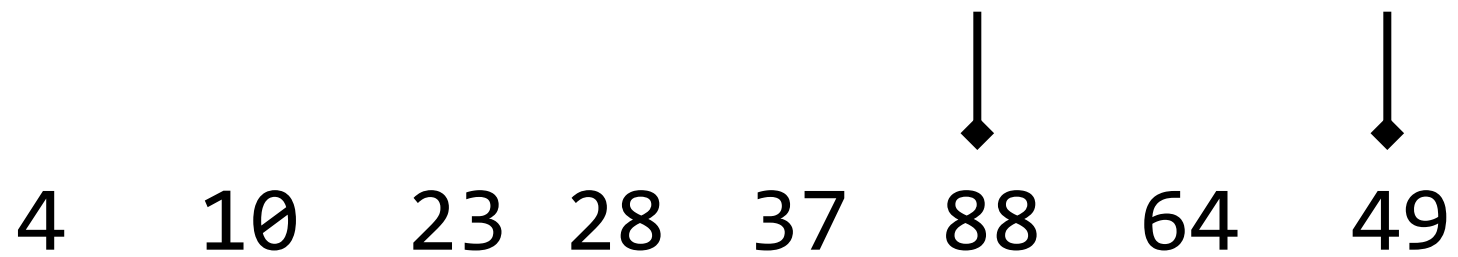
4 10 23 28 37 88 64 49



The diagram shows a horizontal sequence of numbers: 4, 10, 23, 28, 37, 88, 64, and 49. Above the number 88, there is a vertical line with a diamond-shaped arrowhead pointing down to the number. Similarly, above the number 64, there is a vertical line with a diamond-shaped arrowhead pointing down to the number.

Smallest: 49

4 10 23 28 37 88 64 49



A horizontal sequence of numbers: 4, 10, 23, 28, 37, 88, 64, 49. Two vertical arrows point downwards to the numbers 88 and 49.

Smallest: 49

4 10 23 28 37 88 64 49



Smallest: 49

4 10 23 28 37 49 64 88

A black arrow pointing downwards, positioned directly above the number 49 in the sequence.

Smallest:

4 10 23 28 37 49 64 88

A black arrow pointing downwards from the word 'Smallest:' to the number 64 in the sequence of numbers.

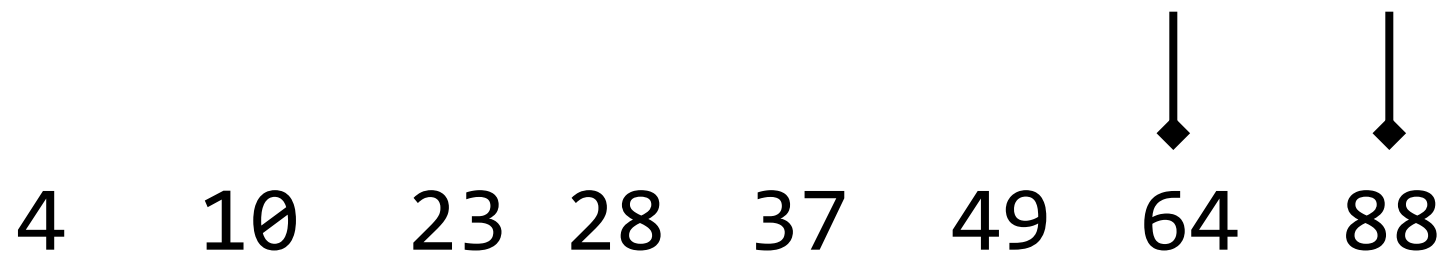
Smallest: 64

4 10 23 28 37 49 64 88

Two vertical arrows pointing downwards, one on the left and one on the right of the number 64, highlighting it as the smallest value in the sequence.

Smallest: 64

4 10 23 28 37 49 64 88



A horizontal sequence of numbers: 4, 10, 23, 28, 37, 49, 64, 88. Above the number 64, there is a vertical arrow pointing downwards. Above the number 88, there is a vertical arrow pointing downwards. The text 'Smallest: 64' is located at the top right of the image.

Smallest: 64

4 10 23 28 37 49 64 88

A black arrow pointing downwards from the text 'Smallest: 64' to the number '64' in the array.

Smallest:

4 10 23 28 37 49 64 88



```
void selectionSort(int list[], int n)
{
    for (int bot = 0; bot < n; bot++)
    {
        int indexOfSmallest = bot;
        for (int i = bot + 1; i < n; i++)
        {
            if (list[i] < list[indexOfSmallest])
            {
                indexOfSmallest = i;
            }
        }

        swap(&list[bot], &list[indexOfSmallest]);
    }
}
```

Selection Sort Performance

- Doesn't adapt; always goes through the entire list each time
- Goes through the entire list, compares each item to every other item
- So it's approximately $n * n = n^2$ comparisons
- Minimizes the number of swaps (occasionally useful)

Insertion Sort

- Simplest of the “insertion sorts”
- This is probably the sort algorithm most commonly used by normal people for day-to-day tasks

Divide the list into two parts, a “sorted” part and an “unsorted” part. (Right now, “sorted” is empty, and “unsorted” contains the entire list.)

for (each item in the unsorted list)

{

 Insert it in the appropriate place in the sorted list

}

23 49 10 4 28 88 64 37

|

Sorted

Unsorted

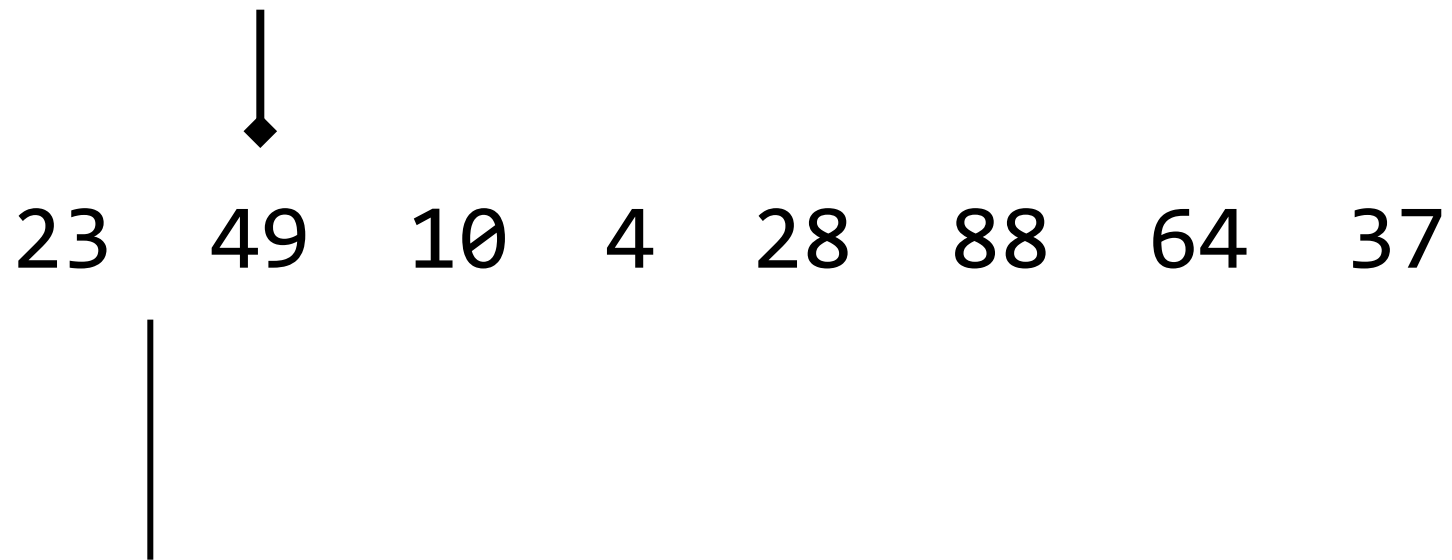


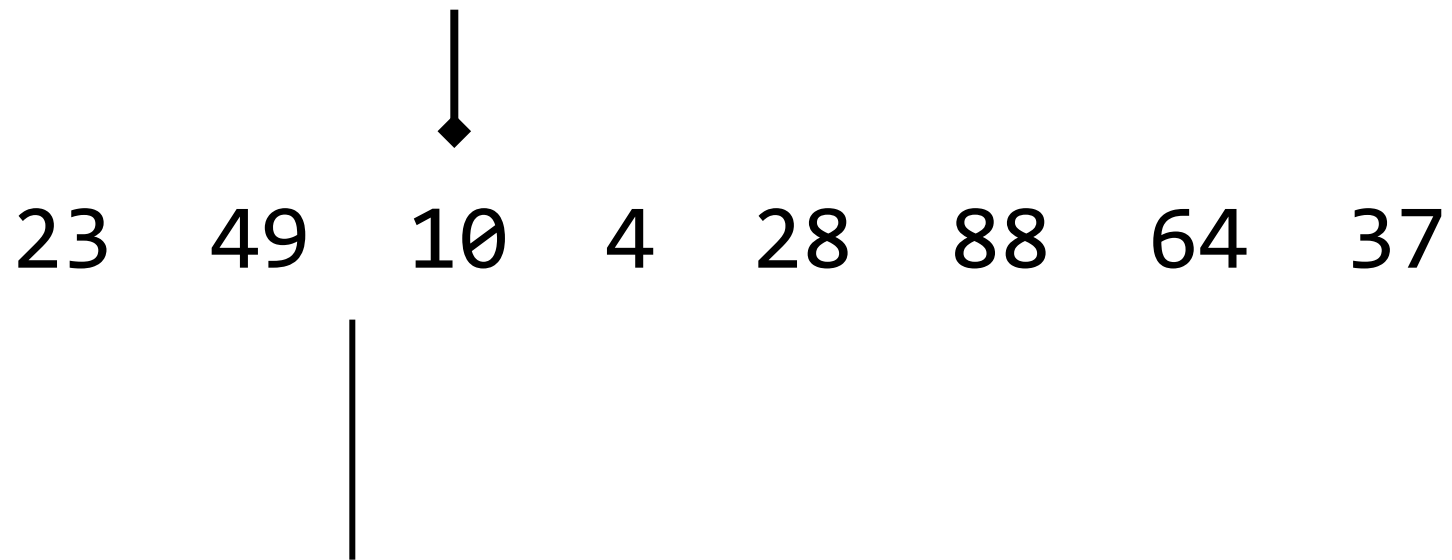
23 49 10 4 28 88 64 37



Sorted

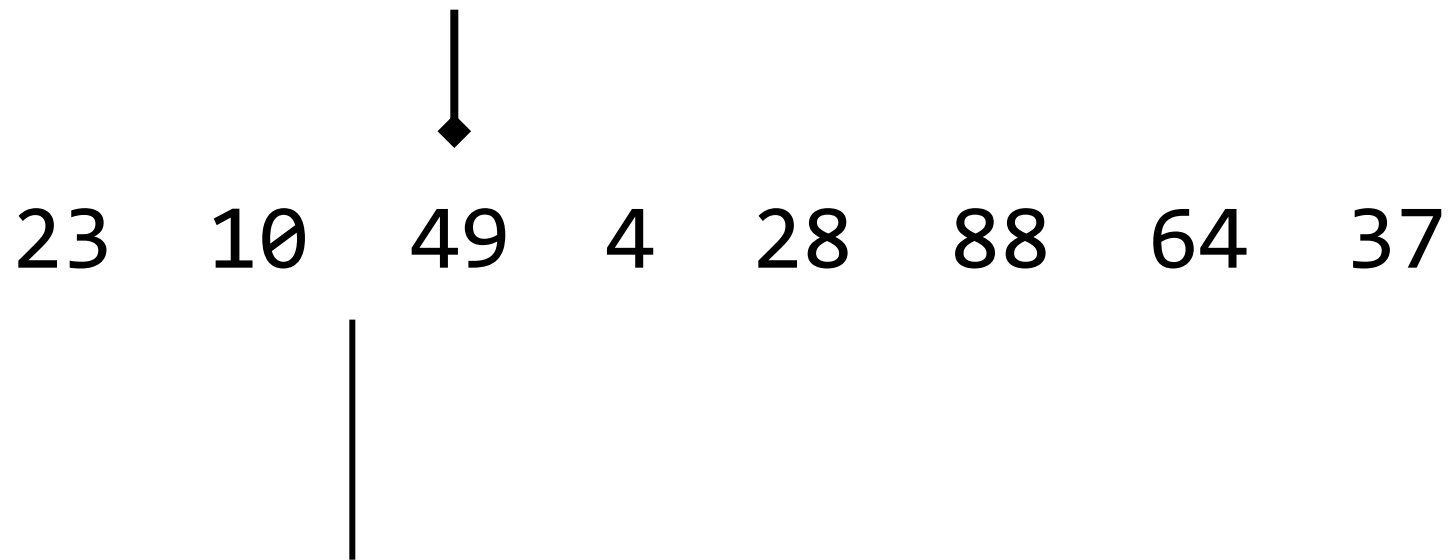
Unsorted





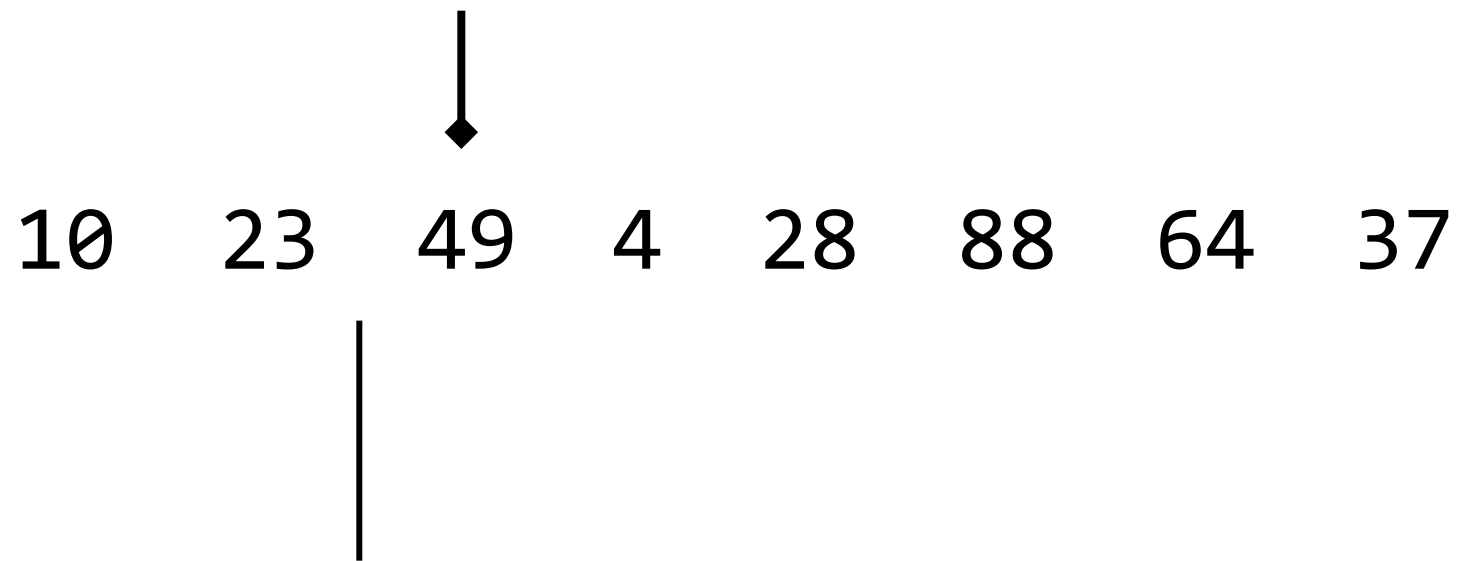
Sorted

Unsorted



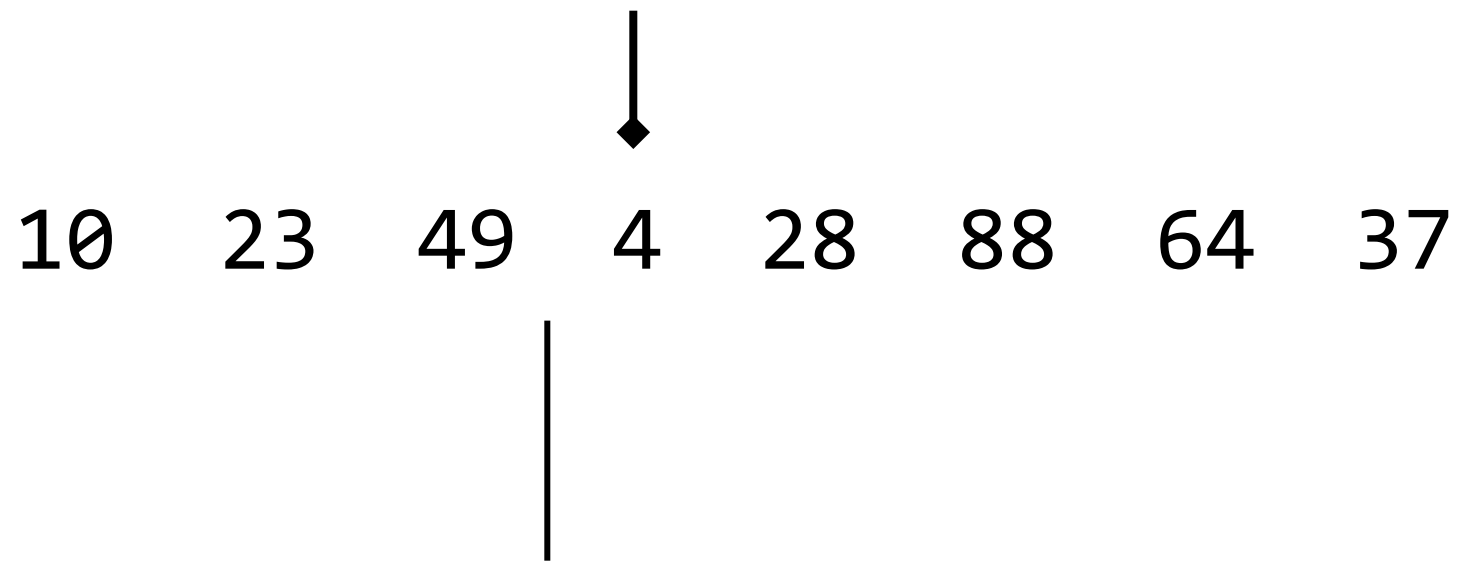
Sorted

Unsorted



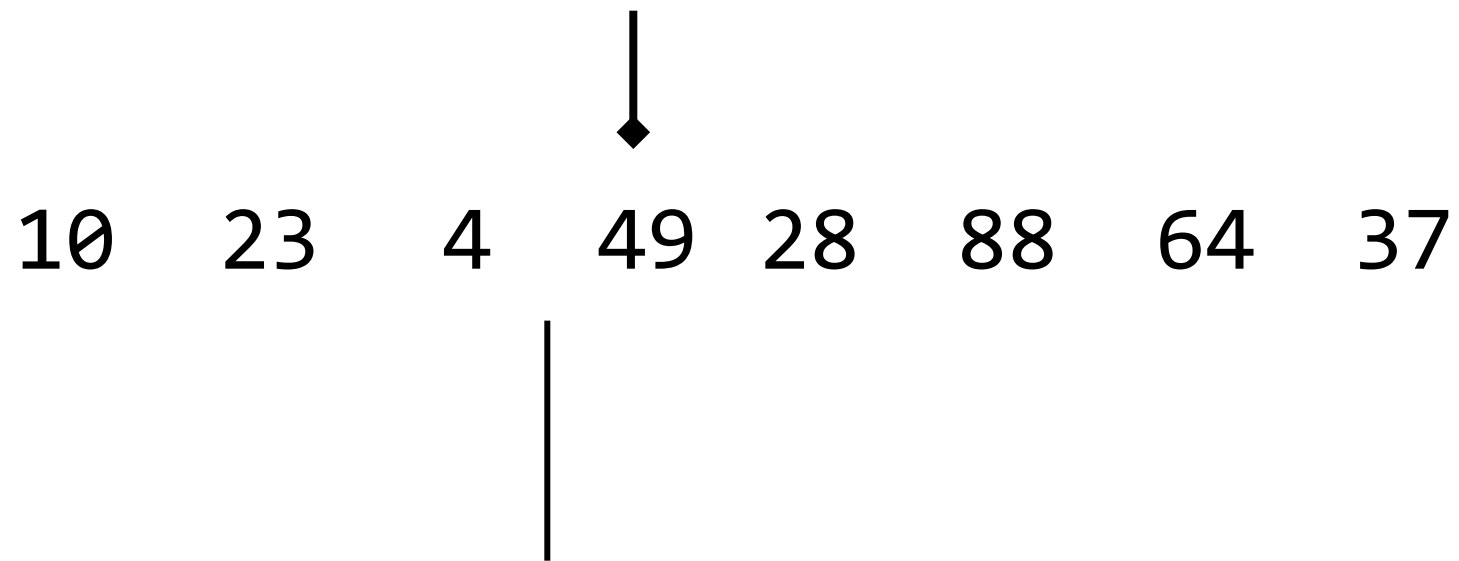
Sorted

Unsorted



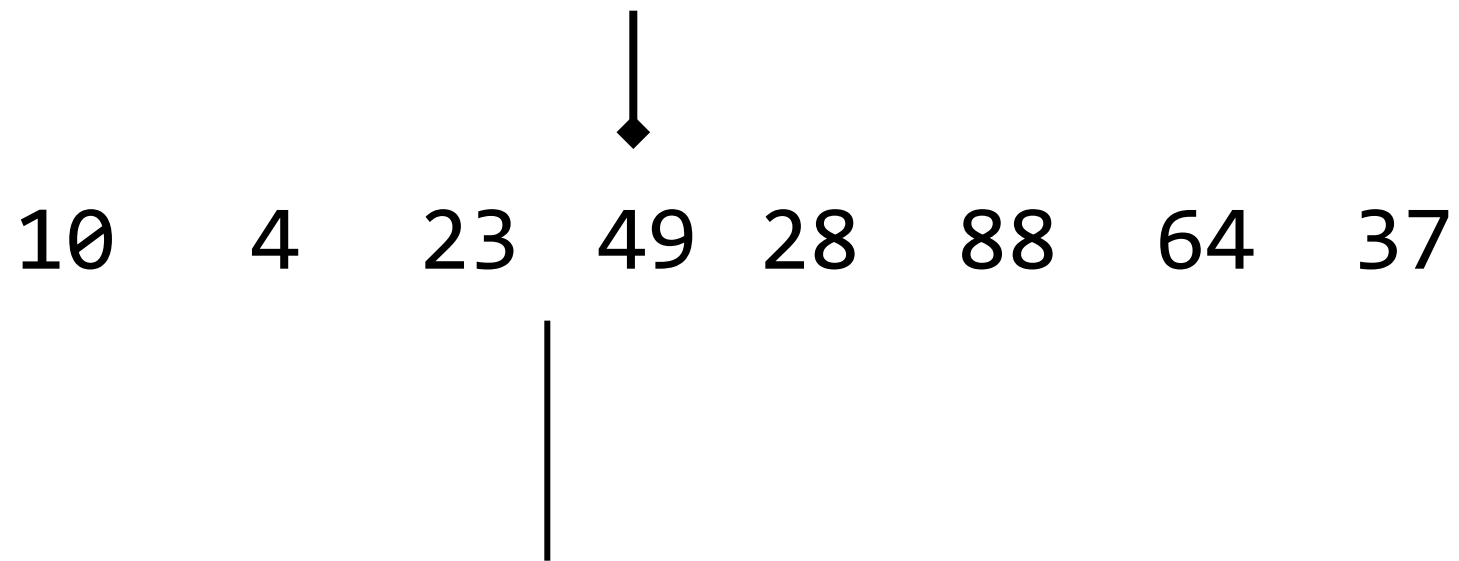
Sorted

Unsorted



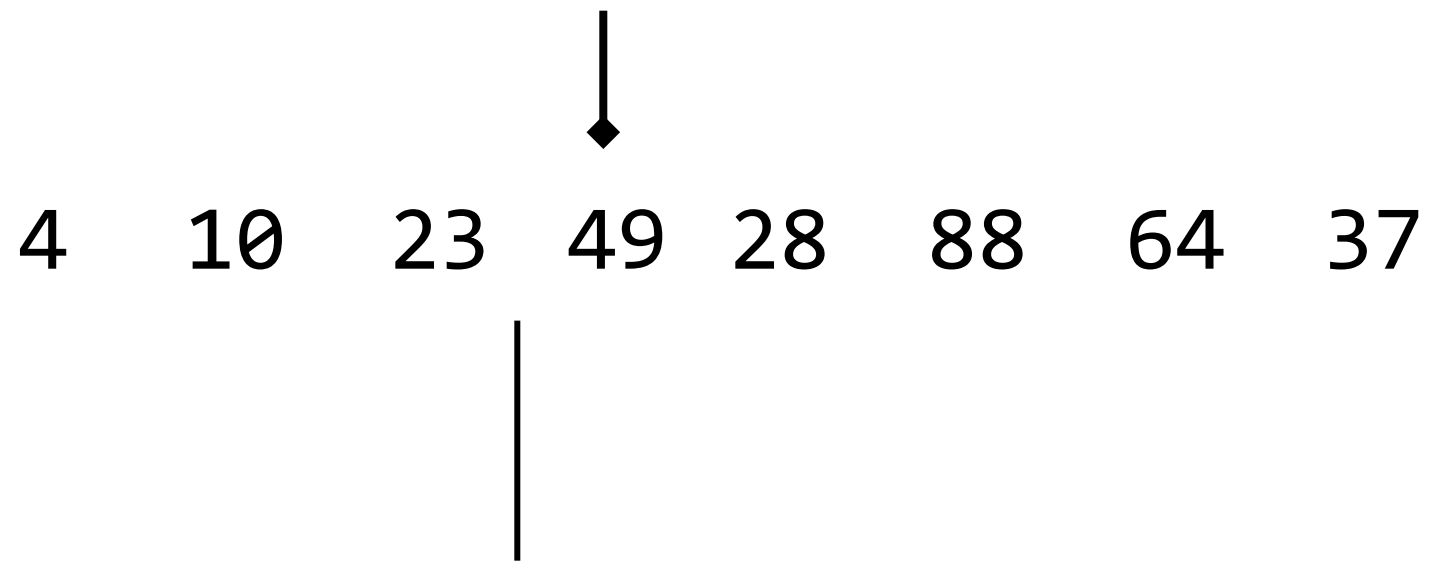
Sorted

Unsorted



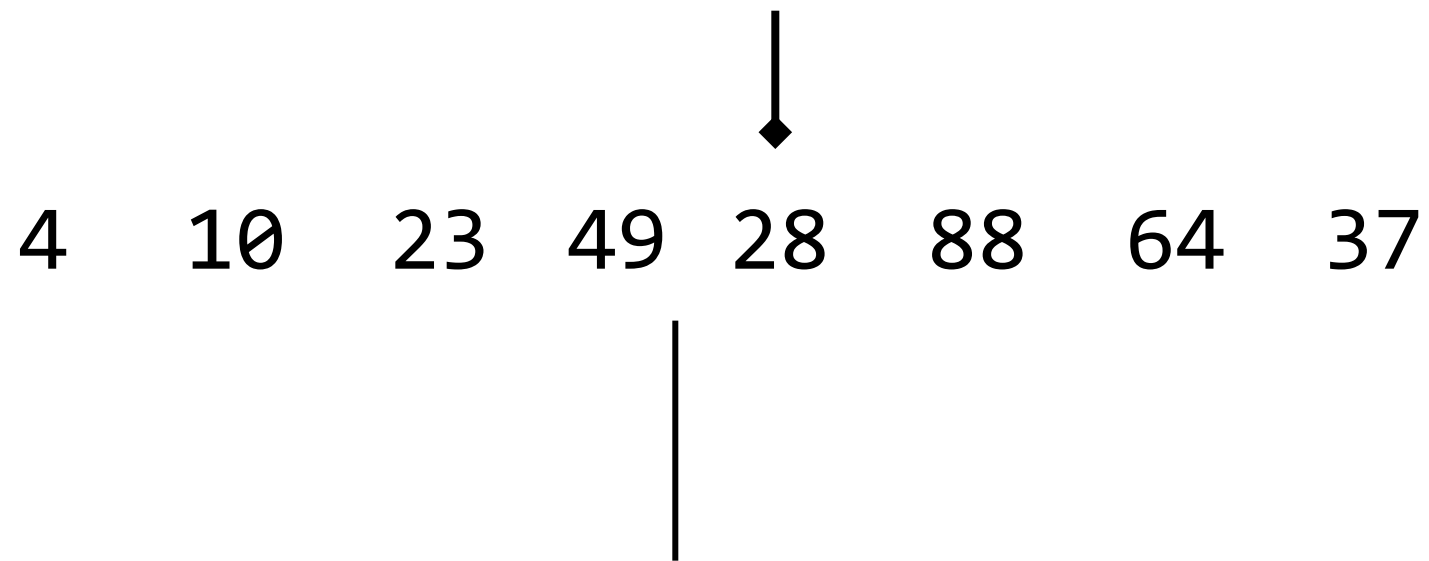
Sorted

Unsorted



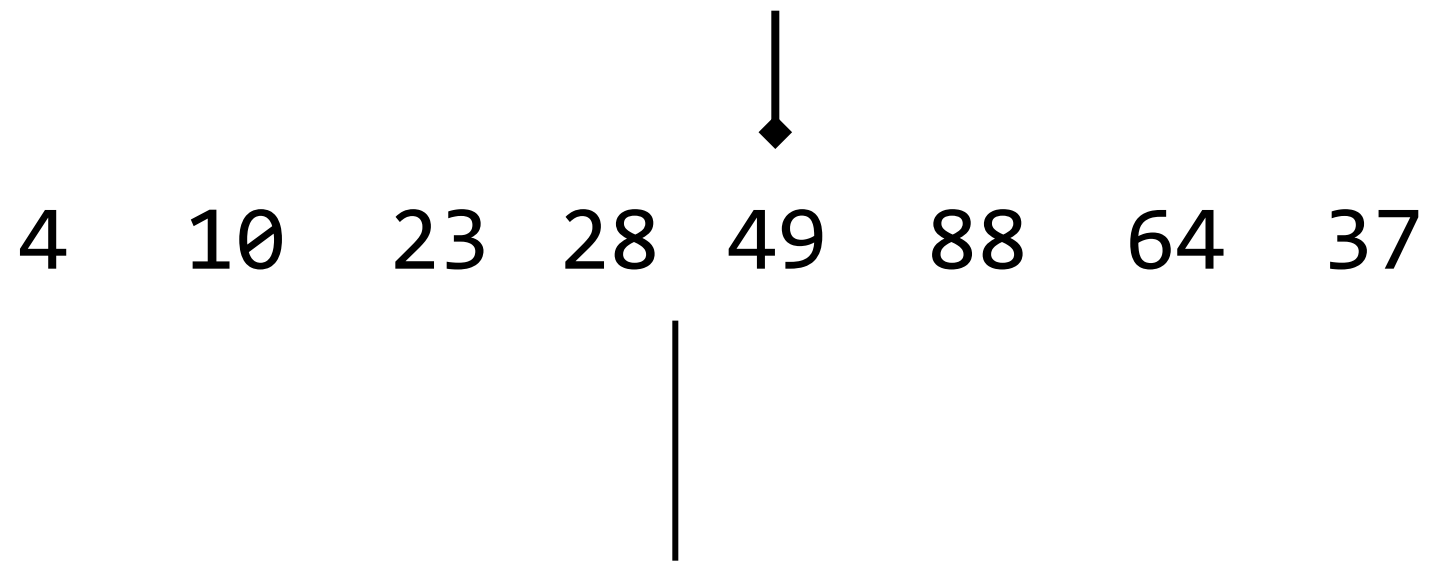
Sorted

Unsorted



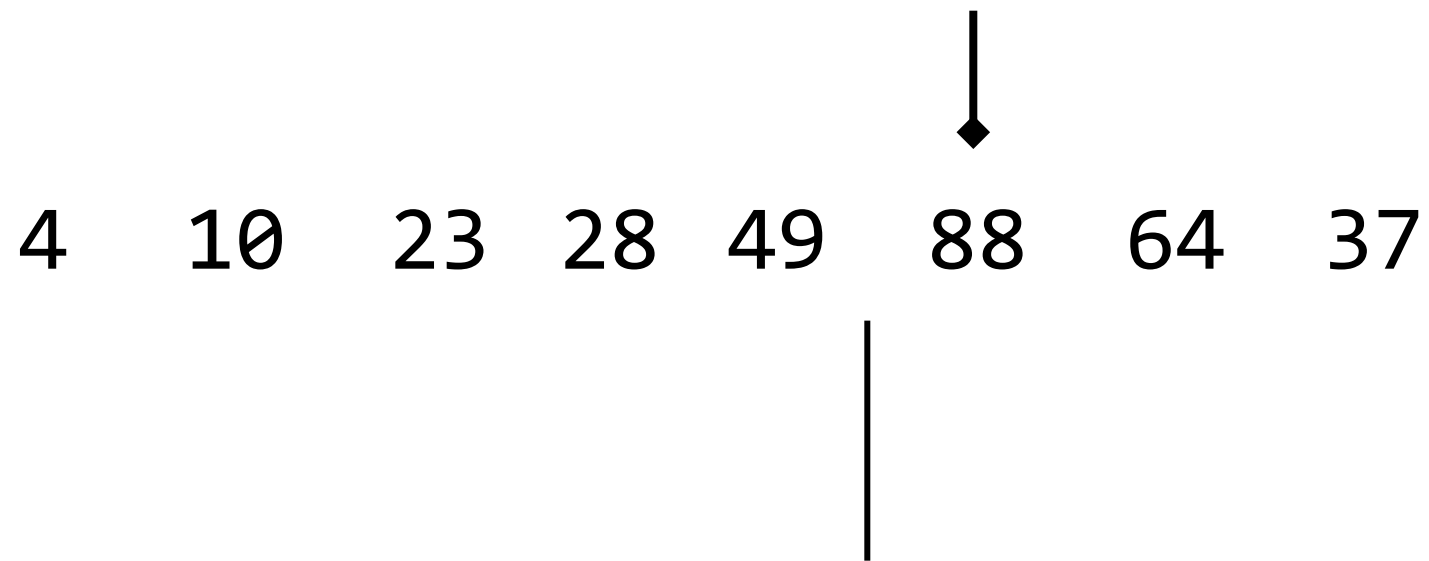
Sorted

Unsorted



Sorted

Unsorted



Sorted

Unsorted

4 10 23 28 49 88 64 37



Sorted

Unsorted

4 10 23 28 49 64 88 37



Sorted

Unsorted

4 10 23 28 49 64 88 37



Sorted

Unsorted

4 10 23 28 49 64 37 88



Sorted

Unsorted

4 10 23 28 49 37 64 88



Sorted

Unsorted

4 10 23 28 37 49 64 88



Sorted

Unsorted

4 10 23 28 37 49 64 88

|

Sorted

Unsorted

```
void insertionSort(int list[], int n)
{
    for (int bot = 1; bot < n; bot++)
    {
        for (int i = bot; i > 0 && list[i] < list[i-1]; i--)
        {
            swap(&list[i], &list[i - 1]);
        }
    }
}
```

Insertion Sort Performance

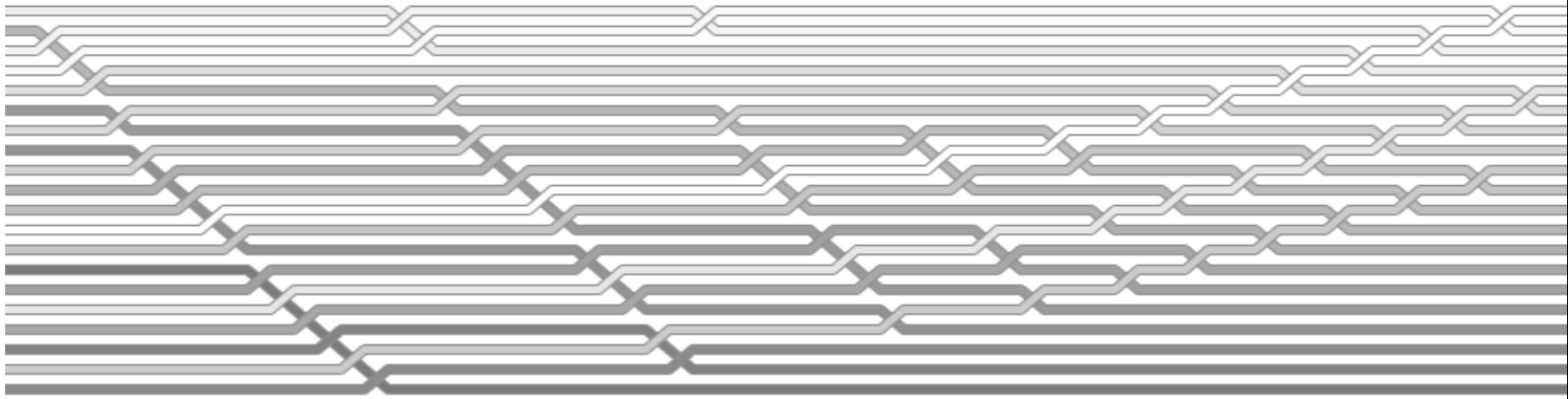
- Goes through the entire list, compares each item to every other item
- So (again) it's approximately $n * n = n^2$ comparisons worst case
- Closer to n comparisons if the list is almost sorted

Bogosort

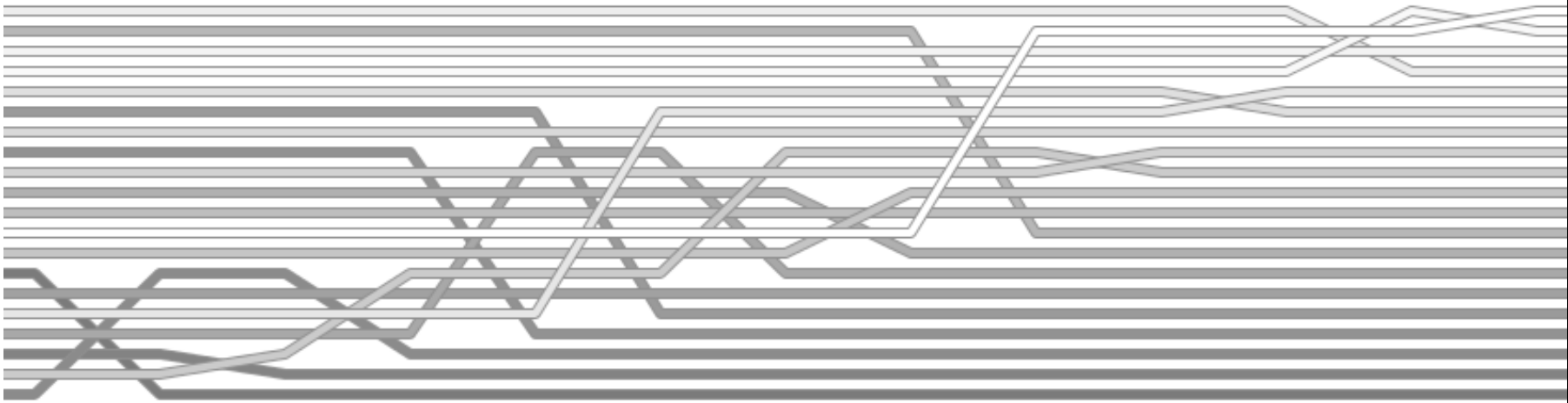
- Running time depends on the number of possible permutations of the list, so it's n factorial
- However, it's non-deterministic, so you might get unlucky and it never terminates

```
while (the list is not sorted)
{
    Randomize the list.
}
```

Bubble



Selection



Insertion

