APS105
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

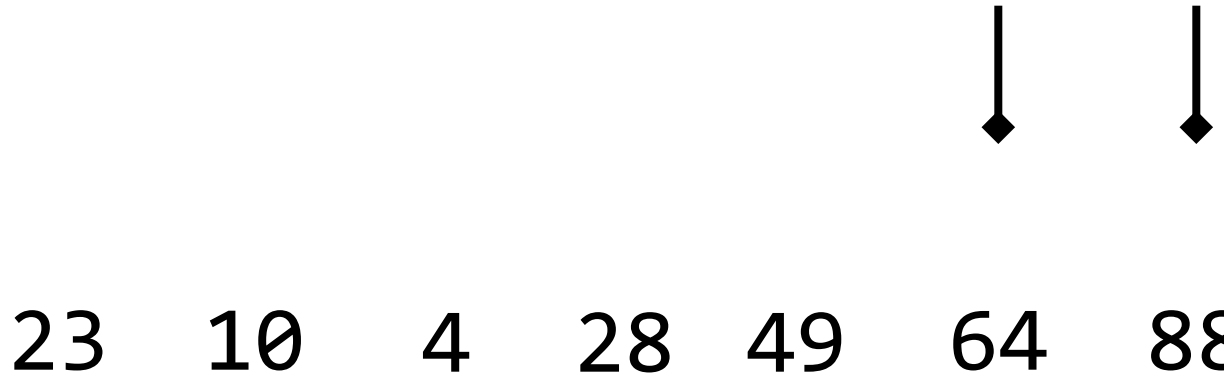
23 10 49 4 28 88 64 37

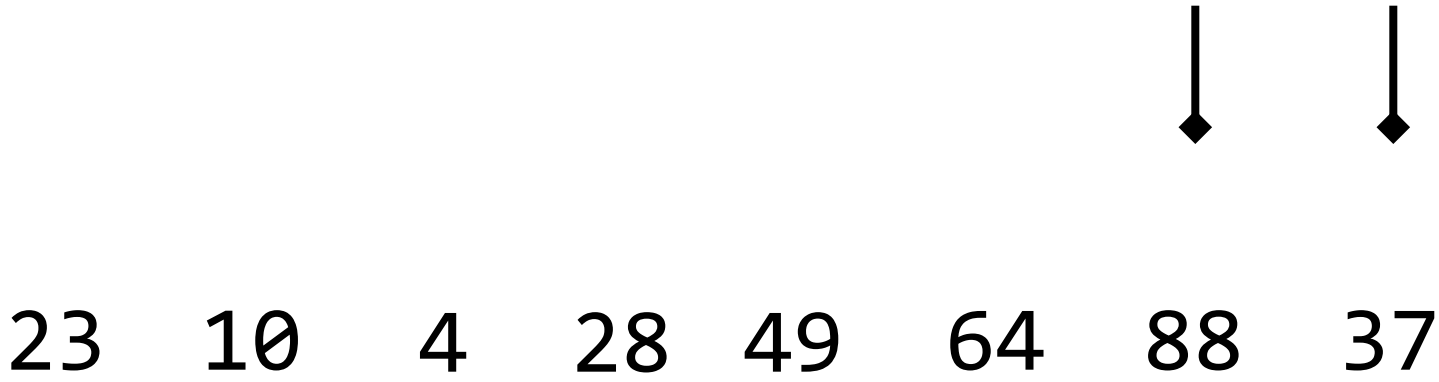
23 10 4 49 28 88 64 37

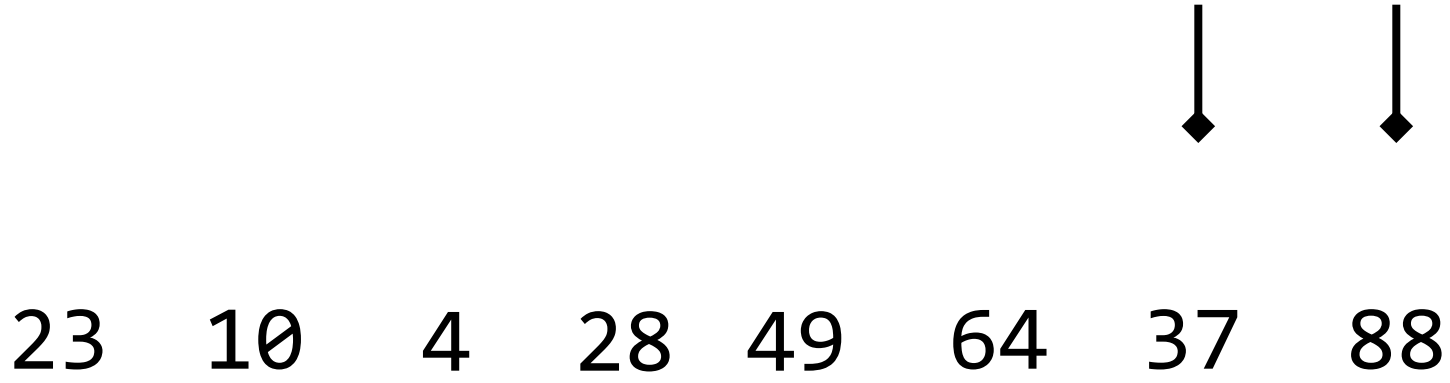
23 10 4 49 28 88 64 37

23 10 4 28 49 88 64 37

23 10 4 28 49 88 64 37


23 10 4 28 49 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

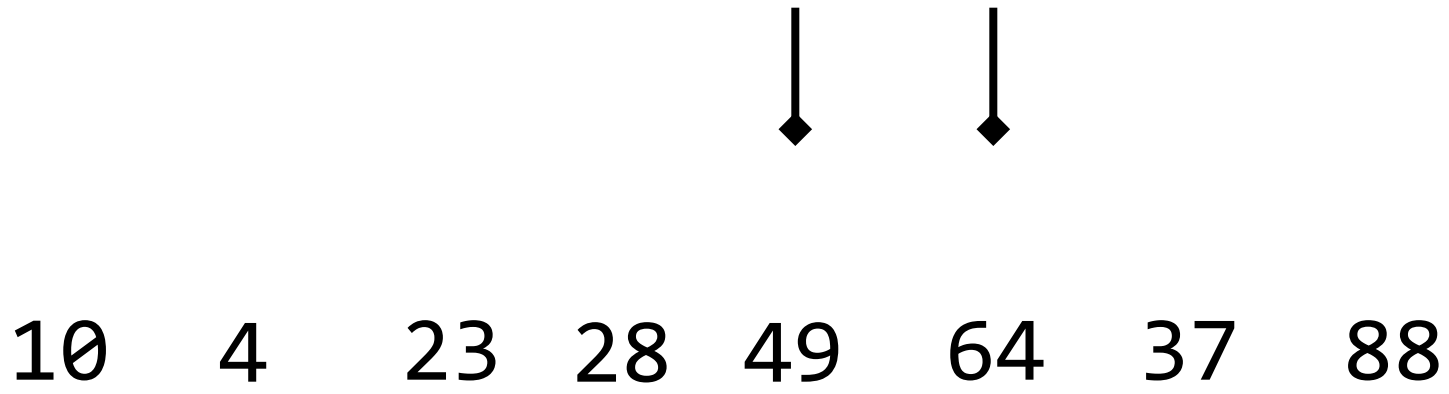
10 23 4 28 49 64 37 88

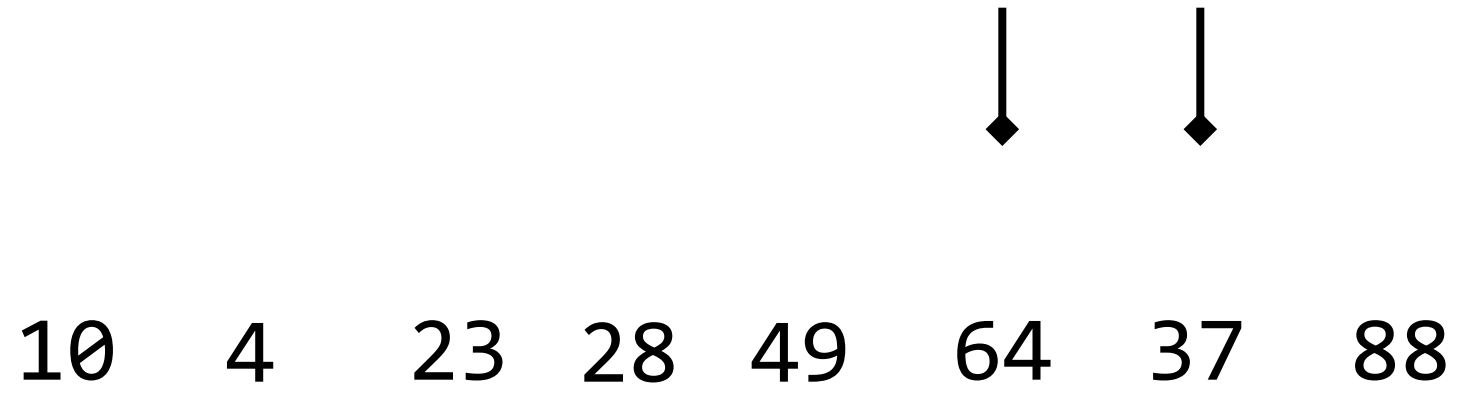
10 23 4 28 49 64 37 88

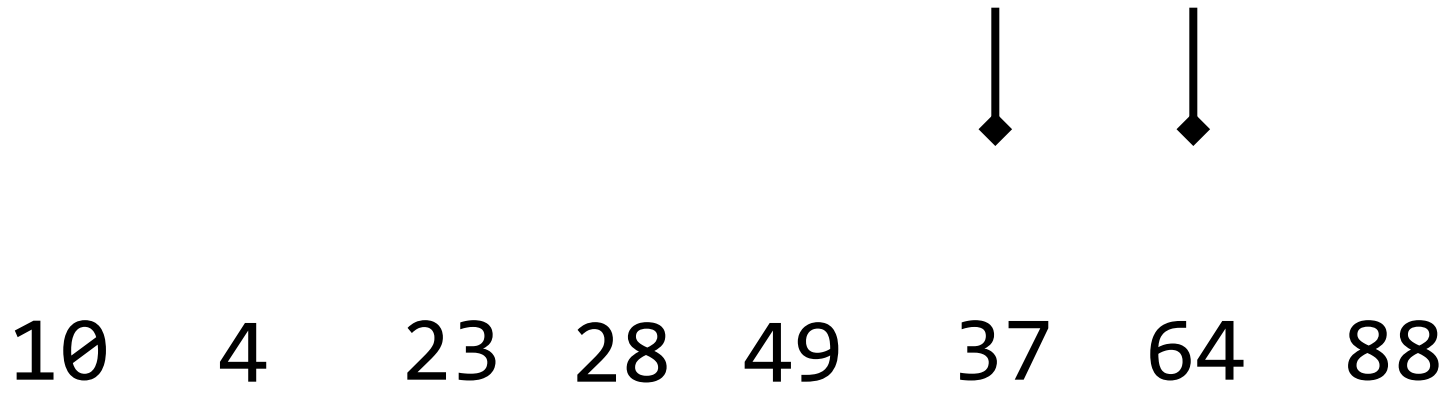
10 4 23 28 49 64 37 88

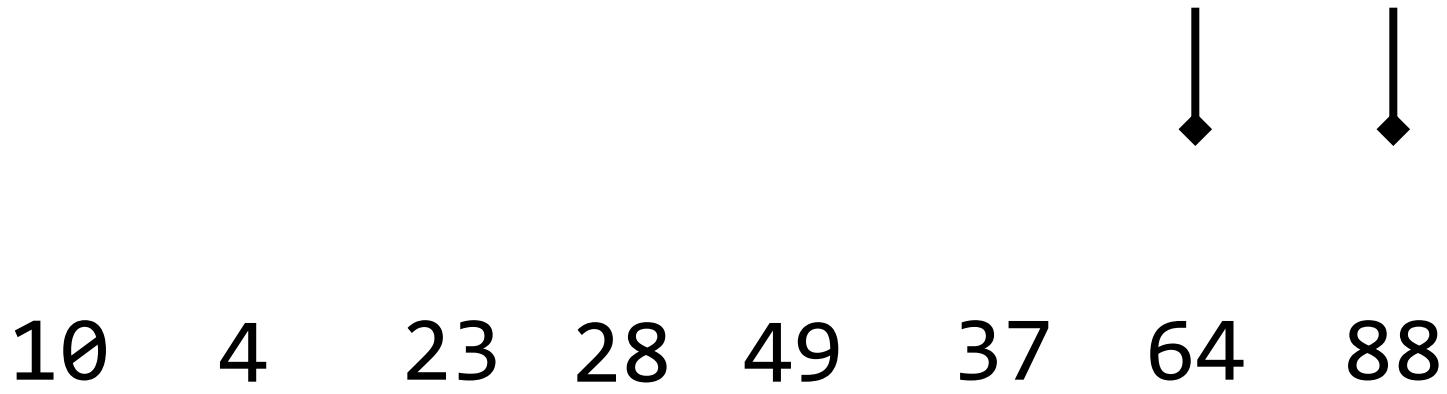
10 4 23 28 49 64 37 88

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

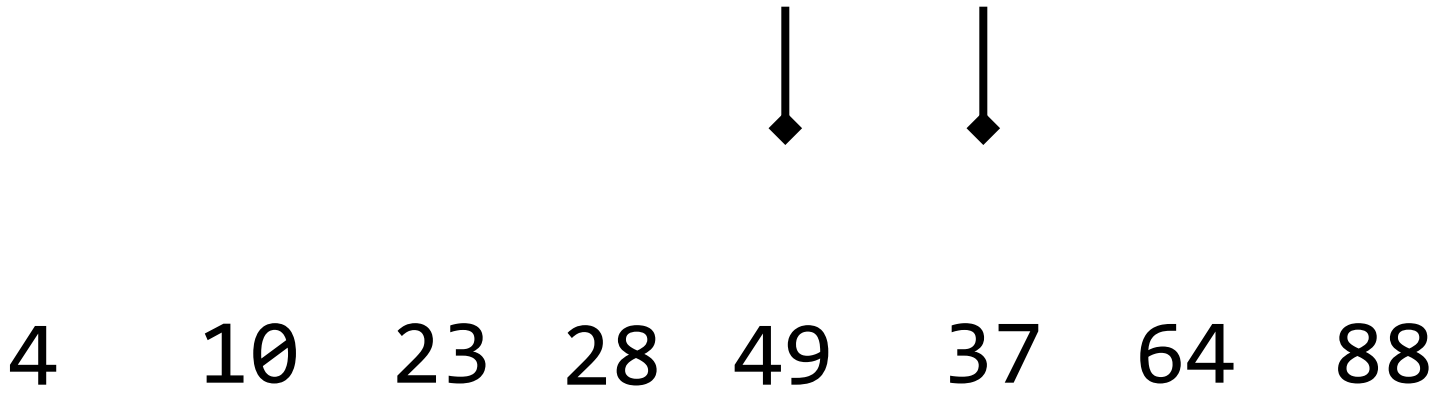
10 4 23 28 49 37 64 88

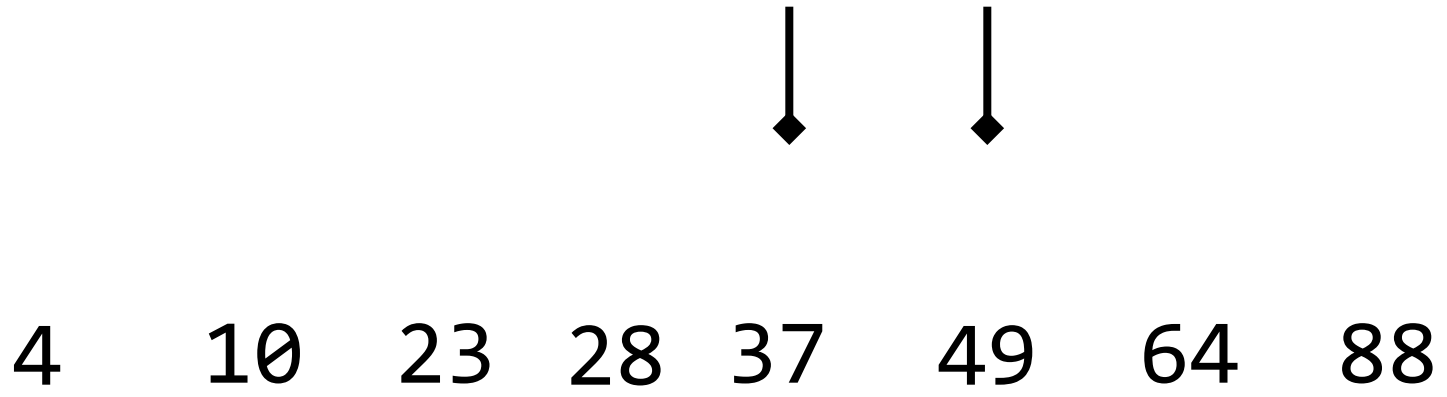
4 10 23 28 49 37 64 88

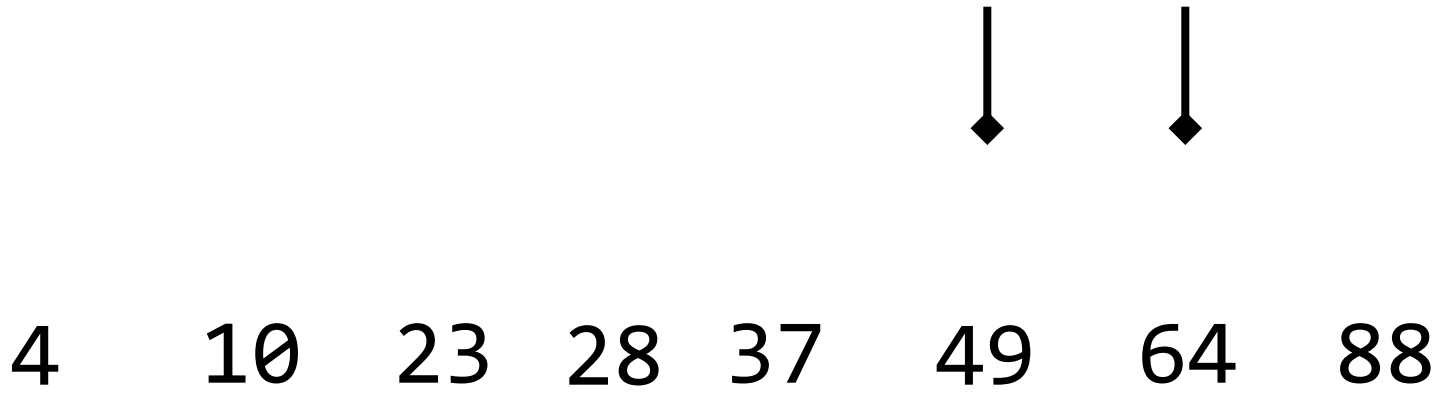
4 10 23 28 49 37 64 88

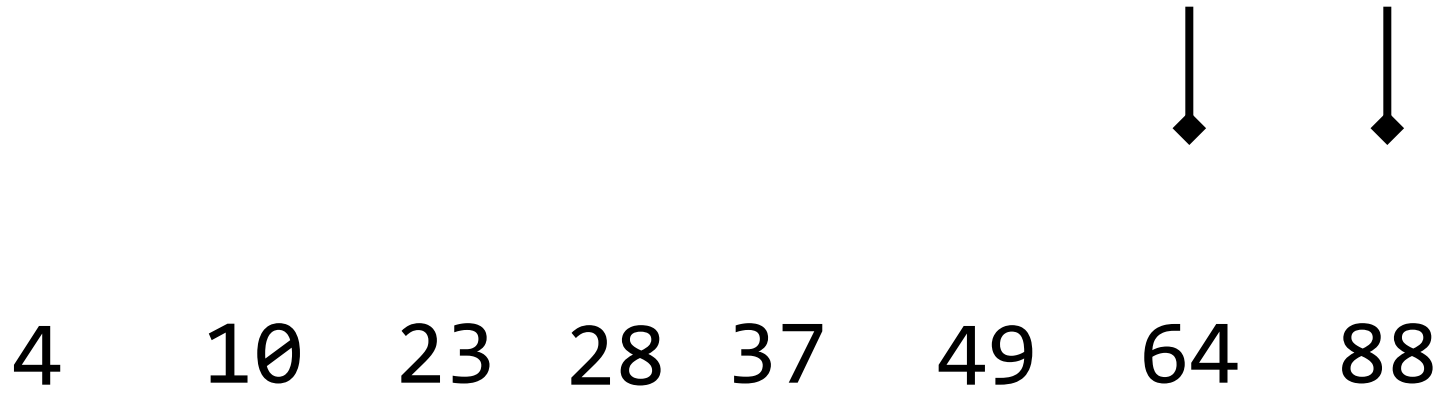
4 10 23 28 49 37 64 88

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

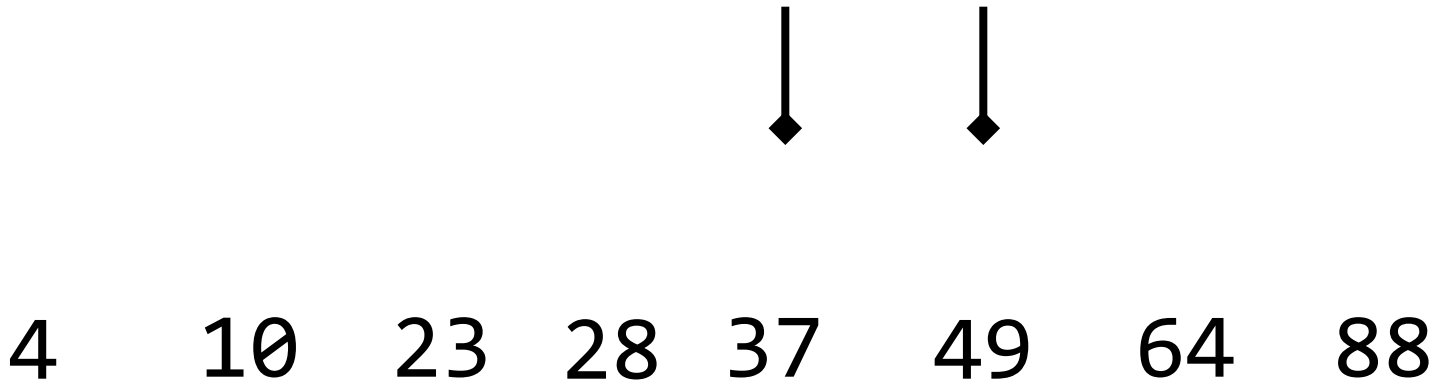
4 10 23 28 37 49 64 88

4 10 23 28 37 49 64 88

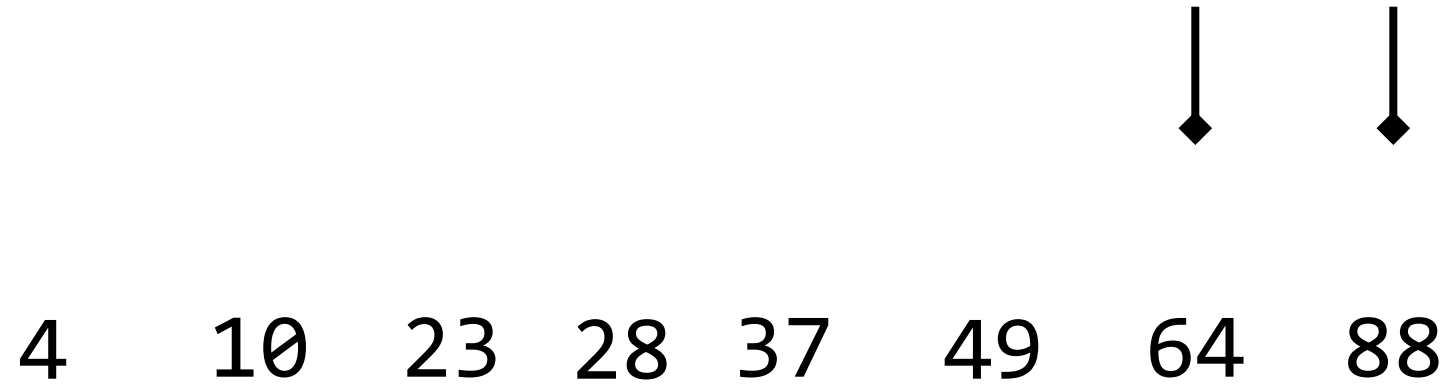
4 10 23 28 37 49 64 88

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

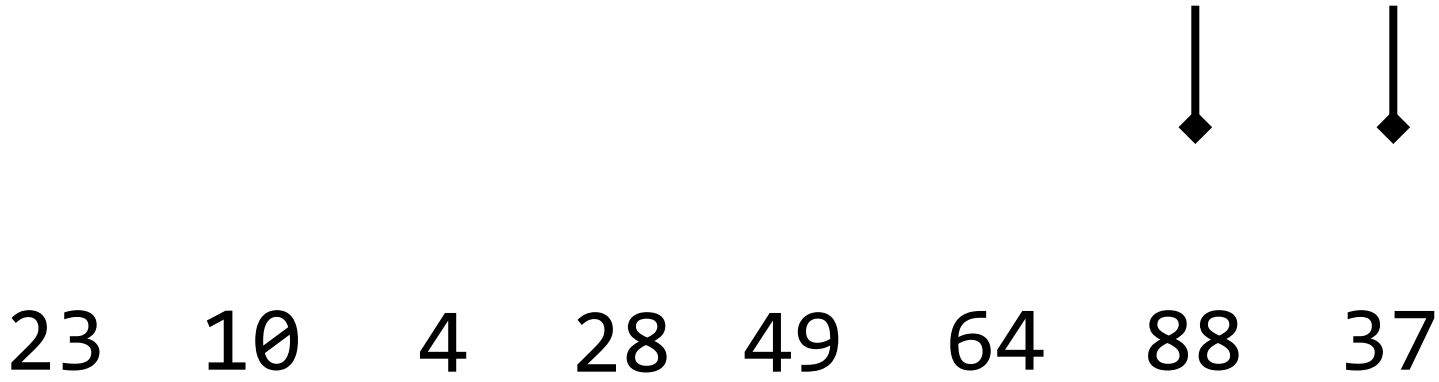
23 10 4 49 28 88 64 37

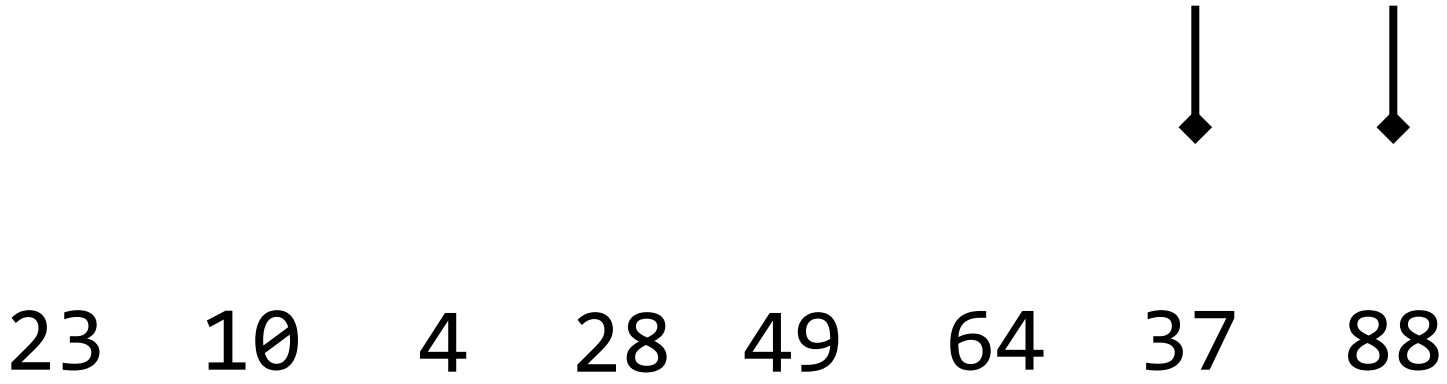
23 10 4 28 49 88 64 37

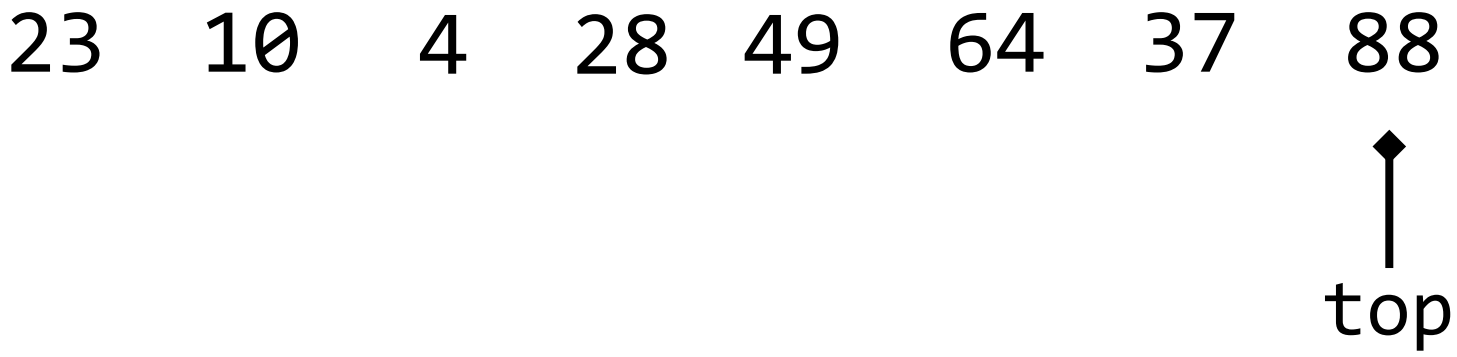
23 10 4 28 49 88 64 37

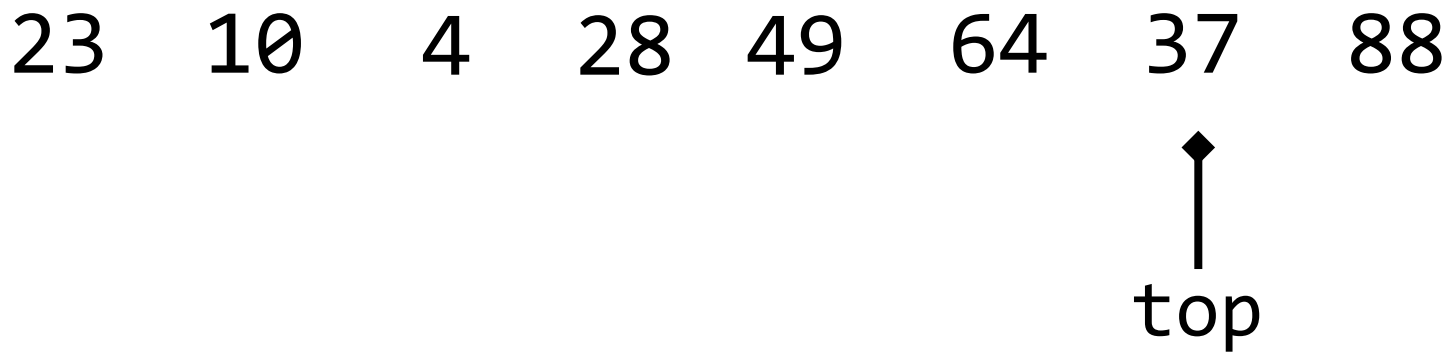
23 10 4 28 49 88 64 37

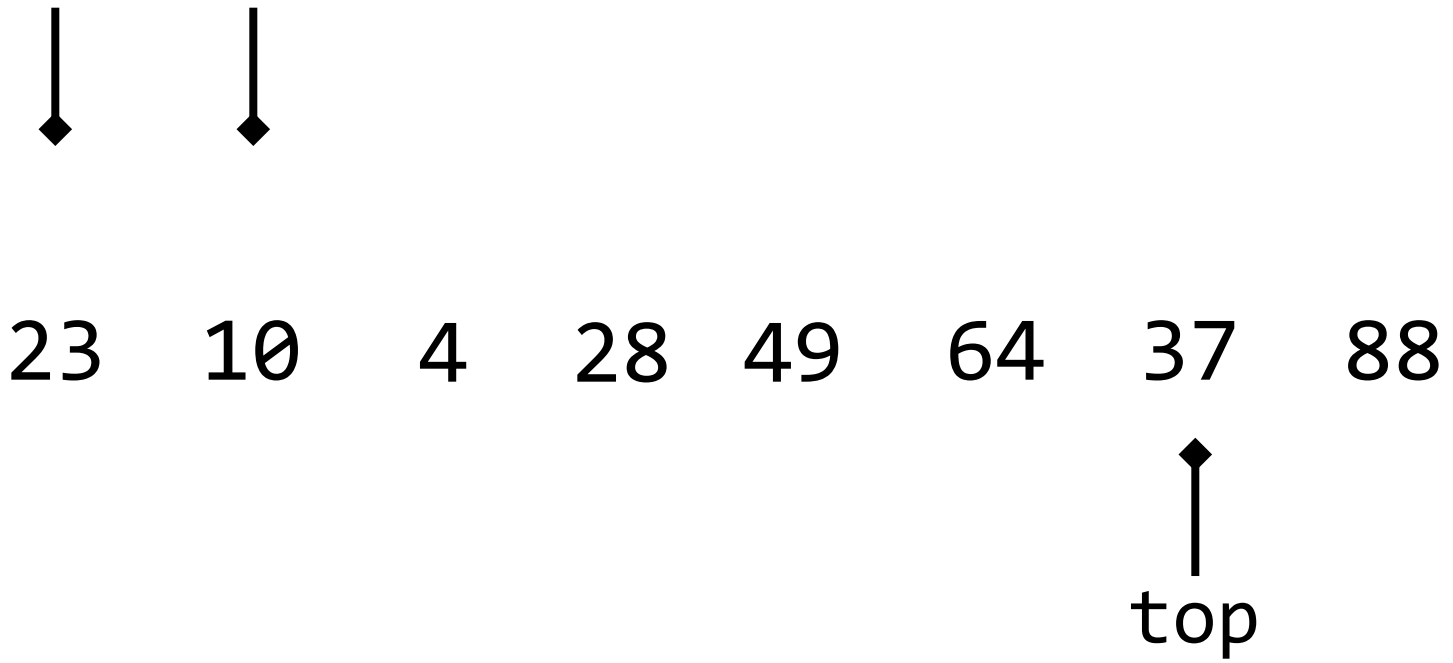
23 10 4 28 49 64 88 37

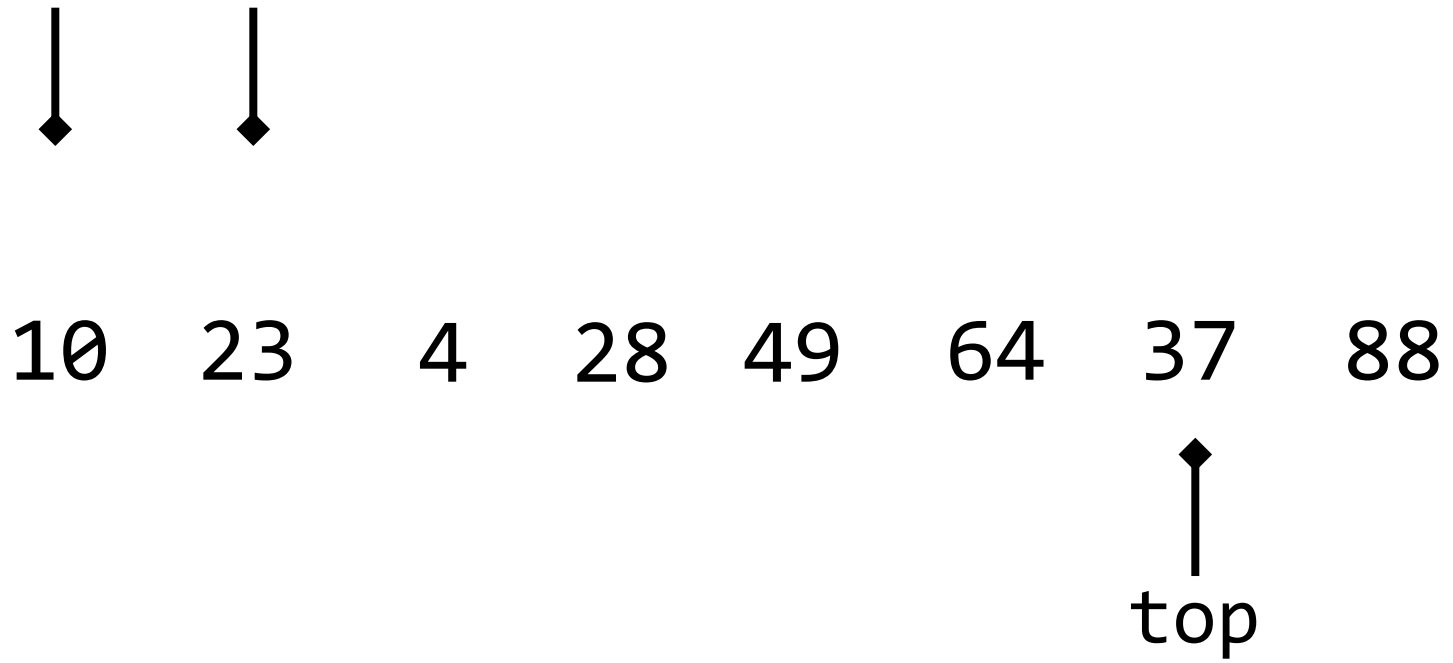

23 10 4 28 49 64 88 37

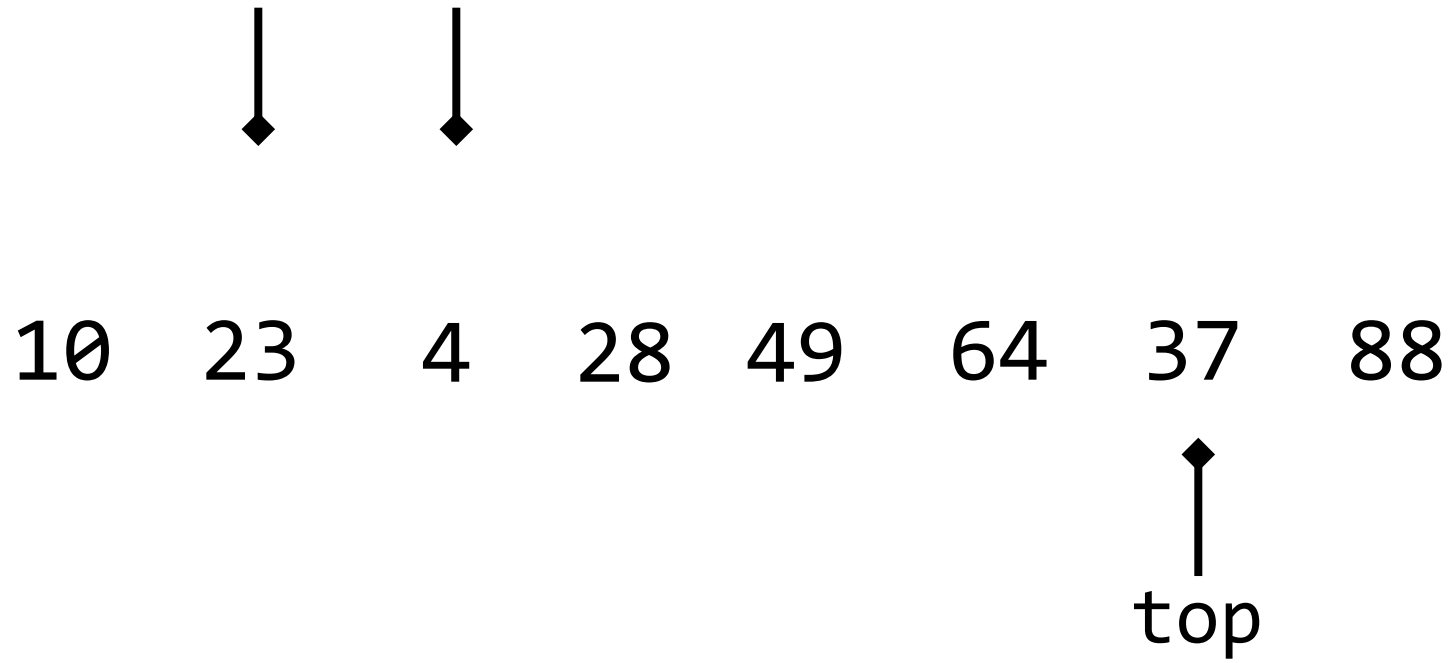

23 10 4 28 49 64 37 88

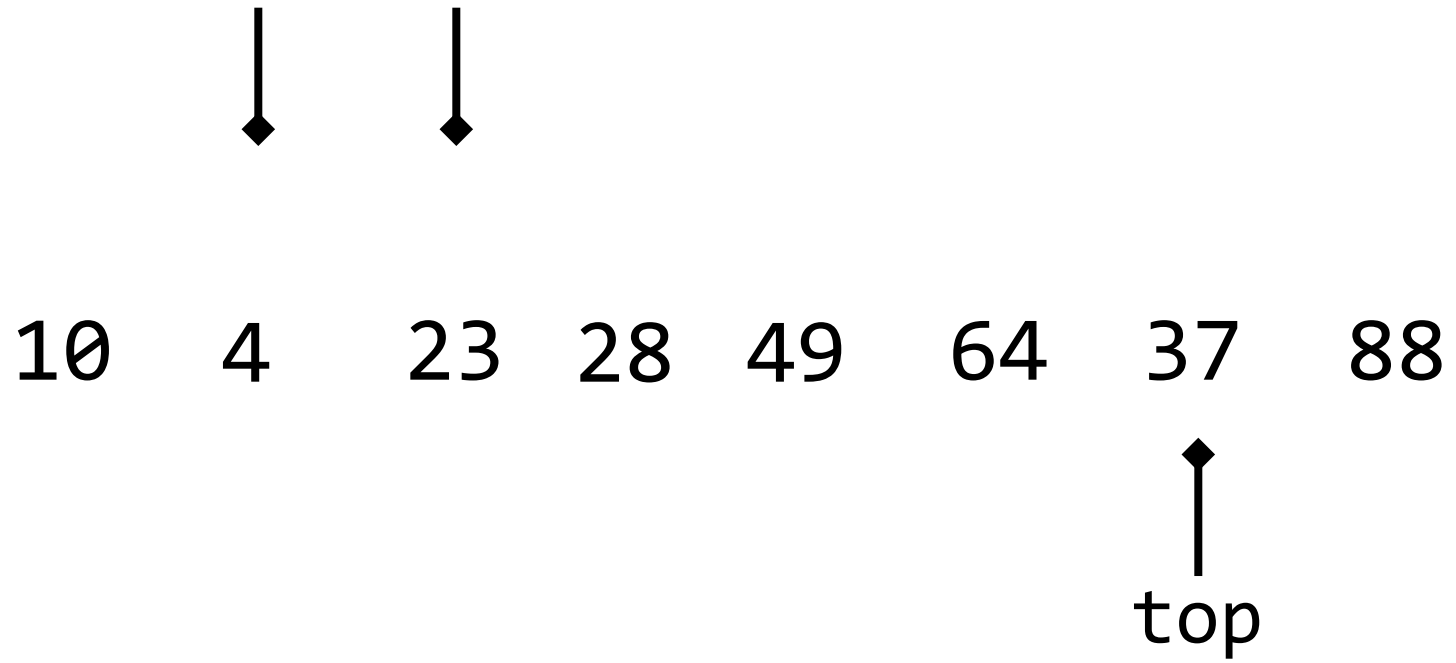

23 10 4 28 49 64 37 88
top

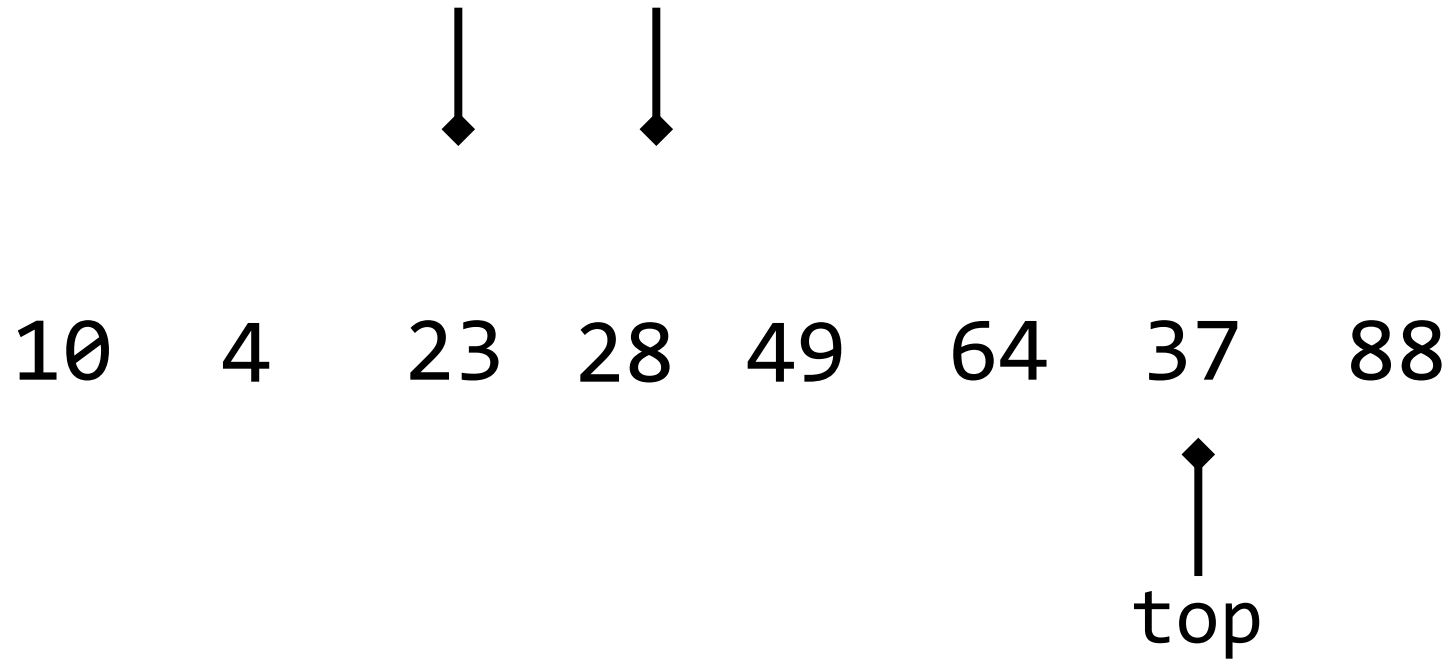

23 10 4 28 49 64 37 88
top

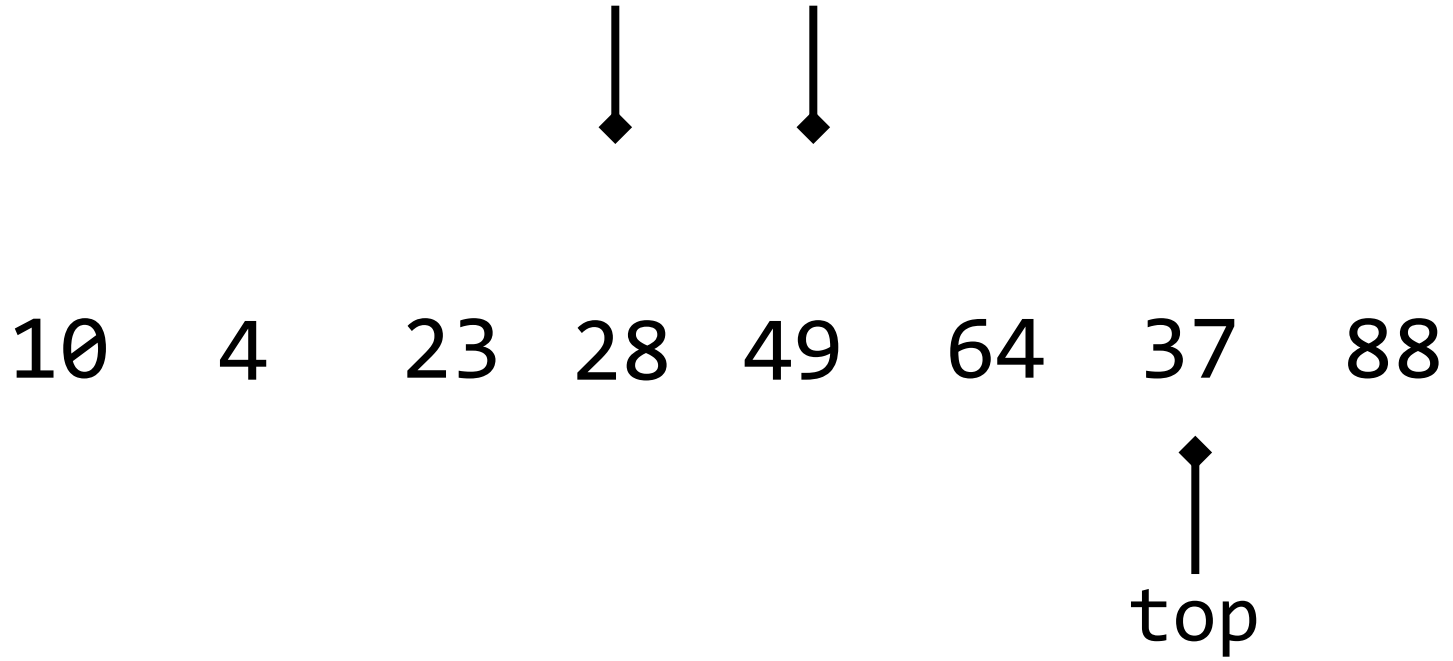

23 10 4 28 49 64 37 88
top

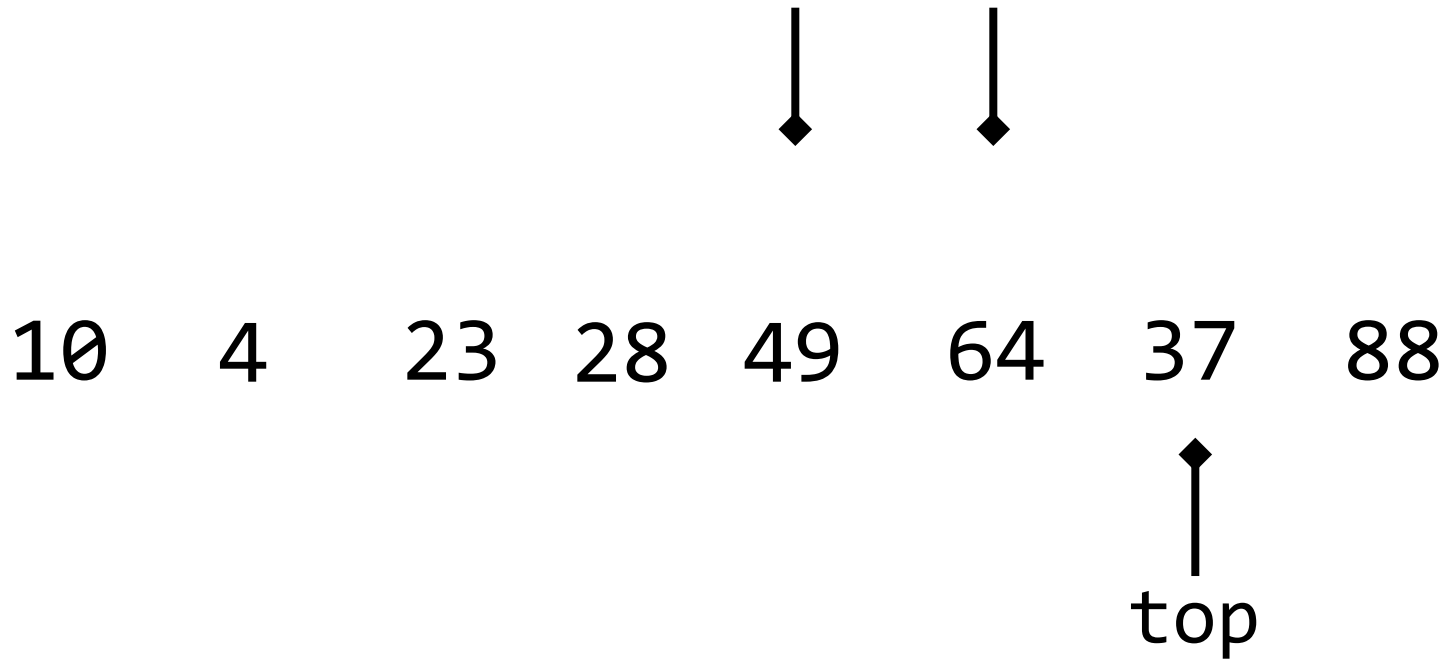

10 23 4 28 49 64 37 88
top

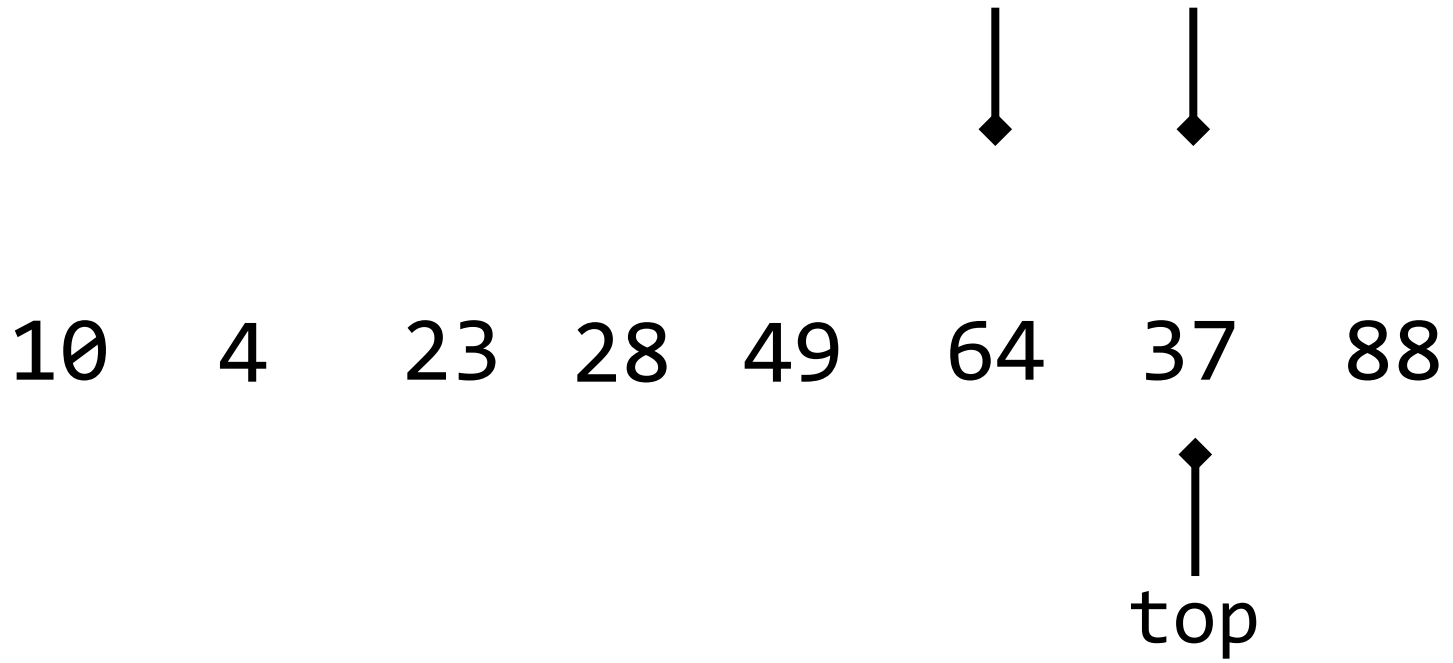

10 23 4 28 49 64 37 88
top

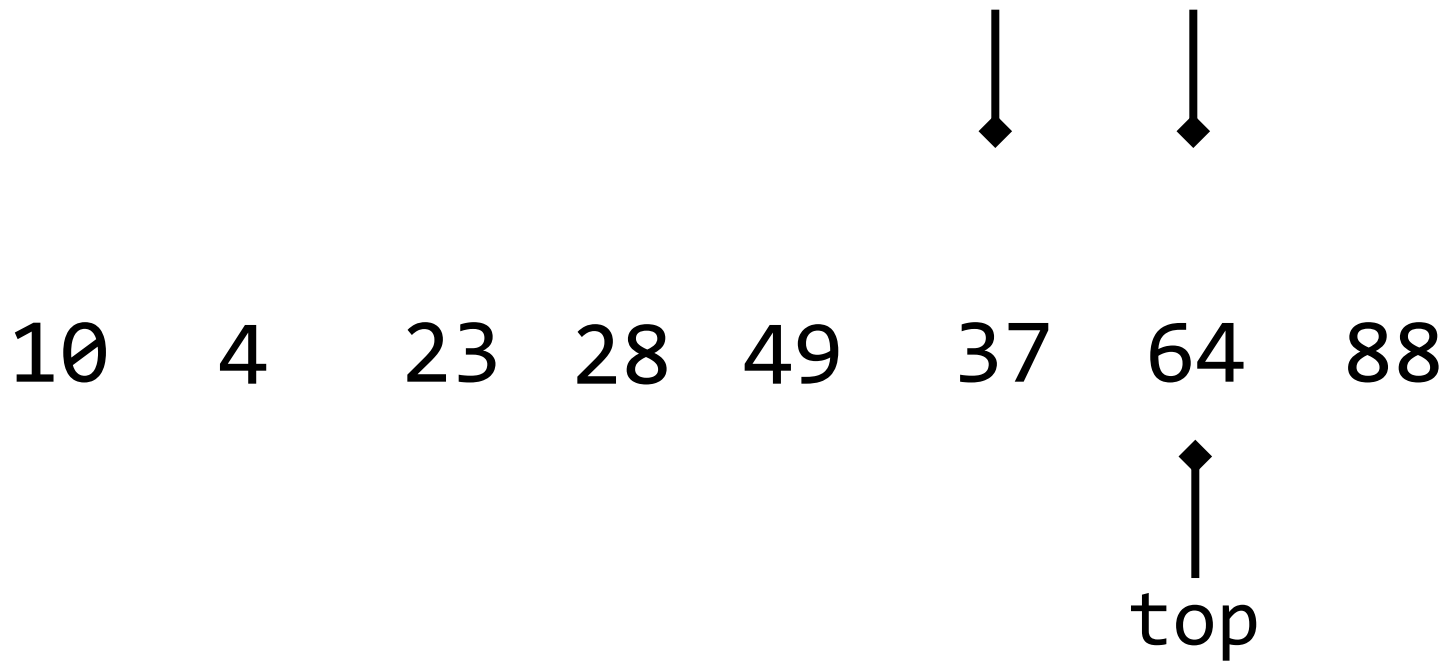

10 4 23 28 49 64 37 88
top

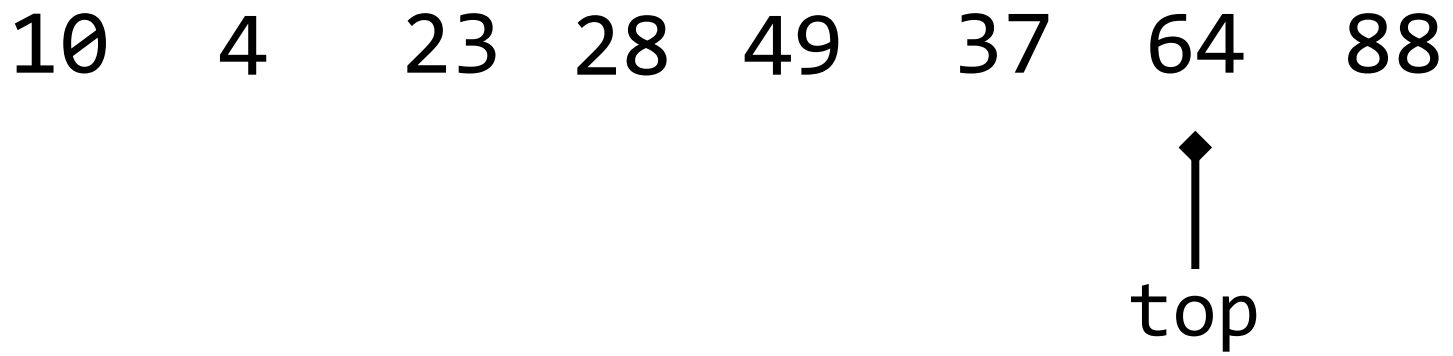

10 4 23 28 49 64 37 88
top

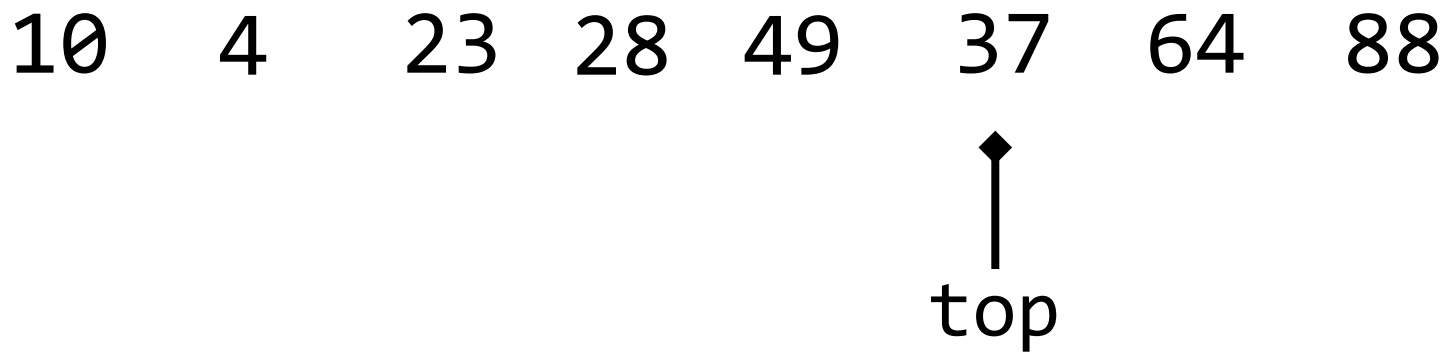

10 4 23 28 49 64 37 88
top

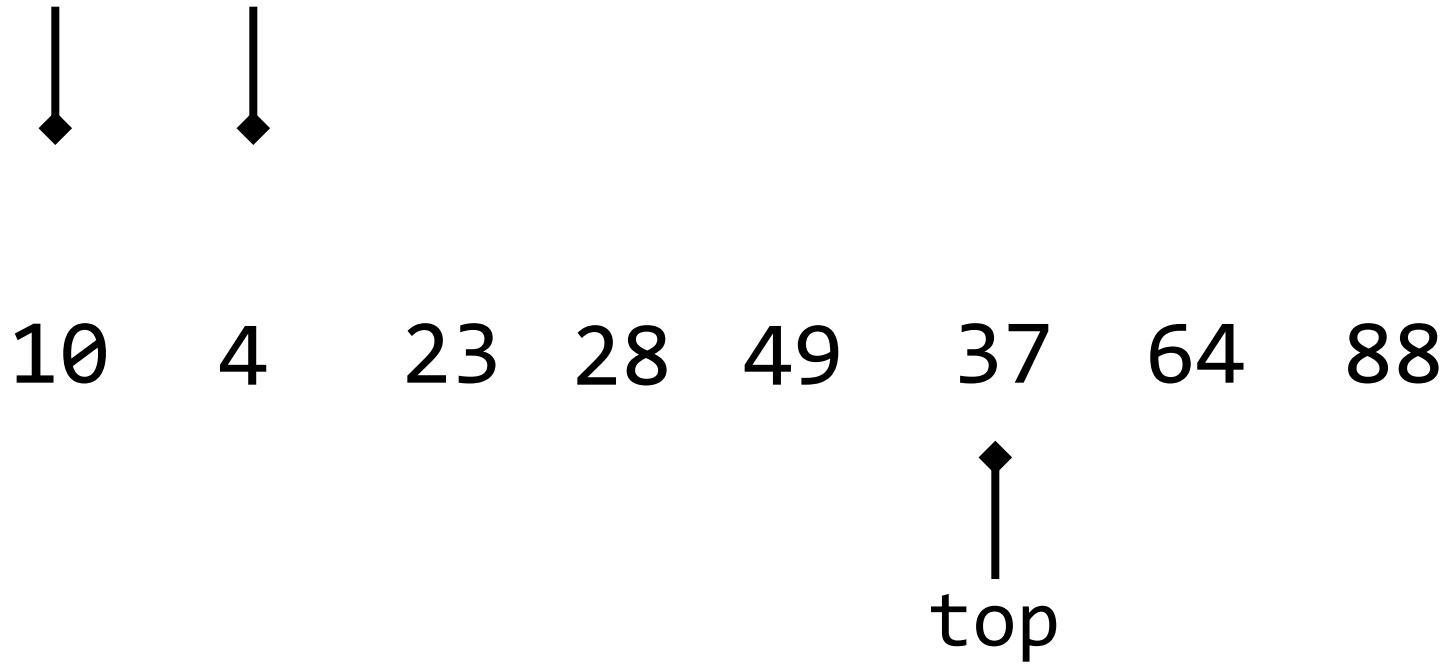

10
4
23
28
49
64
37
88
top

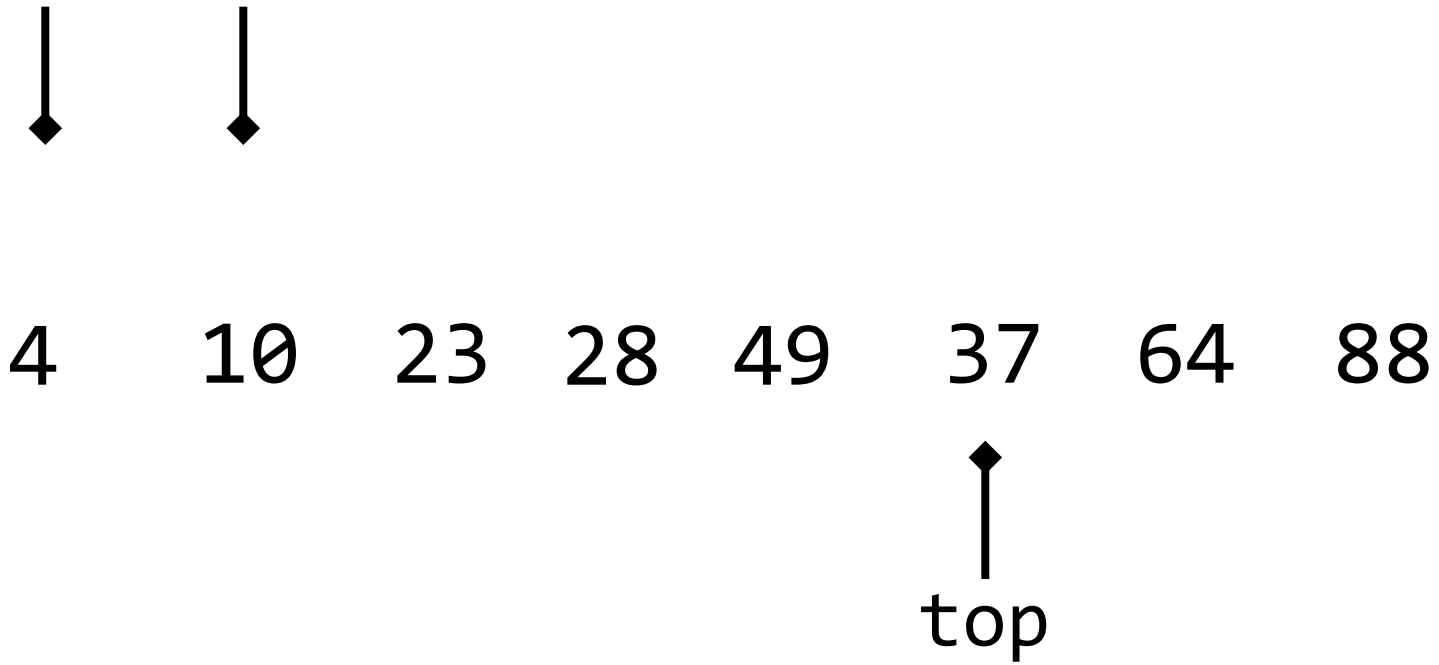

10 4 23 28 49 64 37 88
top

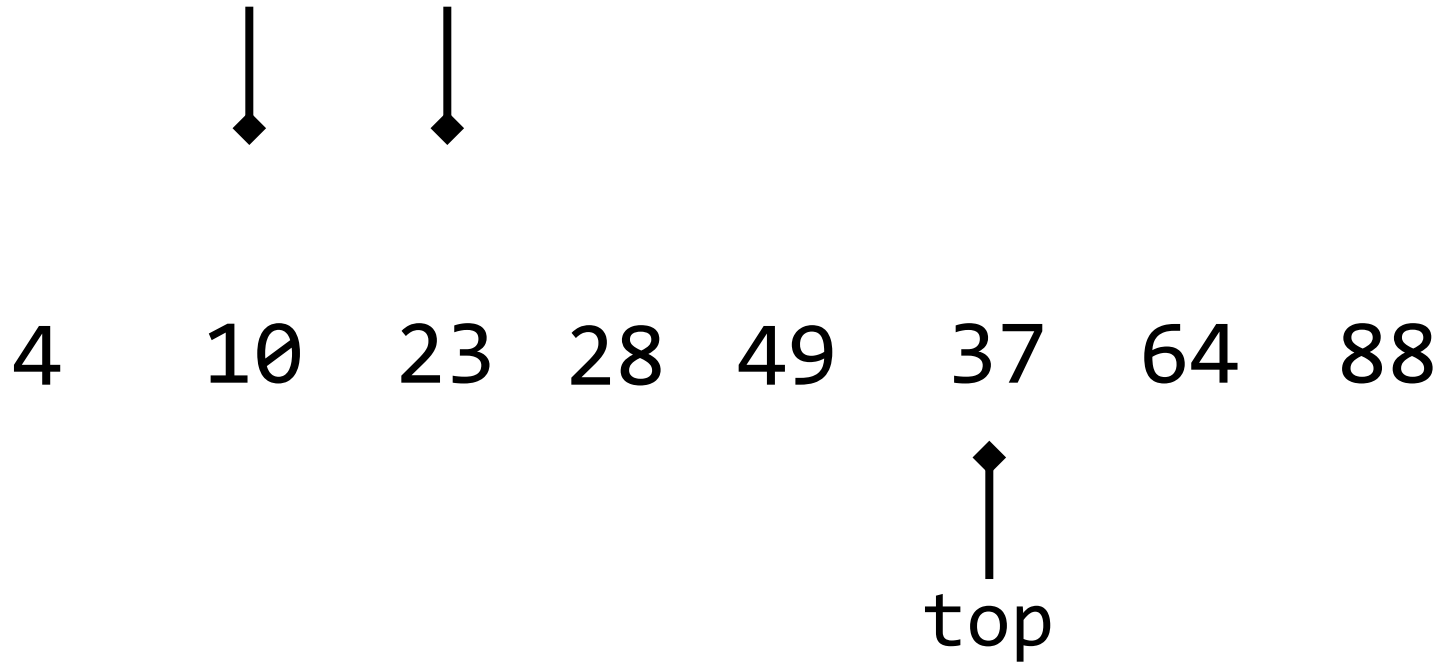

10 4 23 28 49 37 64 88
top

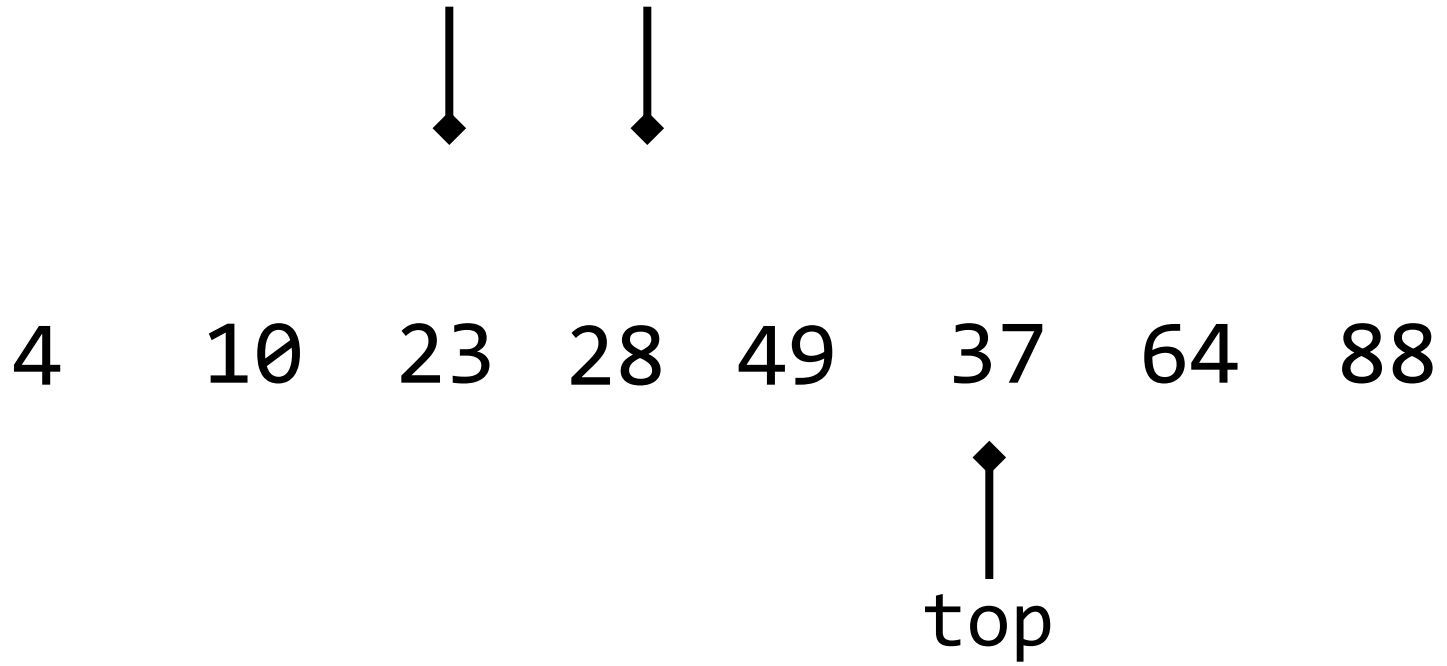

10 4 23 28 49 37 64 88
top

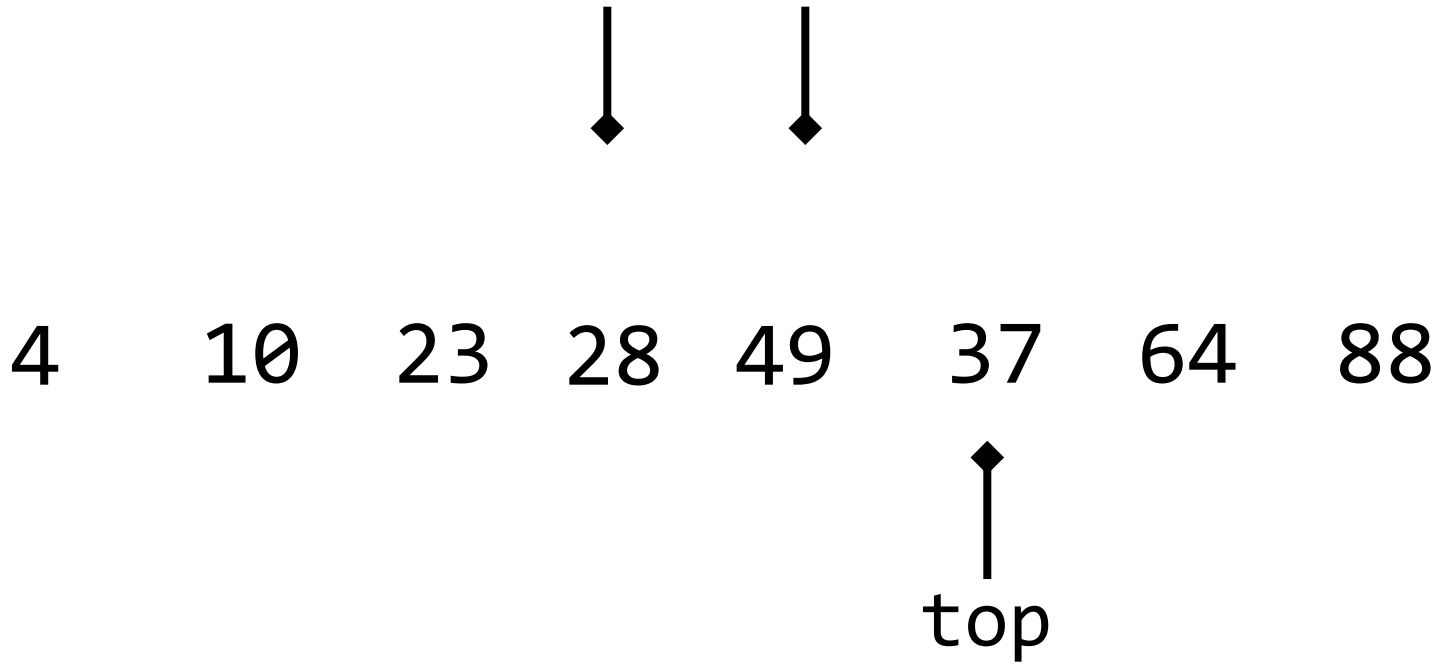

10 4 23 28 49 37 64 88
top

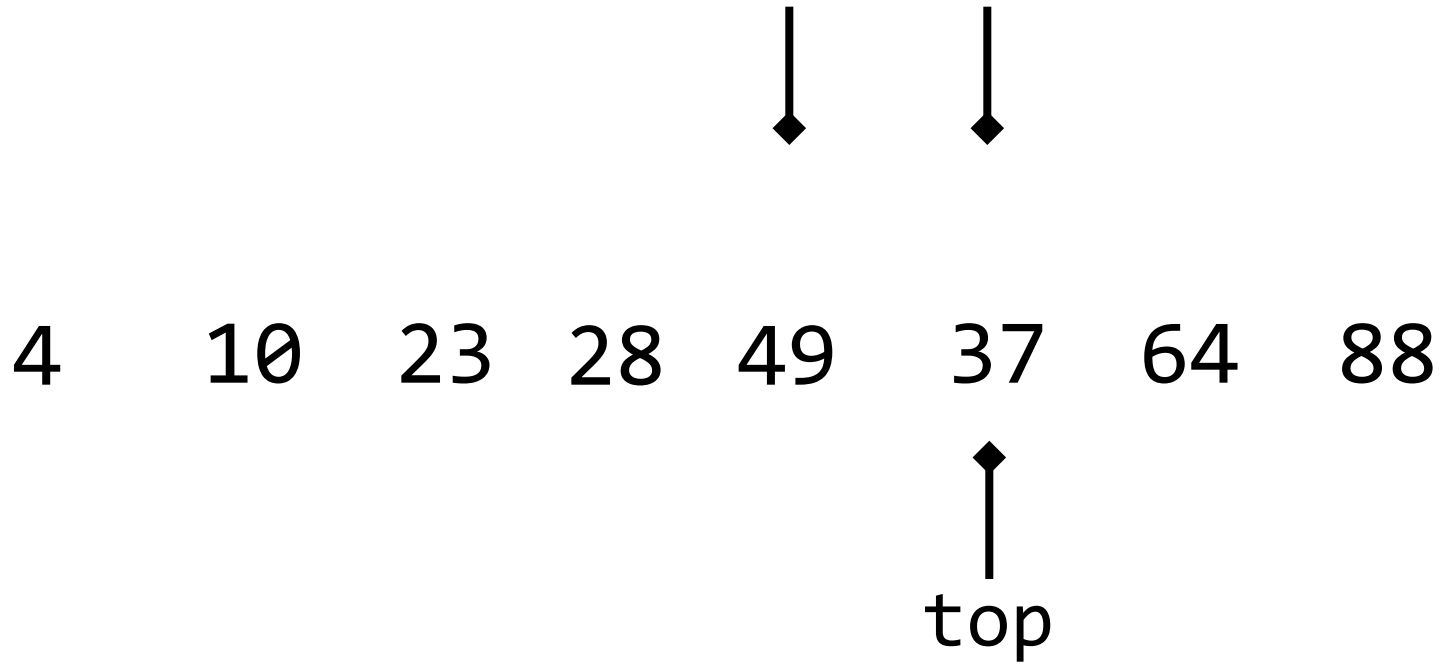

10
4
23
28
49
37
64
88
top

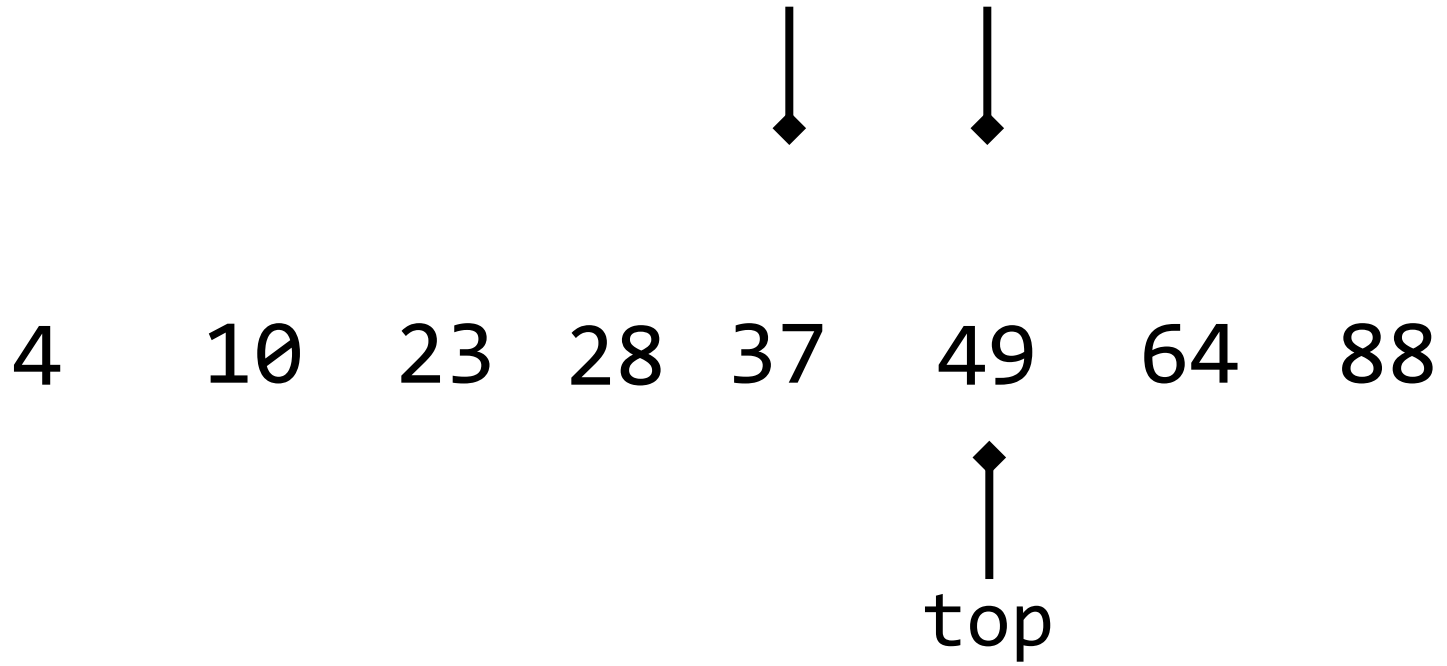

4
10
23
28
49
37
64
88
top

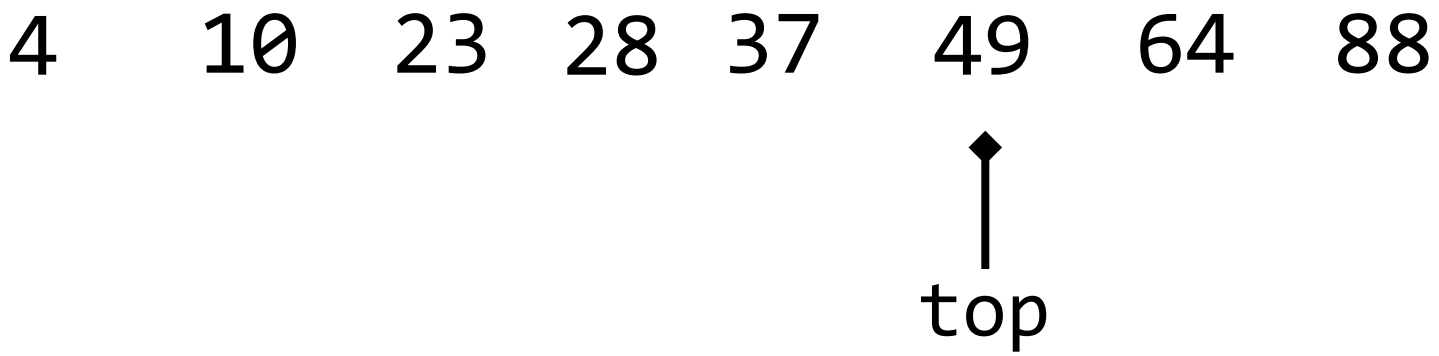

4
10
23
28
49
37
64
88
top

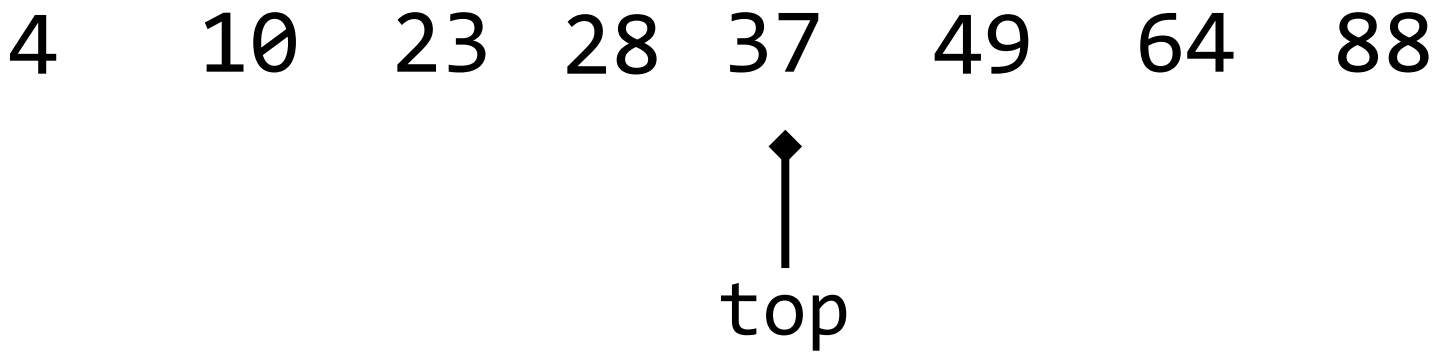

4
10
23
28
49
37
64
88
top

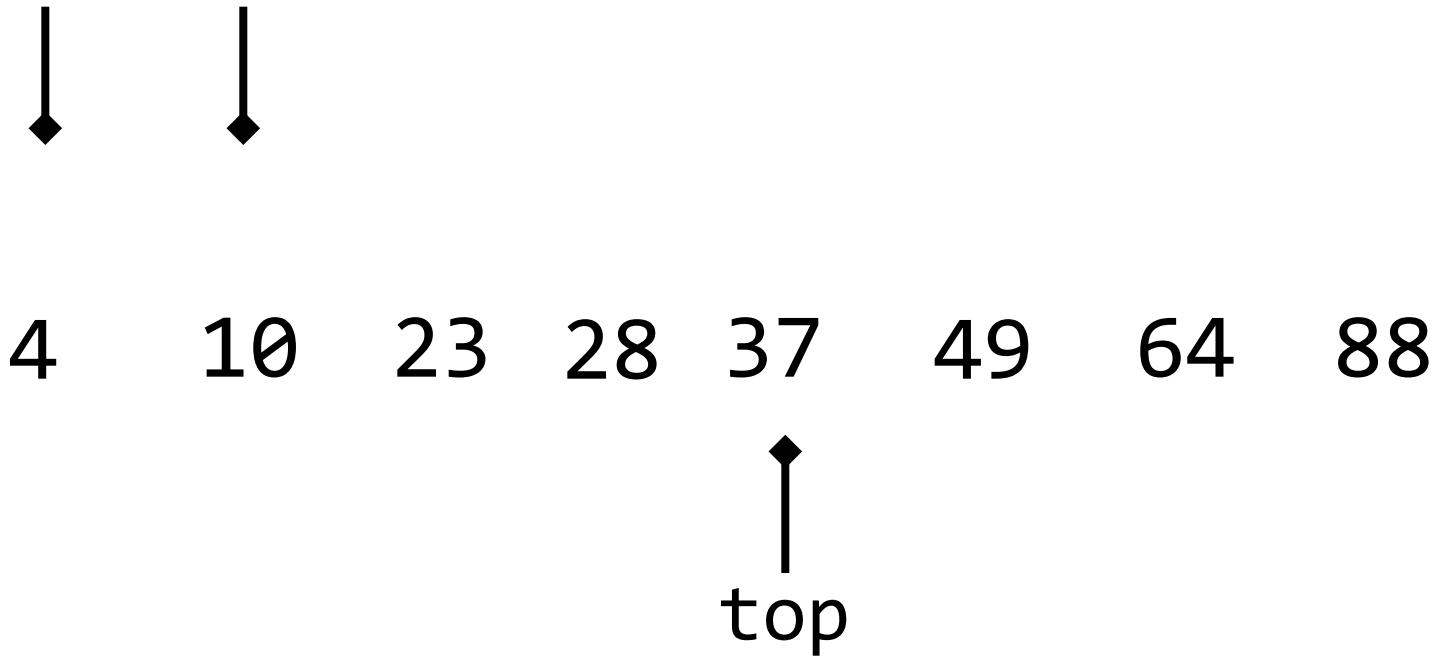

4
10
23
28
49
37
64
88
top

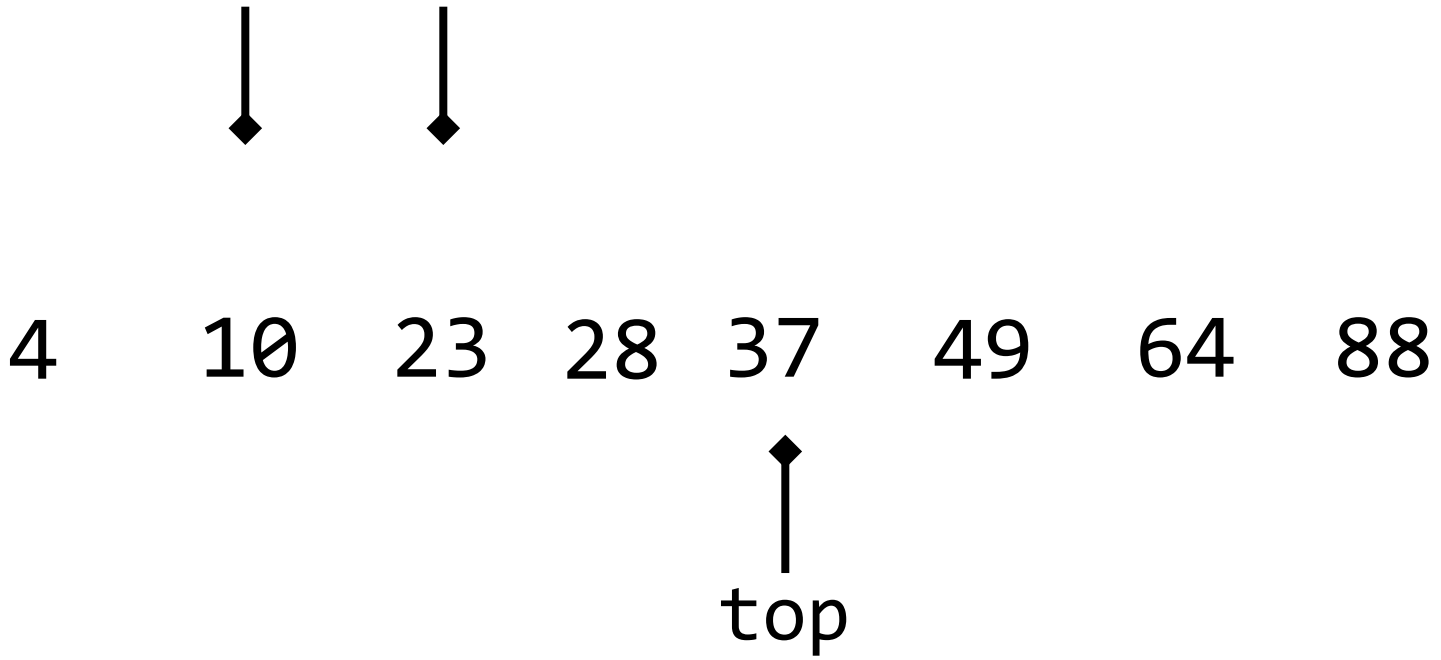

4
10
23
28
49
37
64
88
top

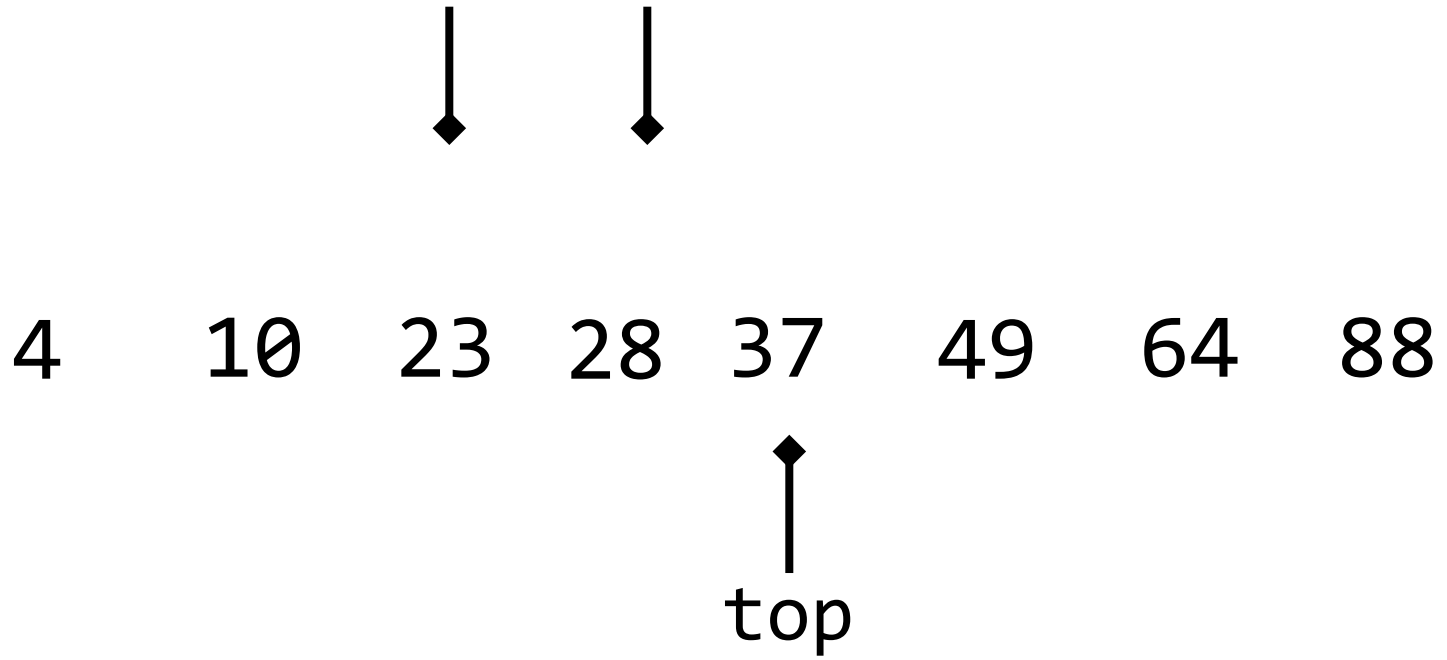

4
10
23
28
37
49
64
88
top

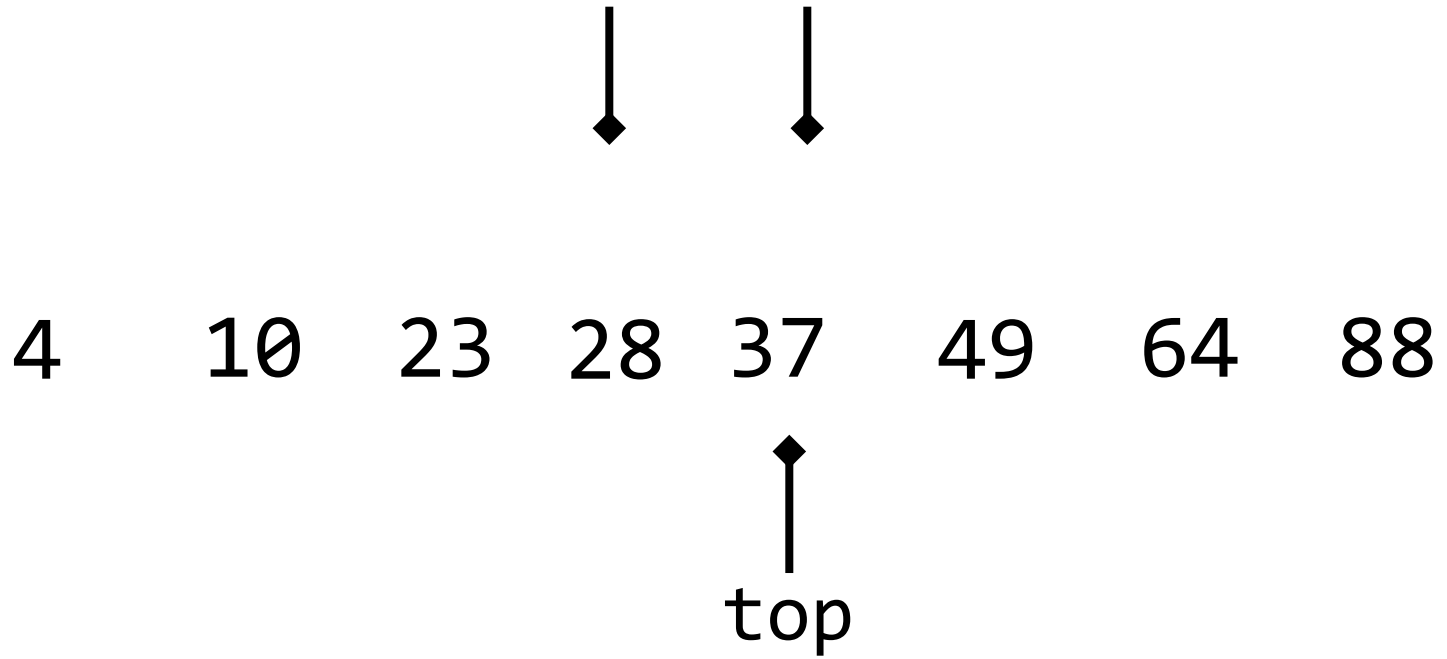

4 10 23 28 37 49 64 88
top

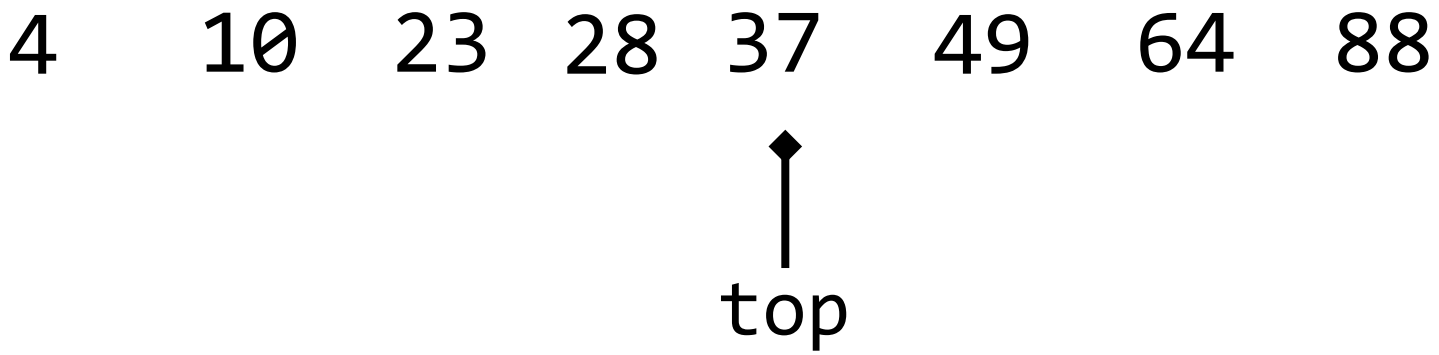

4 10 23 28 37 49 64 88
top


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


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

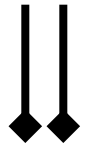
Selection Sort

- 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

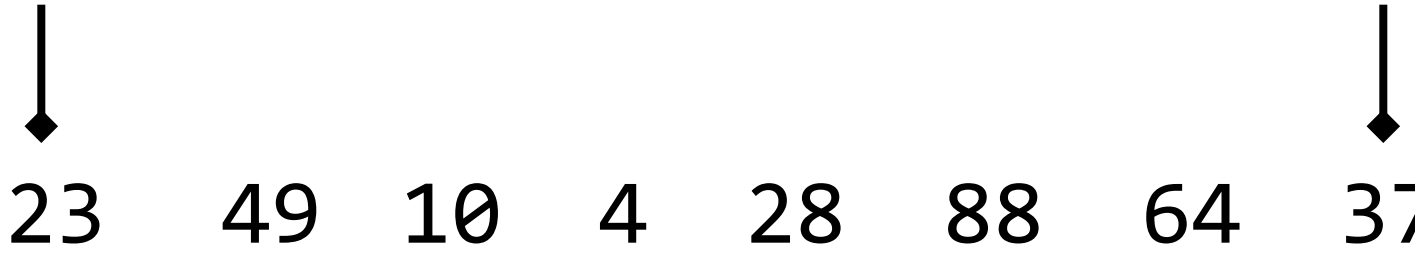
23 49 10 4 28 88 64 37

Smallest: 4

Smallest: 4

Smallest: 4

Smallest: 4



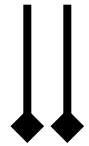
Smallest: 4

4 49 10 23 28 88 64 37

Smallest:

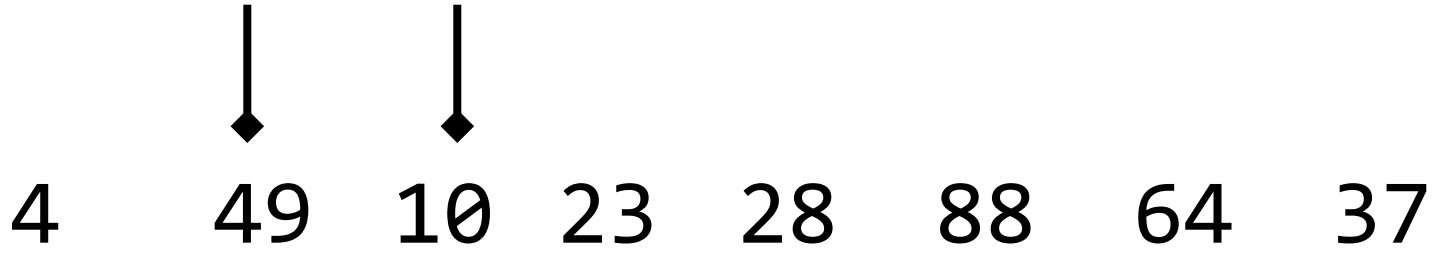
4 49 10 23 28 88 64 37

Smallest: 49

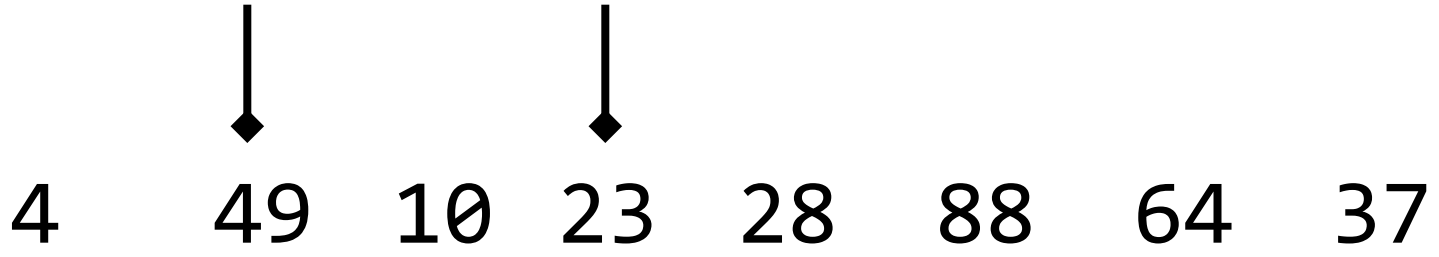


4 49 10 23 28 88 64 37

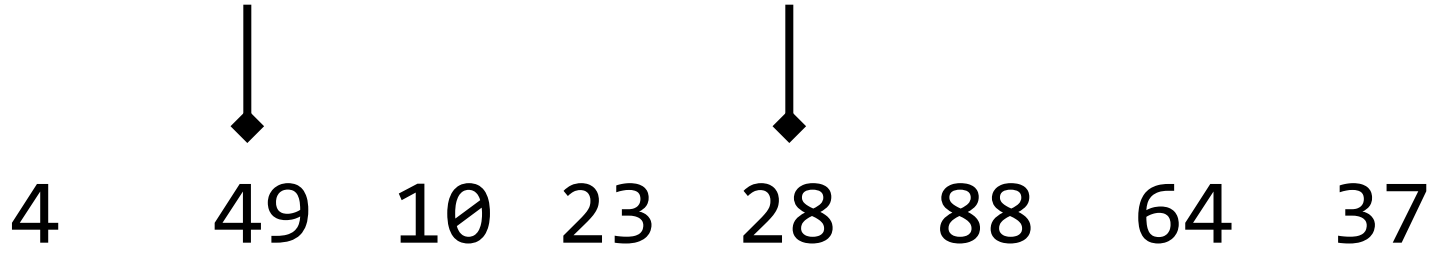
Smallest: 10



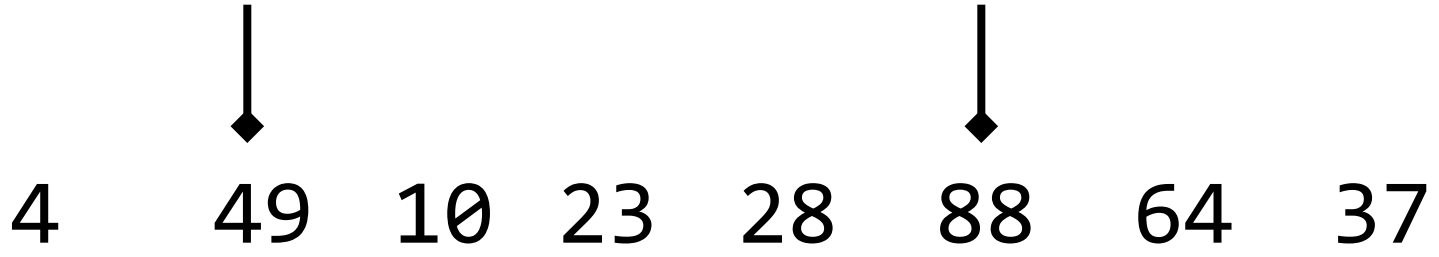
Smallest: 10



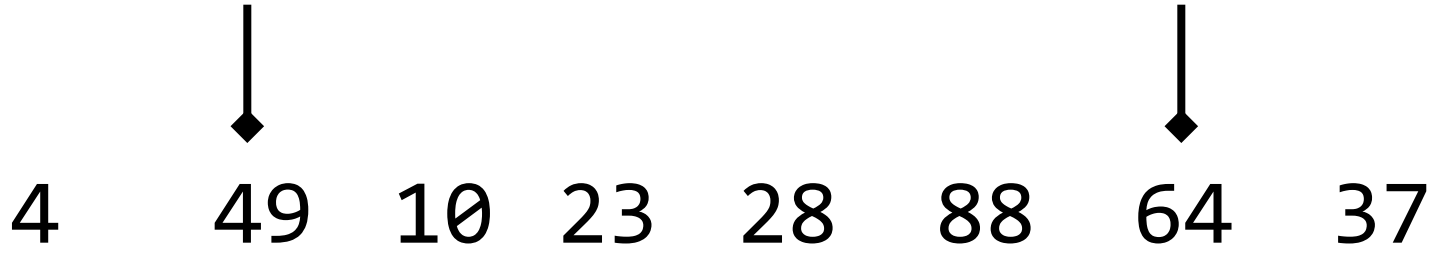
Smallest: 10



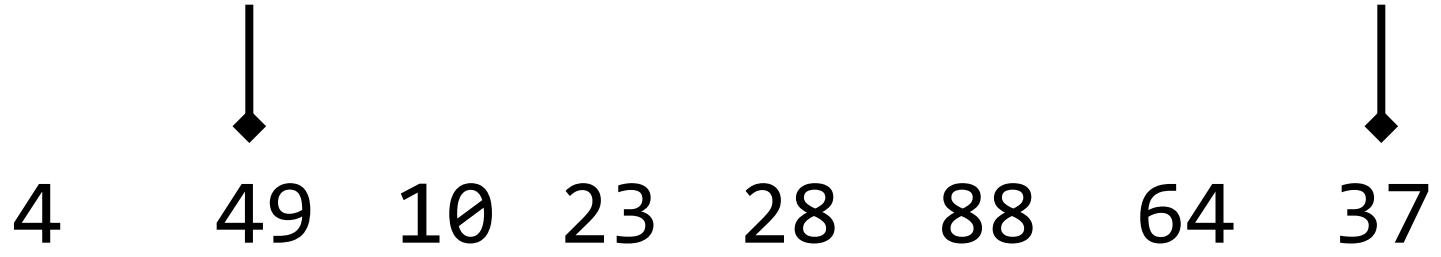
Smallest: 10



Smallest: 10



Smallest: 10



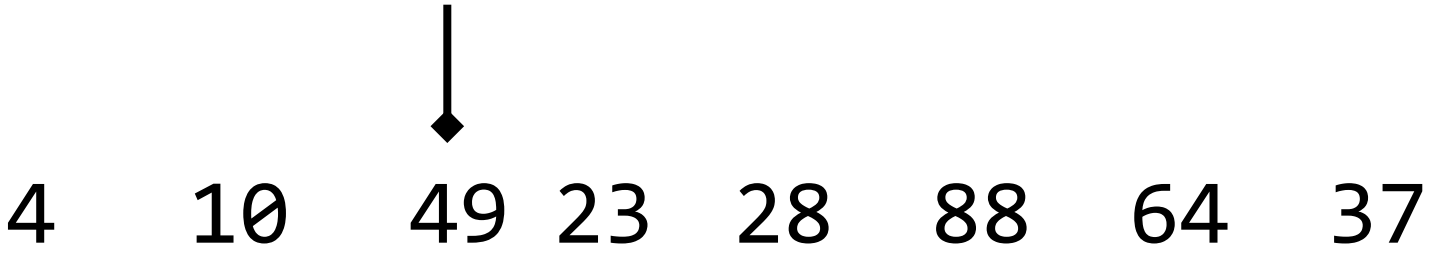
Smallest: 10

4 49 10 23 28 88 64 37

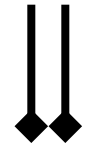
Smallest: 10

4 10 49 23 28 88 64 37

Smallest:

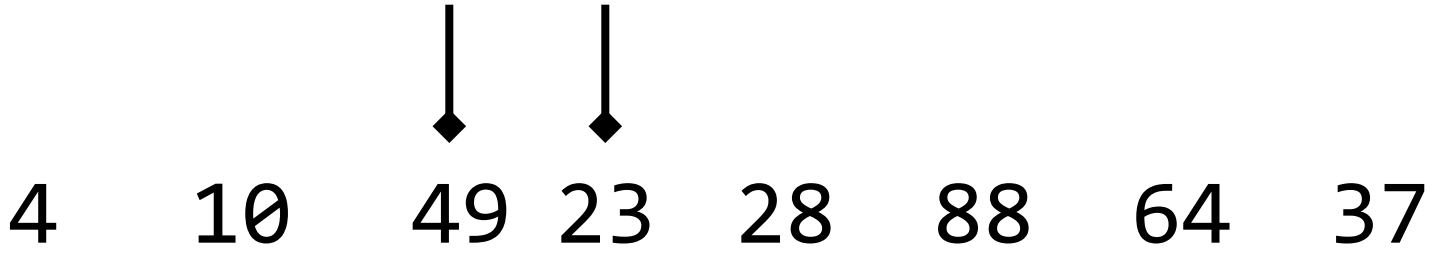


Smallest: 49

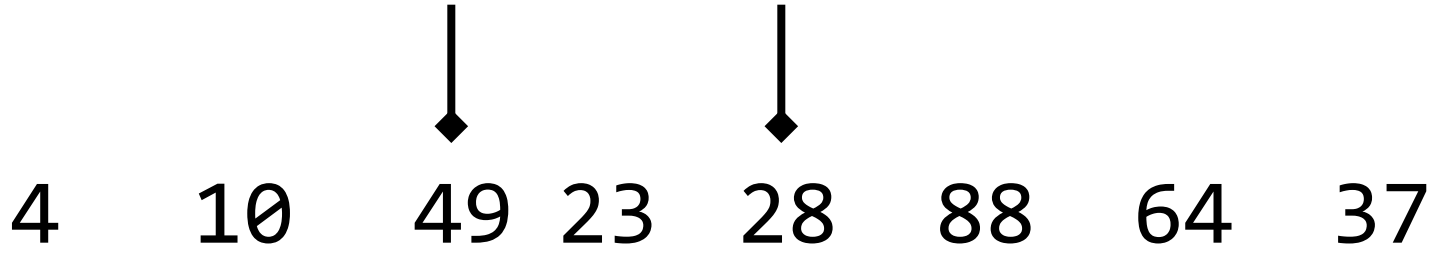


4 10 49 23 28 88 64 37

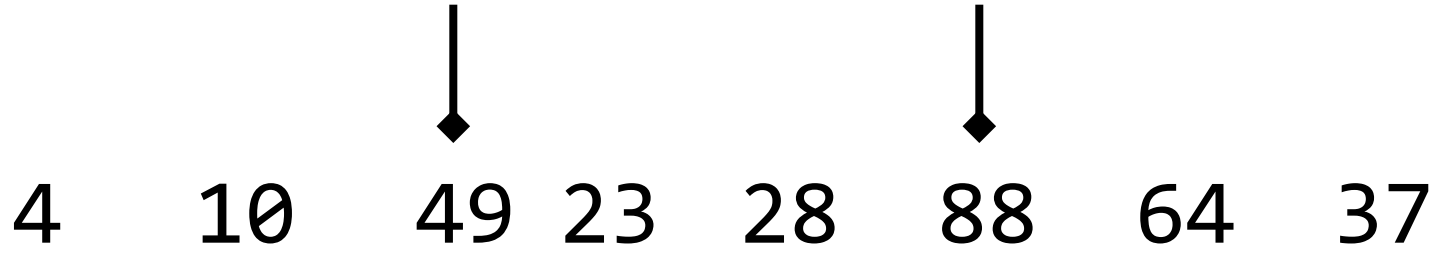
Smallest: 23



Smallest: 23

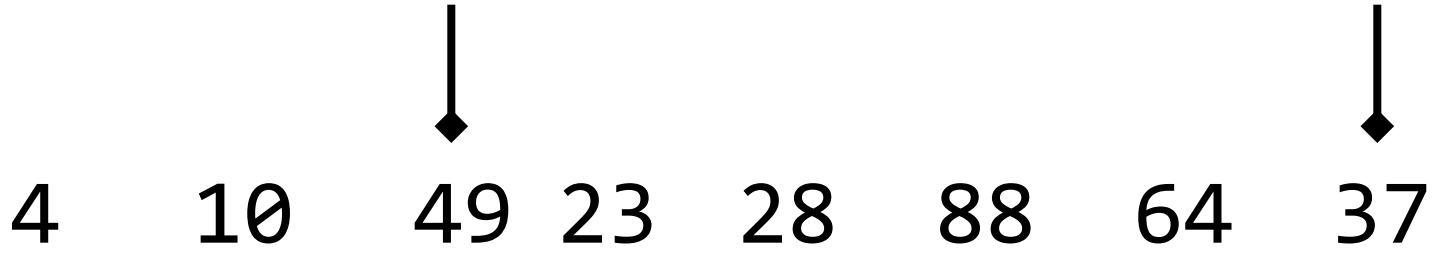


Smallest: 23



Smallest: 23

Smallest: 23

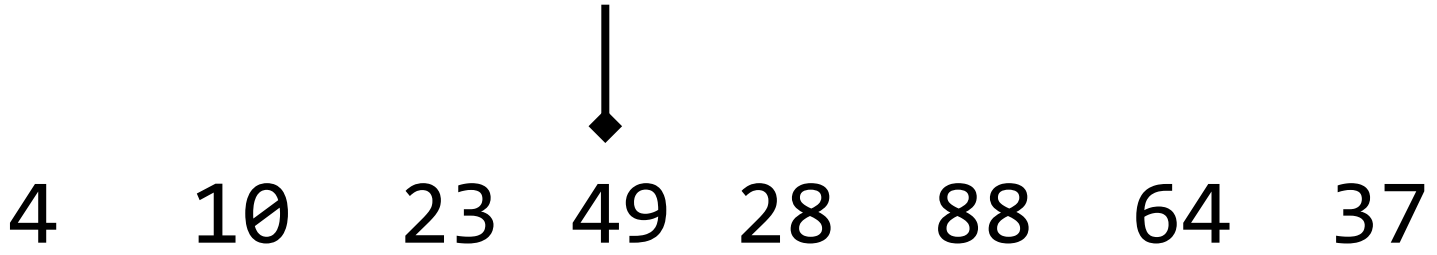


Smallest: 23

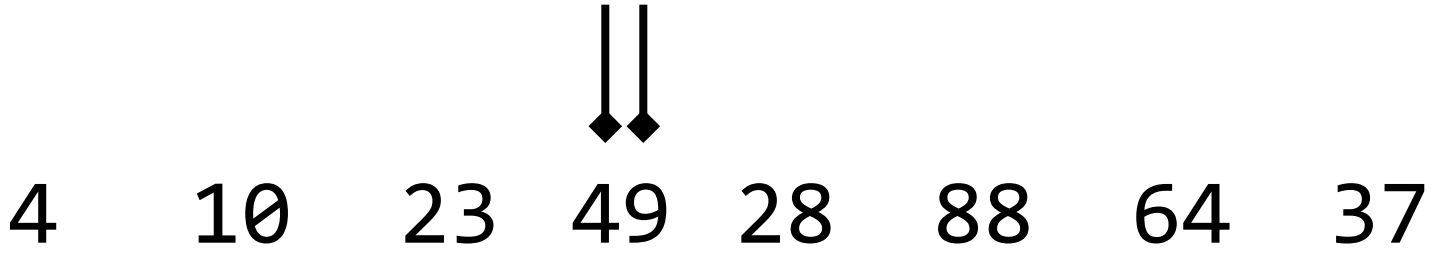
4 10 49 23 28 88 64 37

Smallest: 23

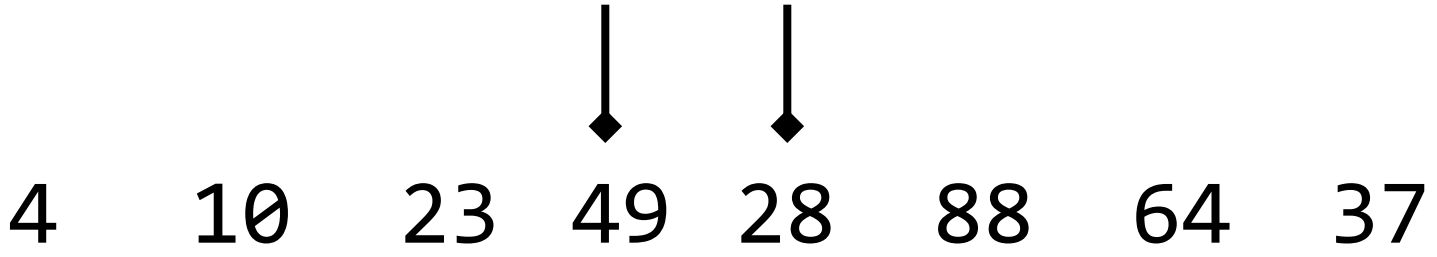
Smallest:



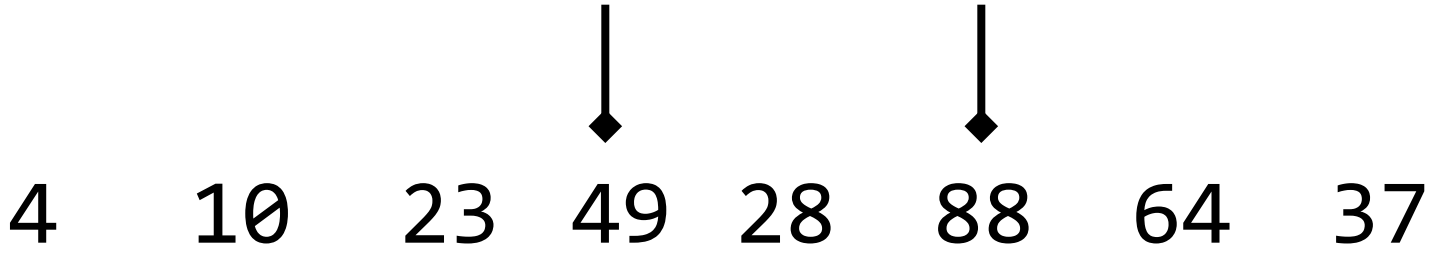
Smallest: 49



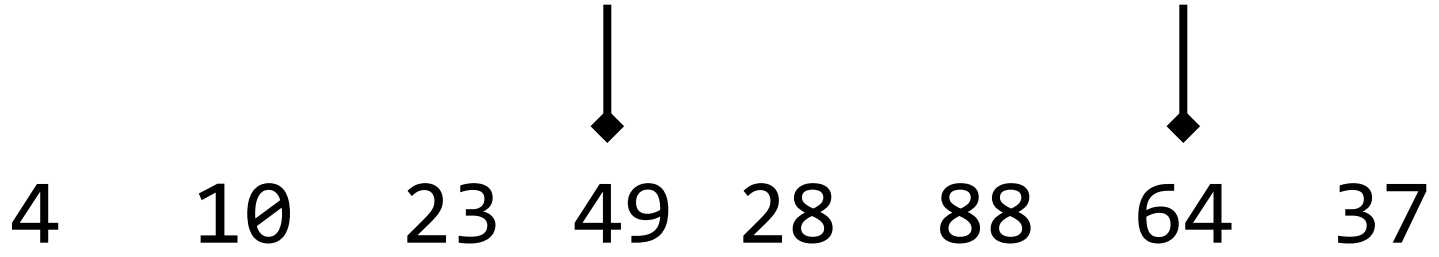
Smallest: 28



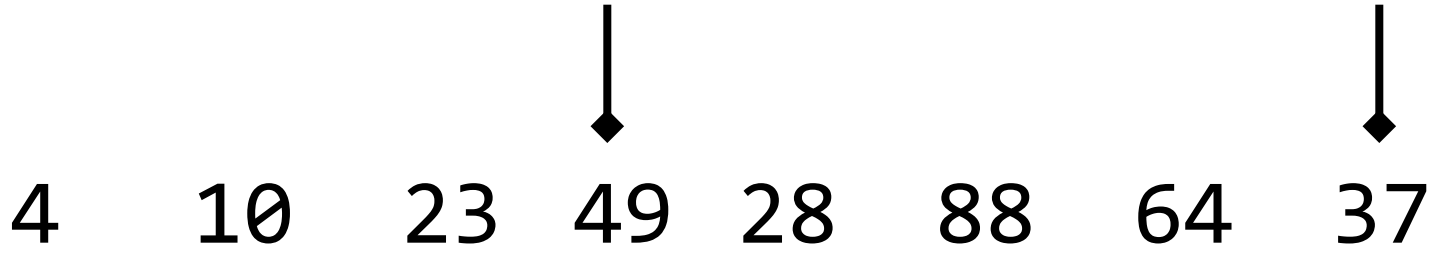
Smallest: 28



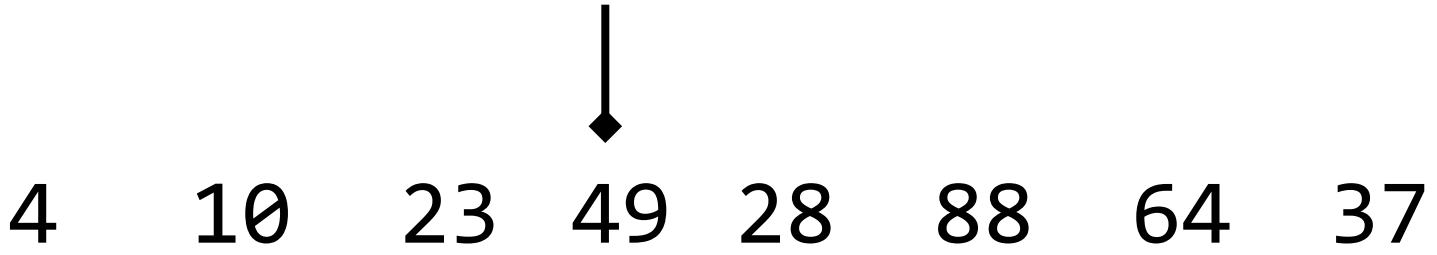
Smallest: 28



Smallest: 28

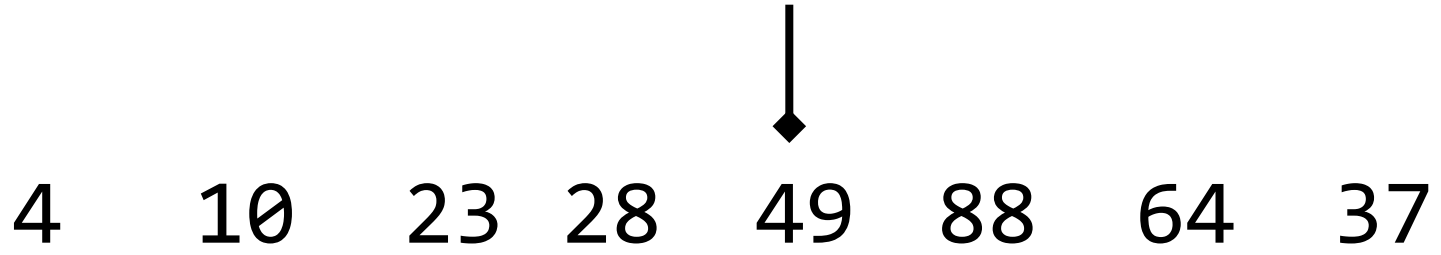


Smallest: 28

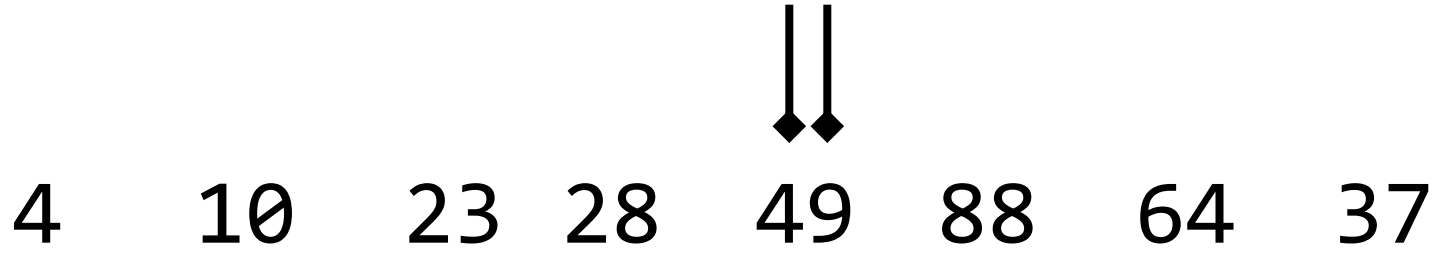


Smallest: 28

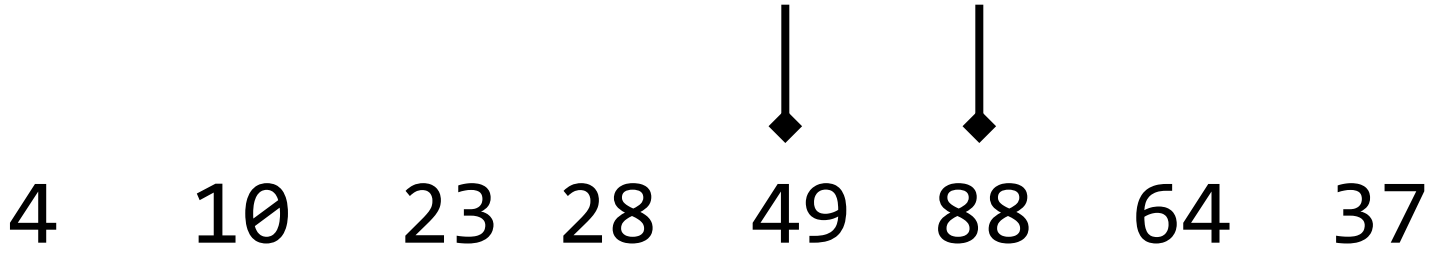
Smallest:



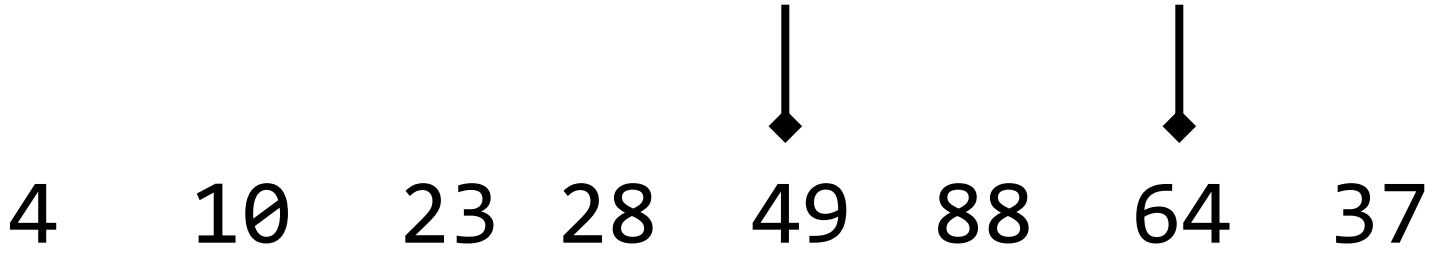
Smallest: 49



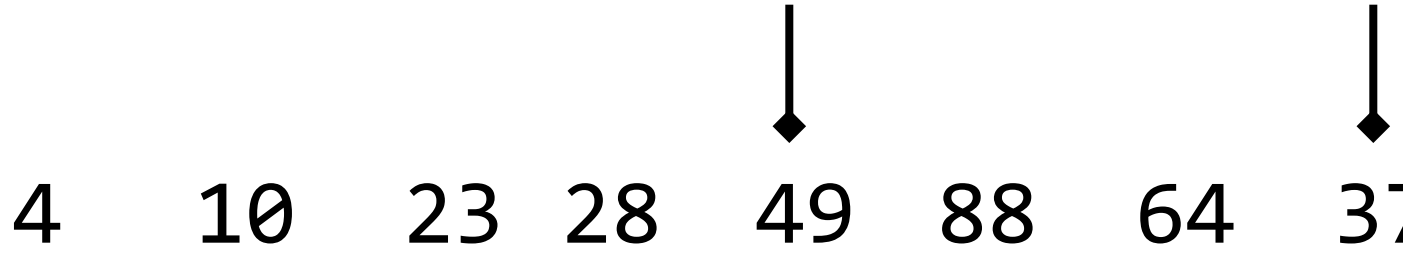
Smallest: 49



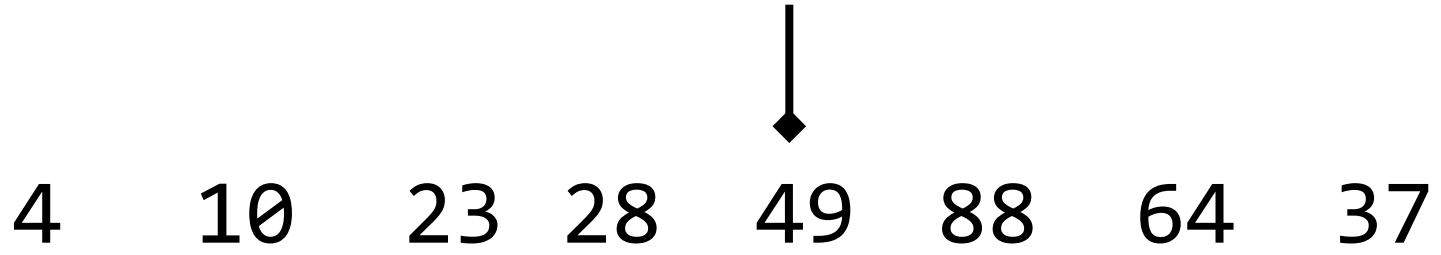
Smallest: 49



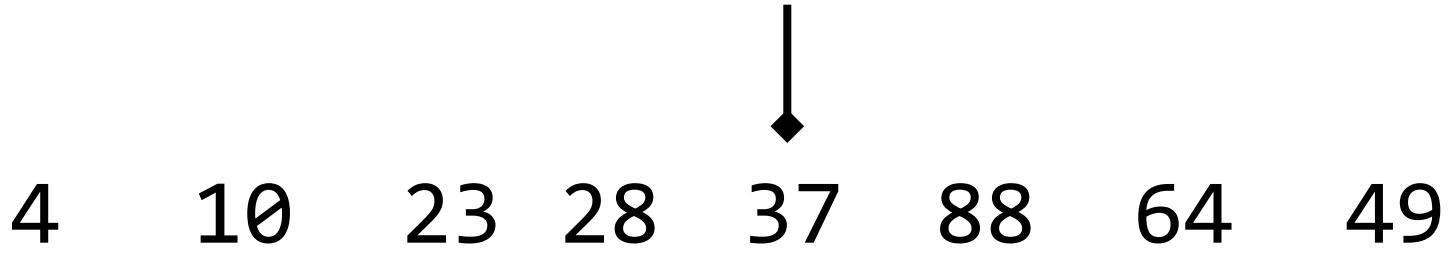
Smallest: 37



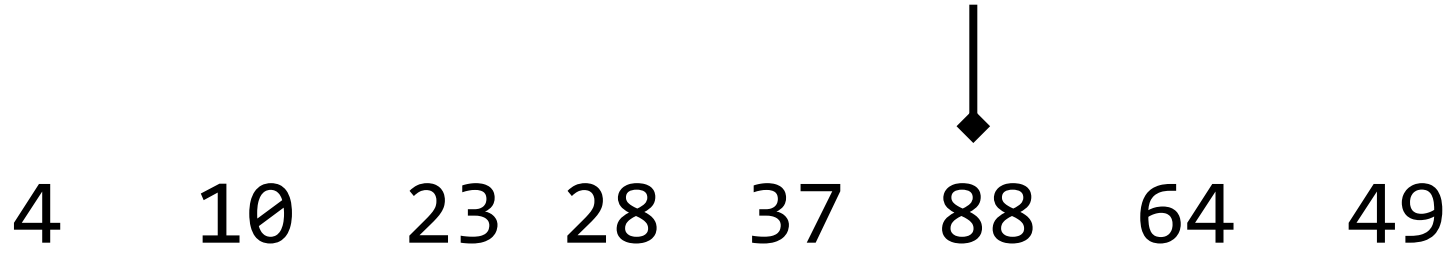
Smallest: 37



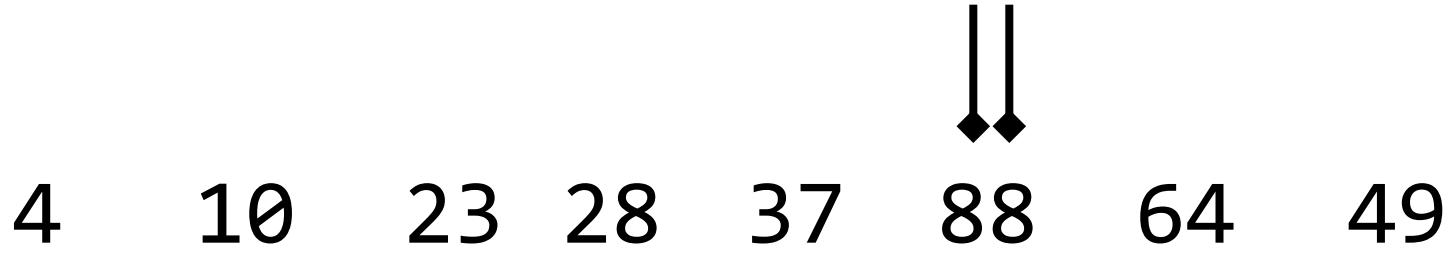
Smallest: 37



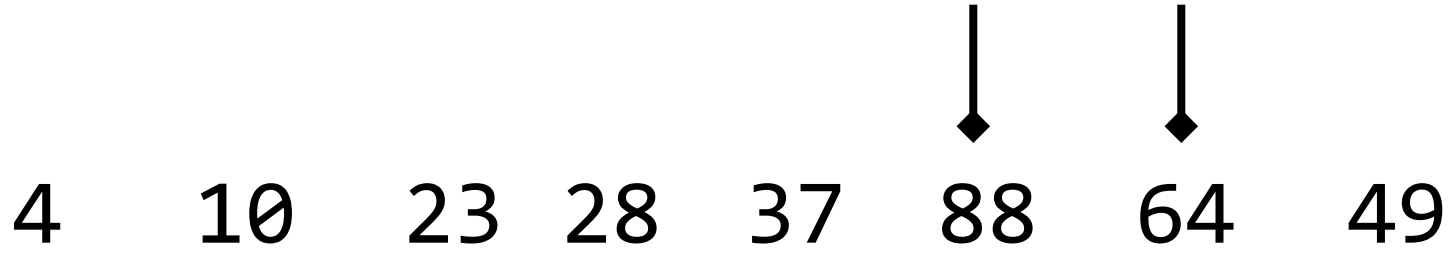
Smallest:



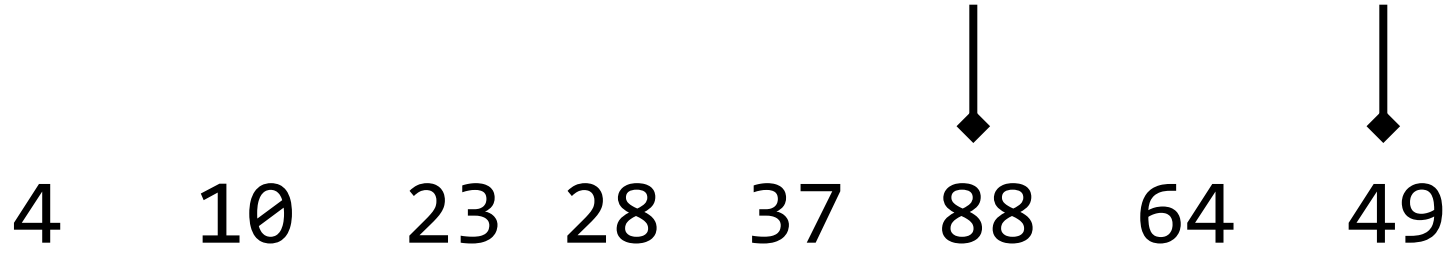
Smallest: 88



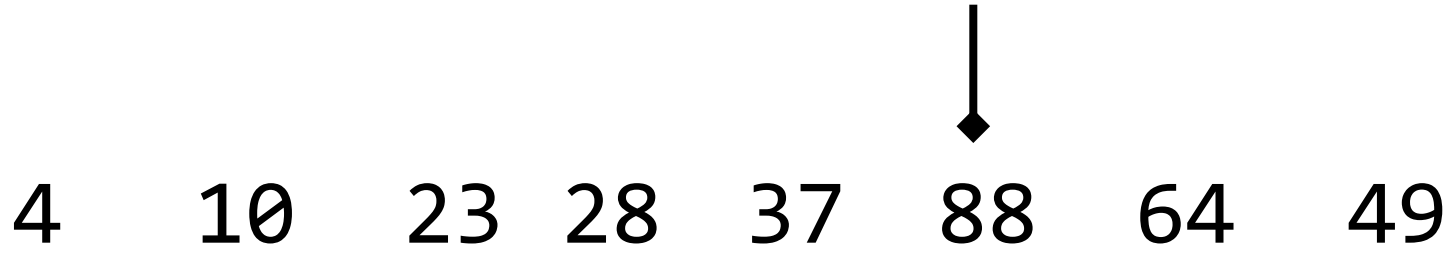
Smallest: 64



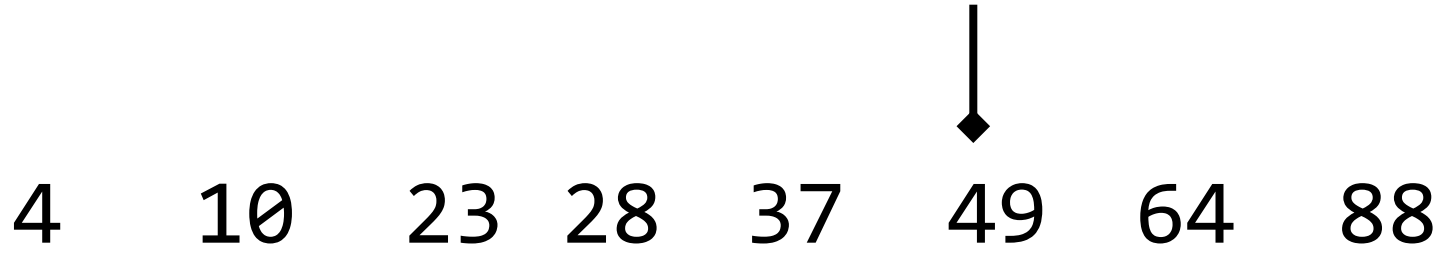
Smallest: 49



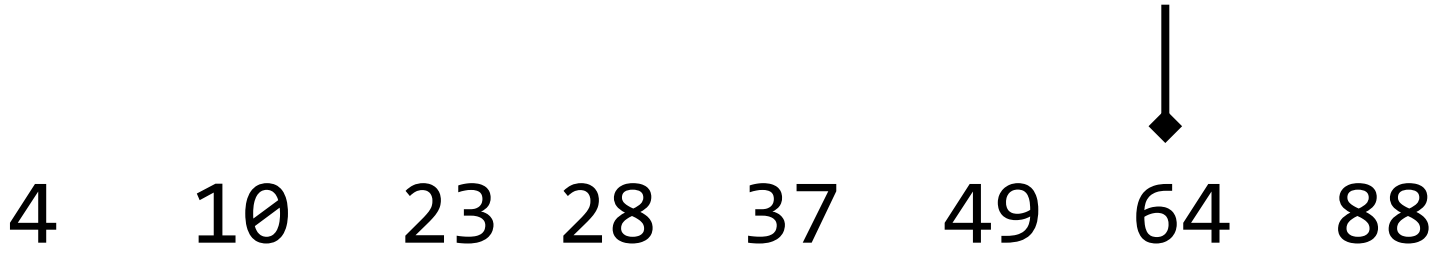
Smallest: 49



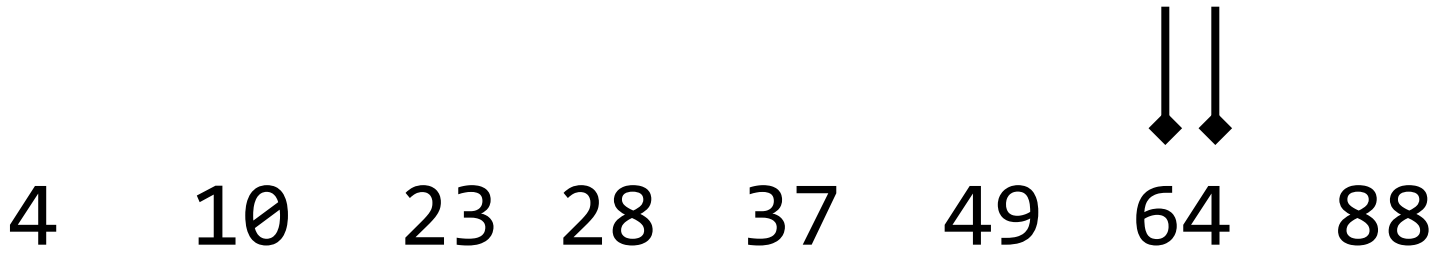
Smallest: 49



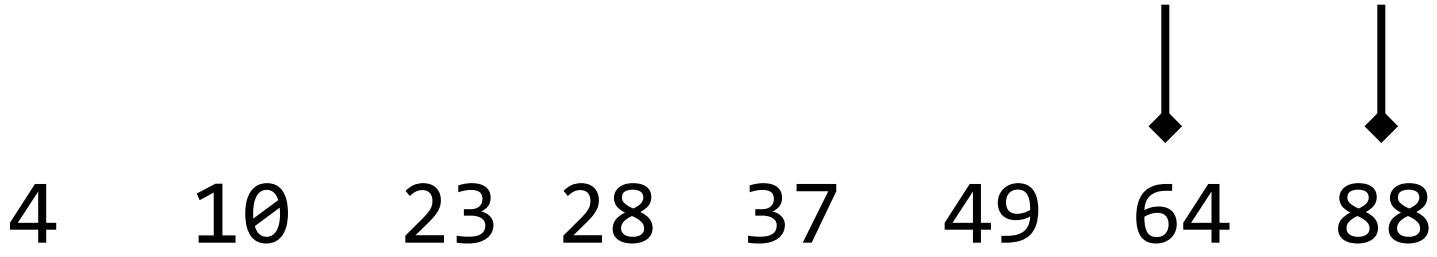
Smallest:



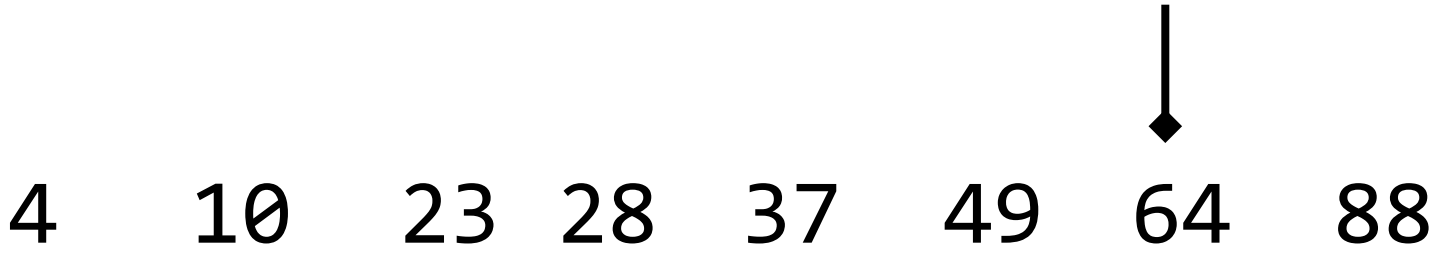
Smallest: 64



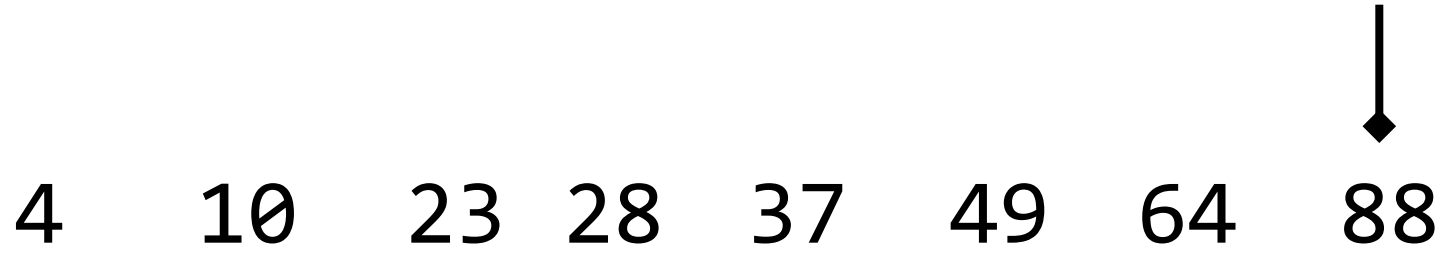
Smallest: 64



Smallest: 64



Smallest:



```
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

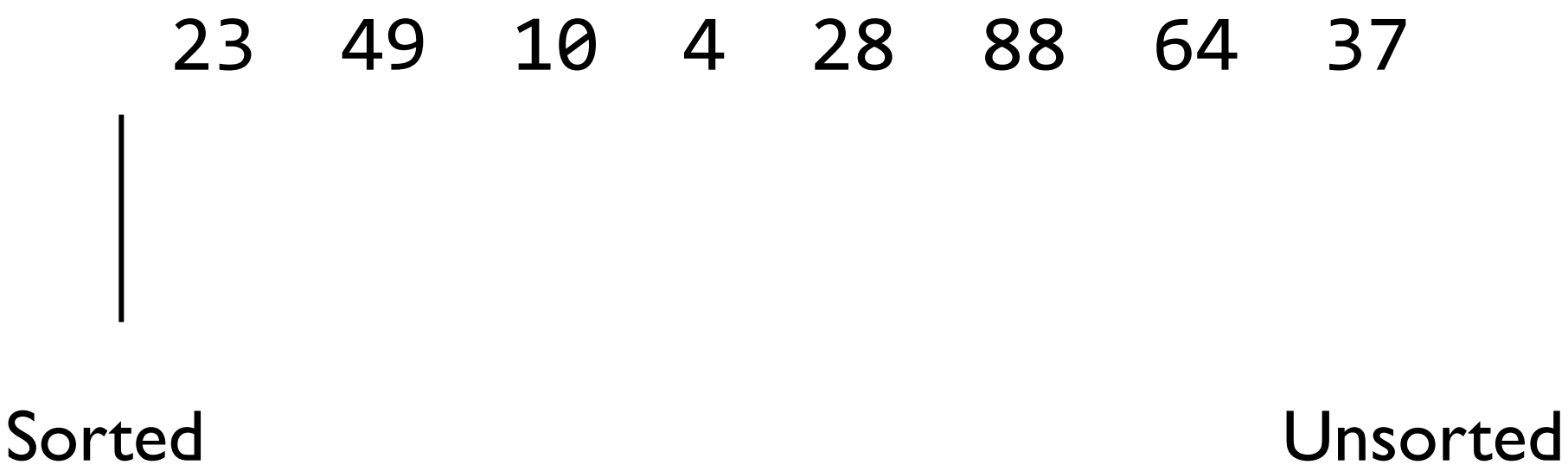
- Doesn't adapt; always goes through the entire list each time
- Goes through the entire list, compares each item to every other item
- So it's approximately n * n = n^2 comparisons
- Minimizes the number of swaps (occasionally useful)

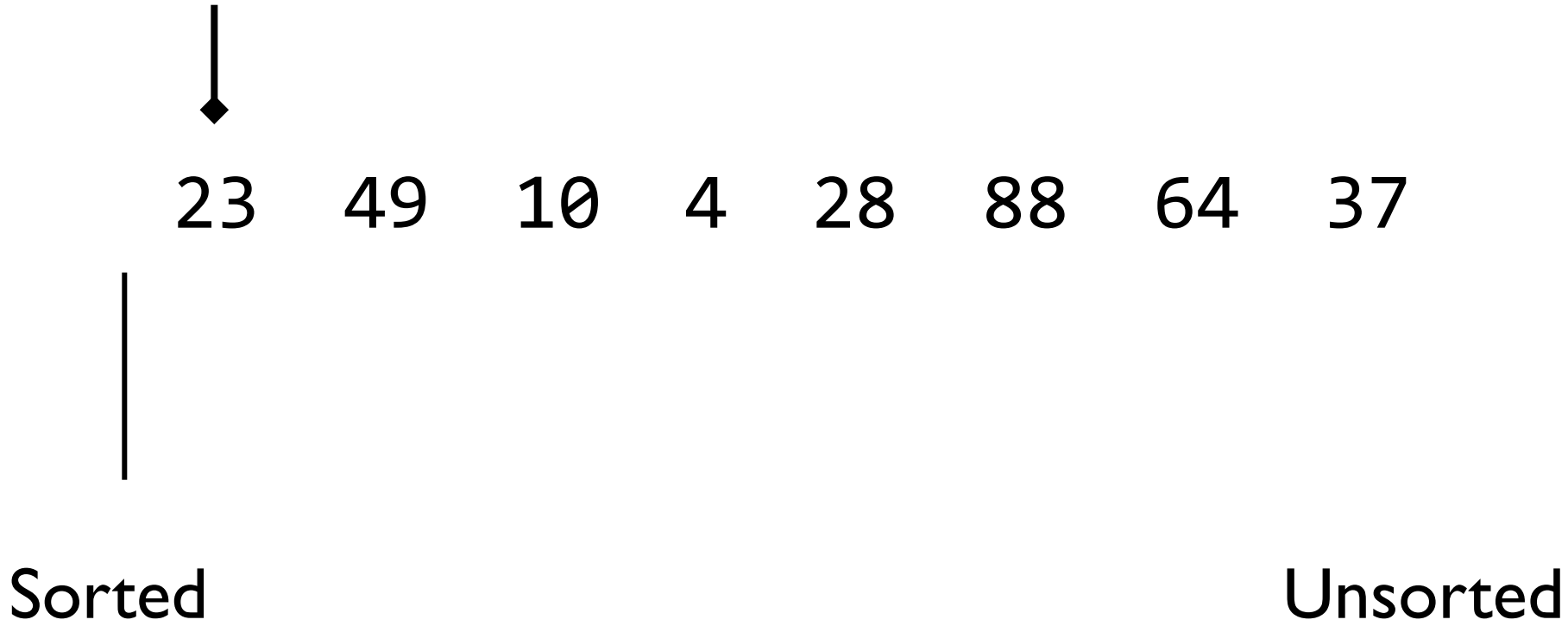
Insertion Sort

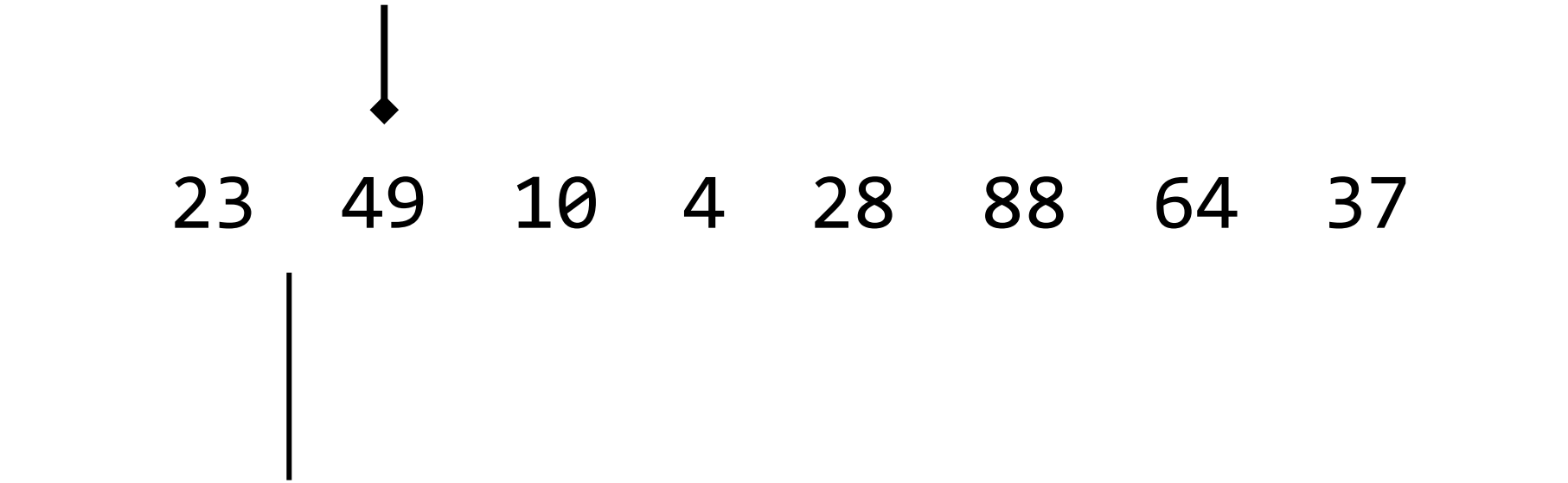
- Simplest of the "insertion sorts"
- This is probably the sort algorithm most commonly used by normal people for day-to-day tasks

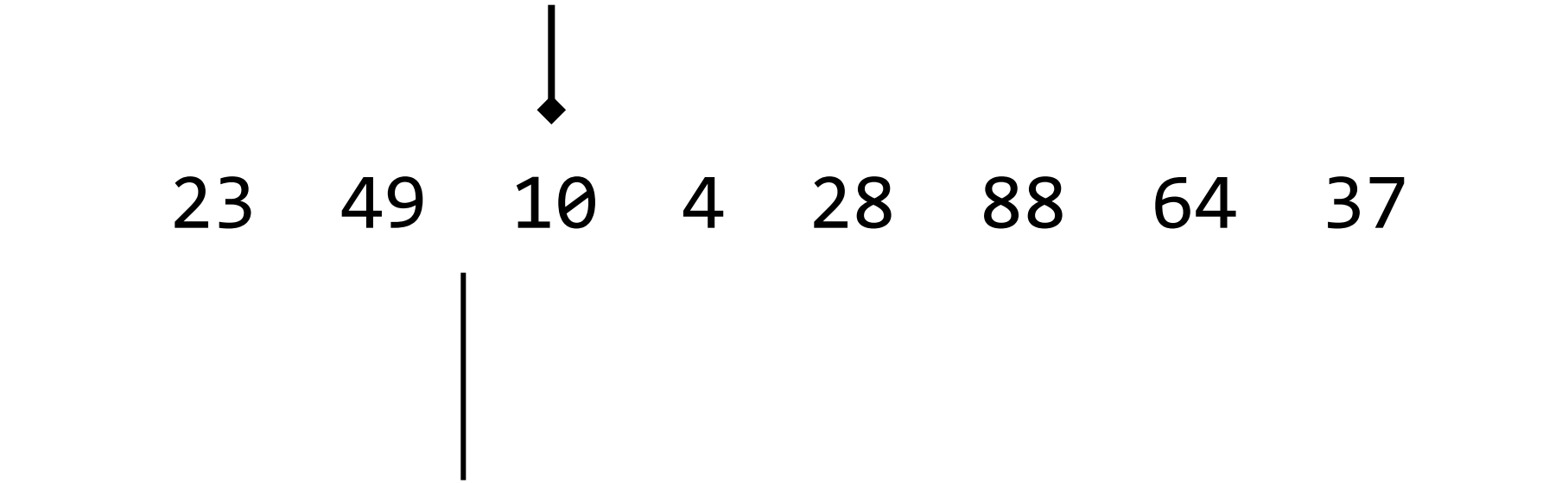
Divide the list into two parts, a "sorted" part and an "unsorted" part. (Right now, "sorted" is empty, and "unsorted" contains the entire list.)

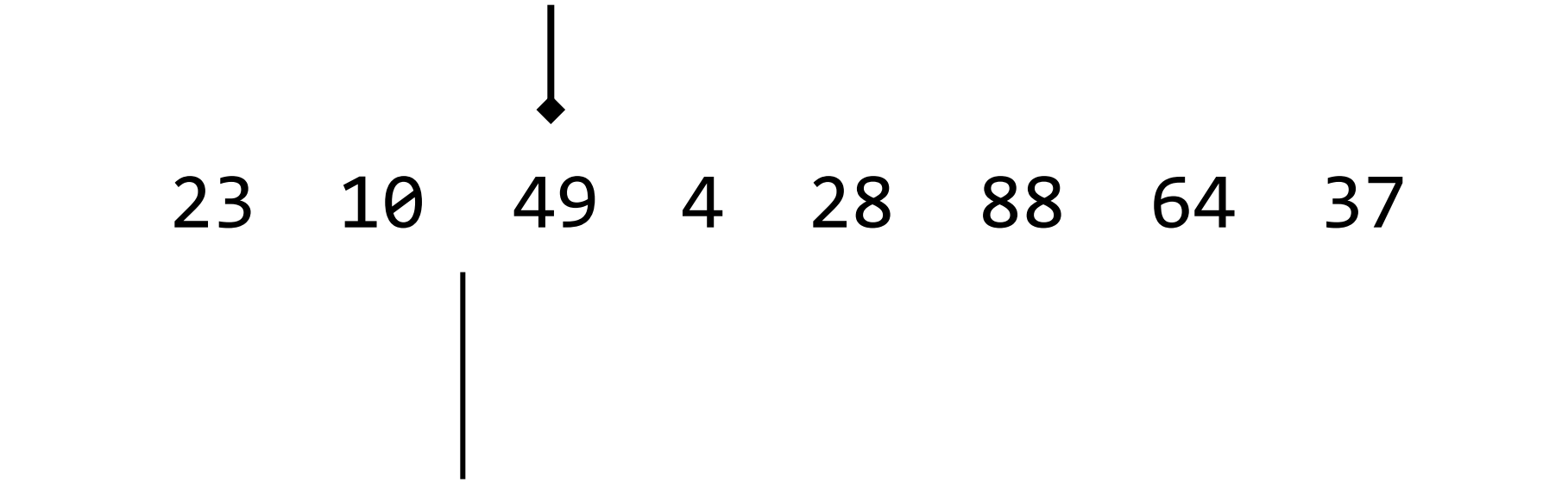
```
for (each item in the unsorted list)
{
    Insert it in the appropriate place in the sorted list
}
```

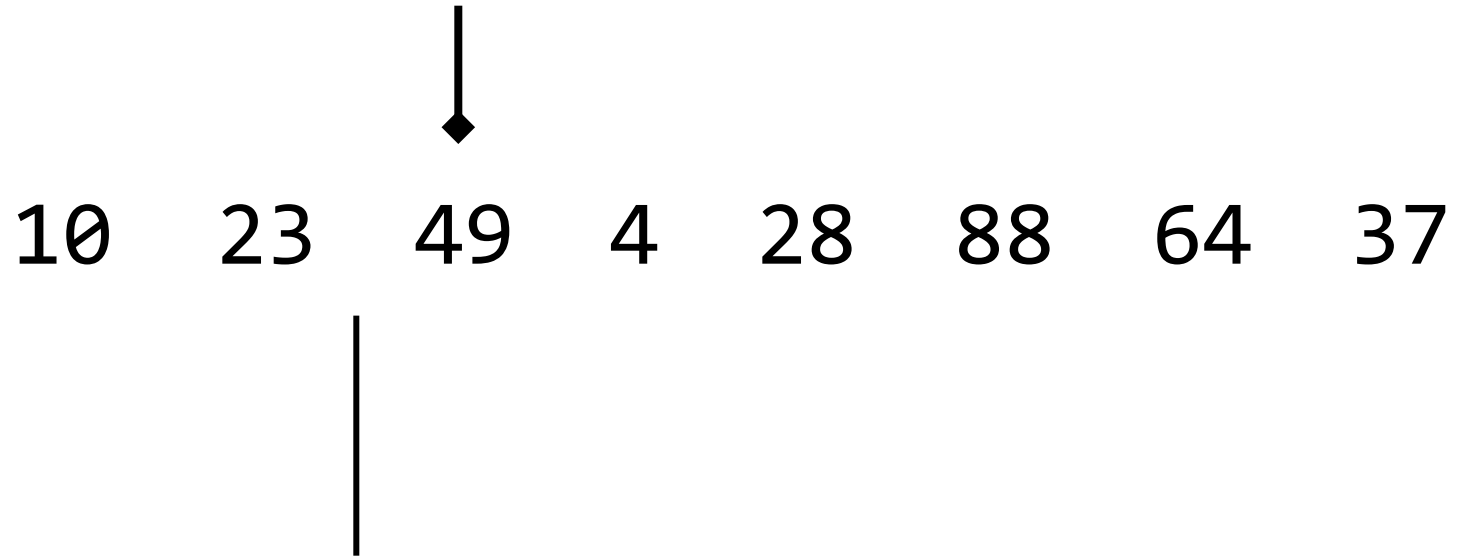

23 49 10 4 28 88 64 37
Sorted
Unsorted


23 49 10 4 28 88 64 37
Sorted
Unsorted


23 49 10 4 28 88 64 37
Sorted
Unsorted

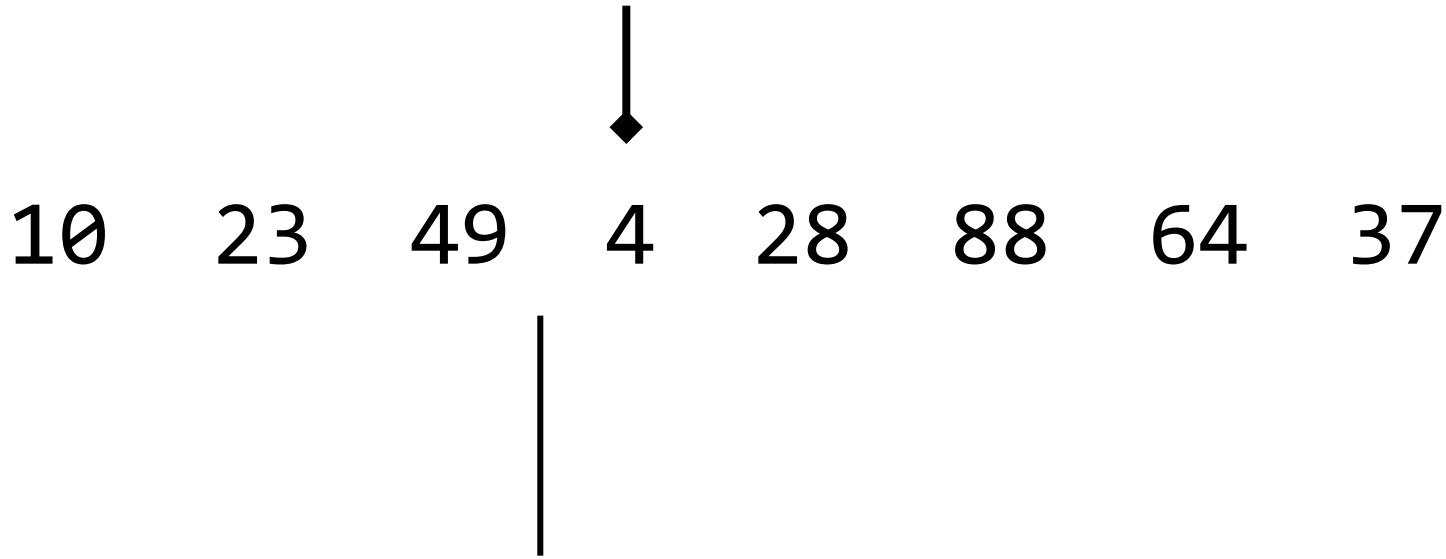

23 49 10 4 28 88 64 37
Sorted
Unsorted


23 10 49 4 28 88 64 37
Sorted
Unsorted



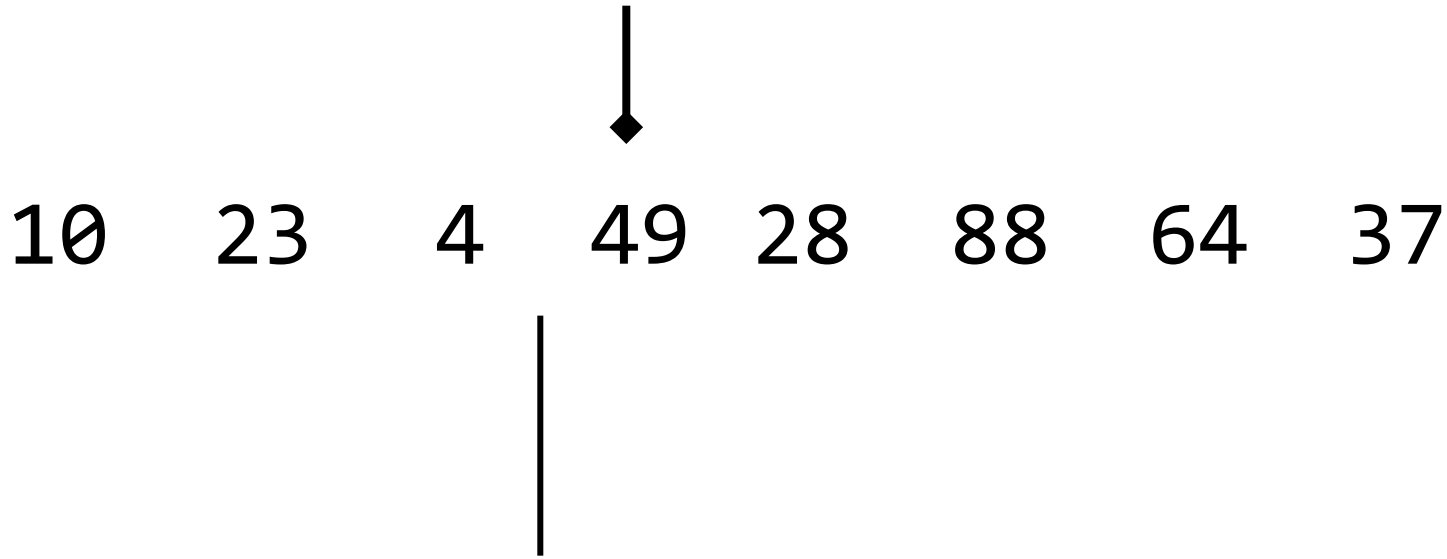
Sorted

Unsorted



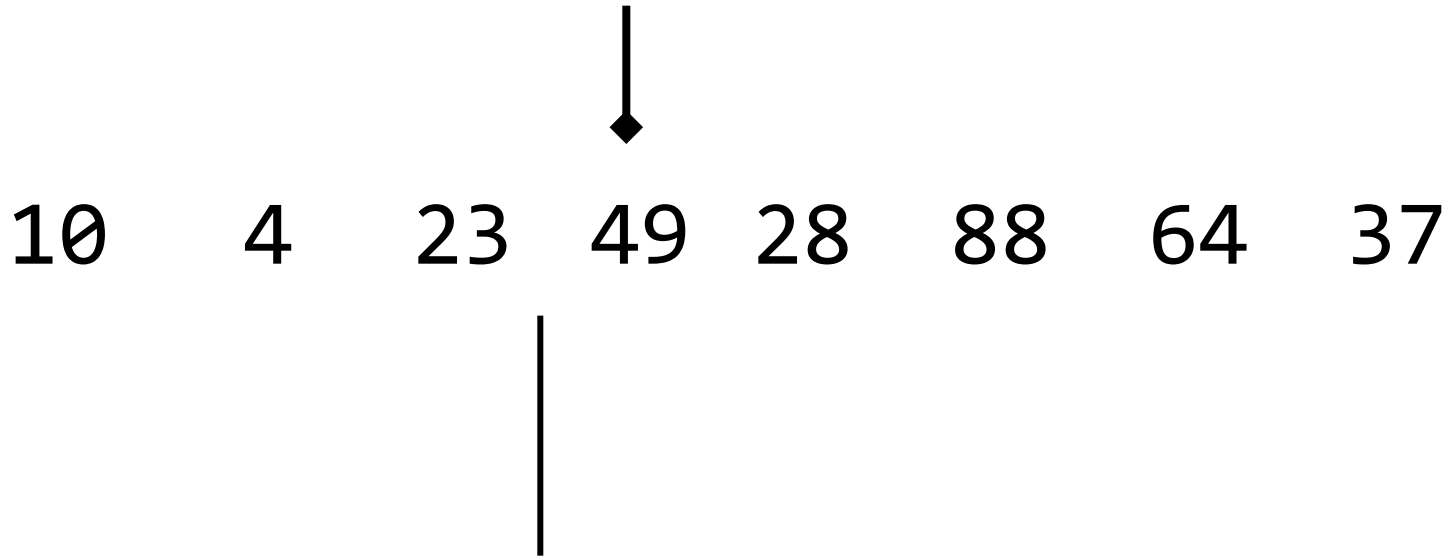
Sorted

Unsorted



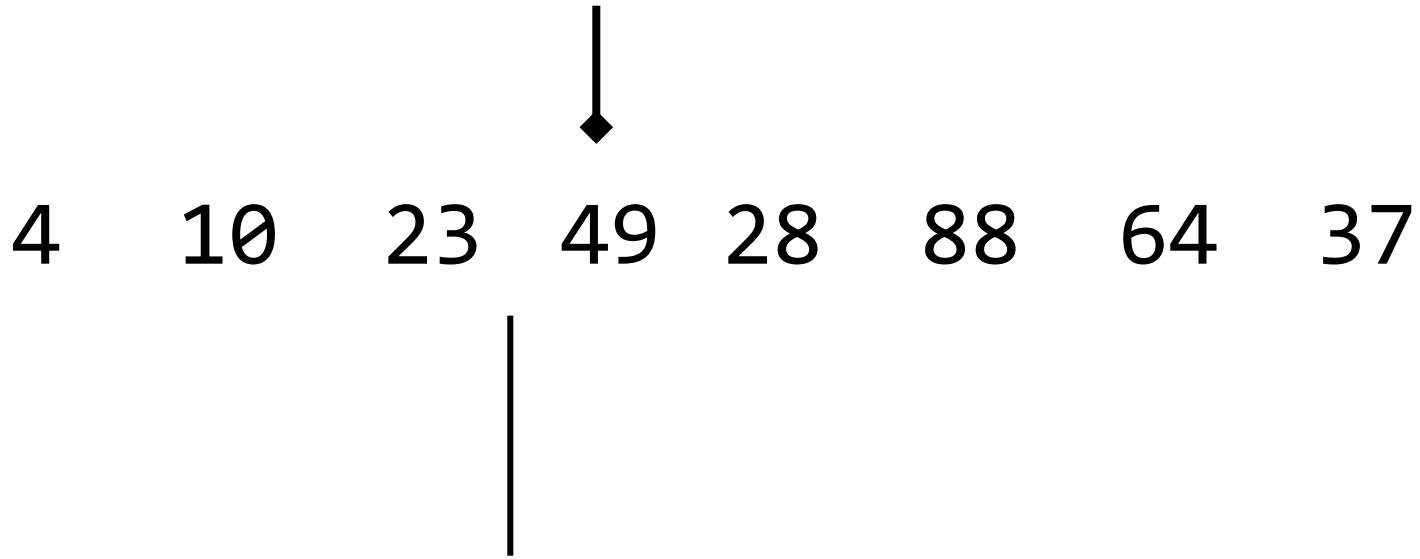
Sorted

Unsorted



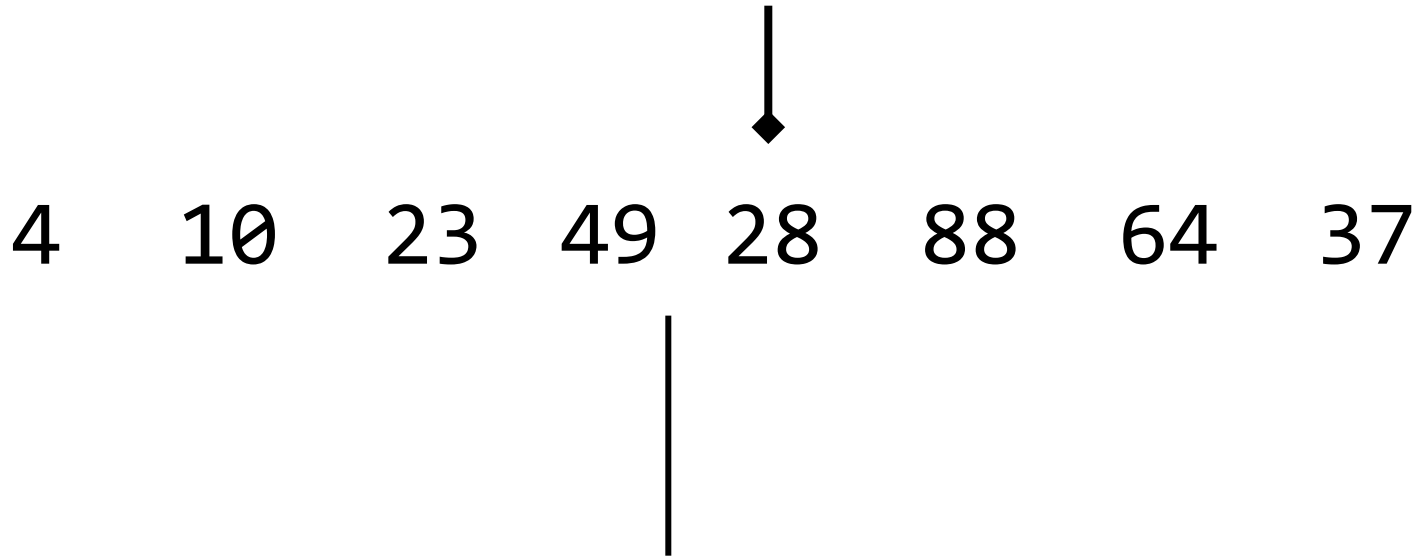
Sorted

Unsorted



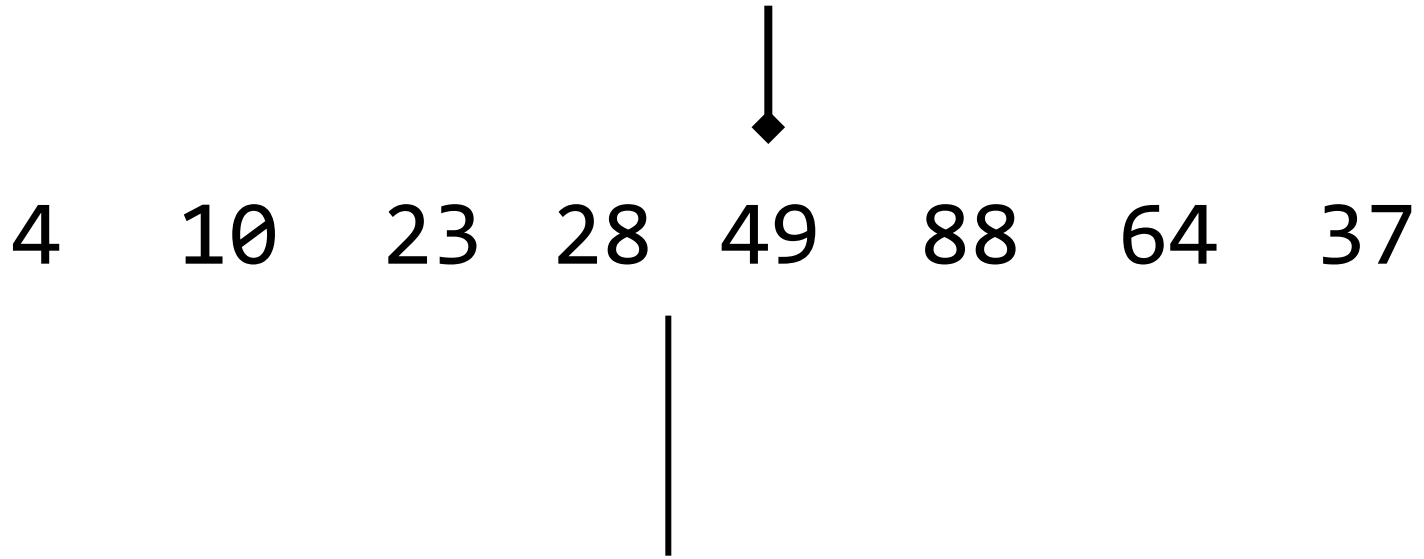
Sorted

Unsorted



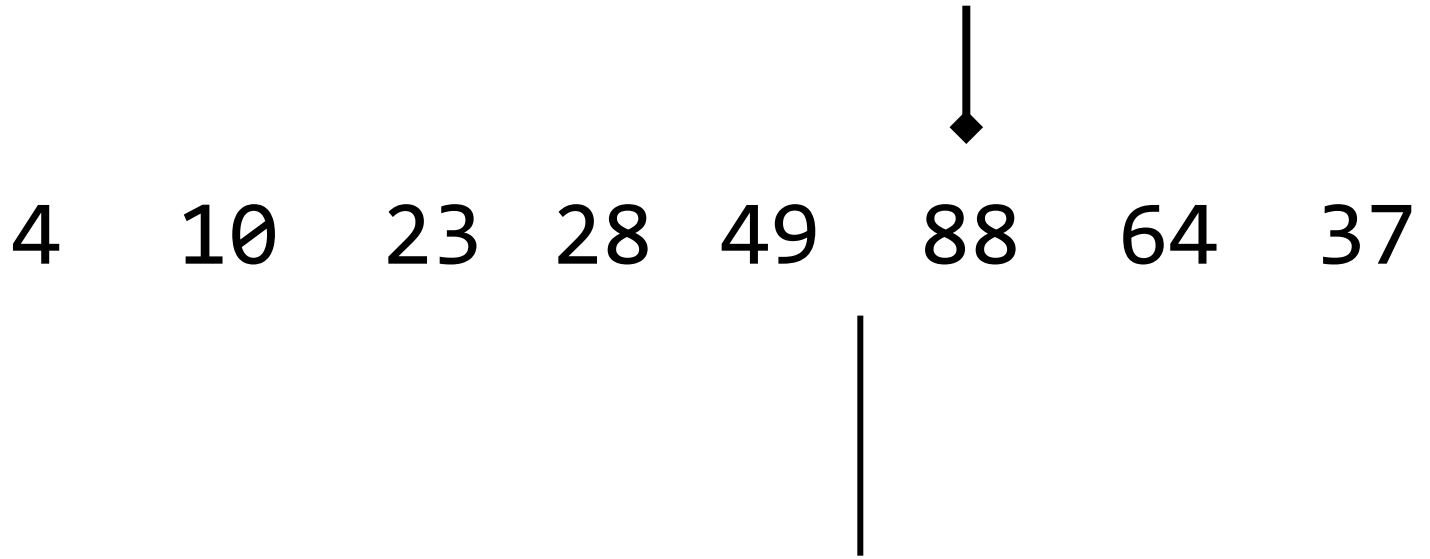
Sorted

Unsorted



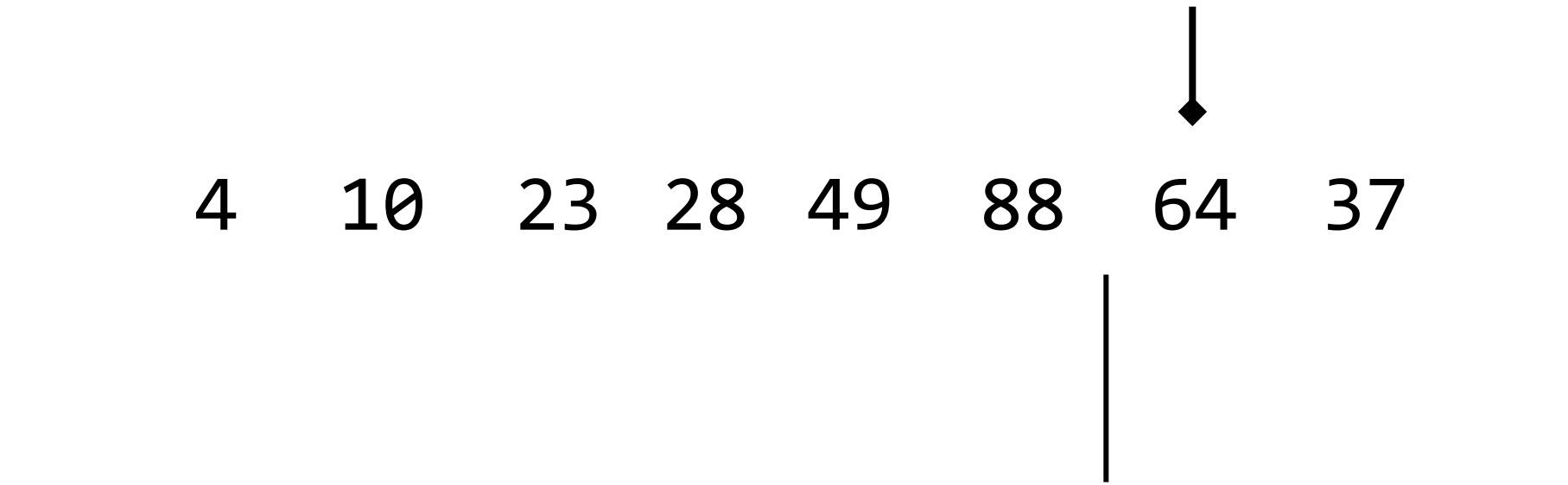
Sorted

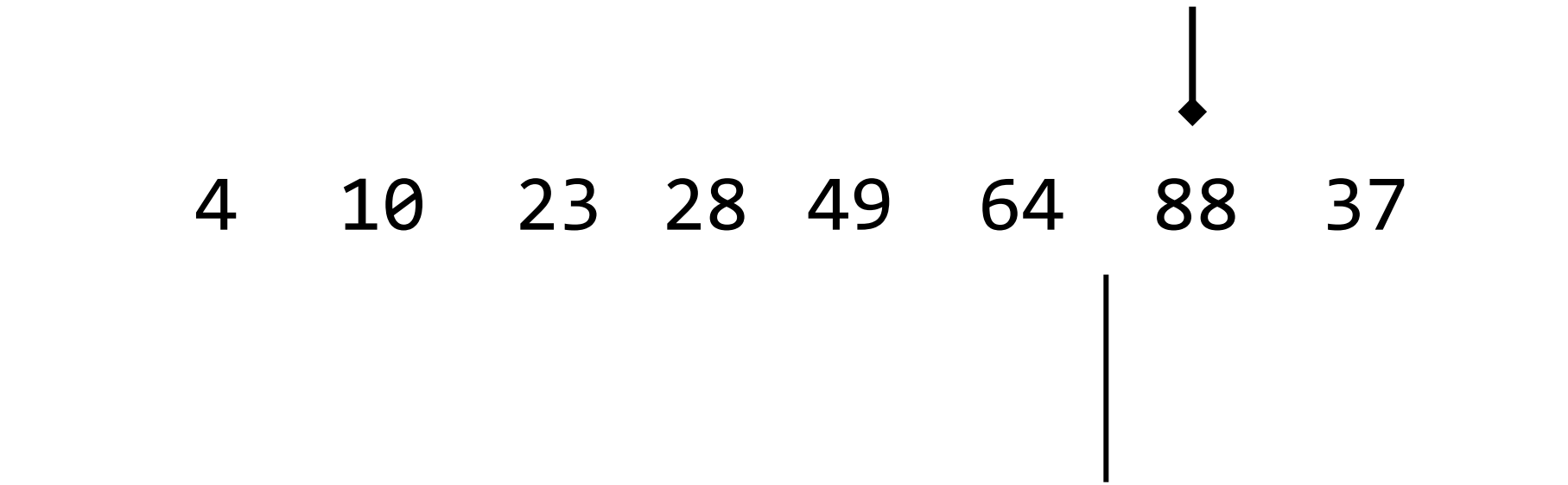
Unsorted

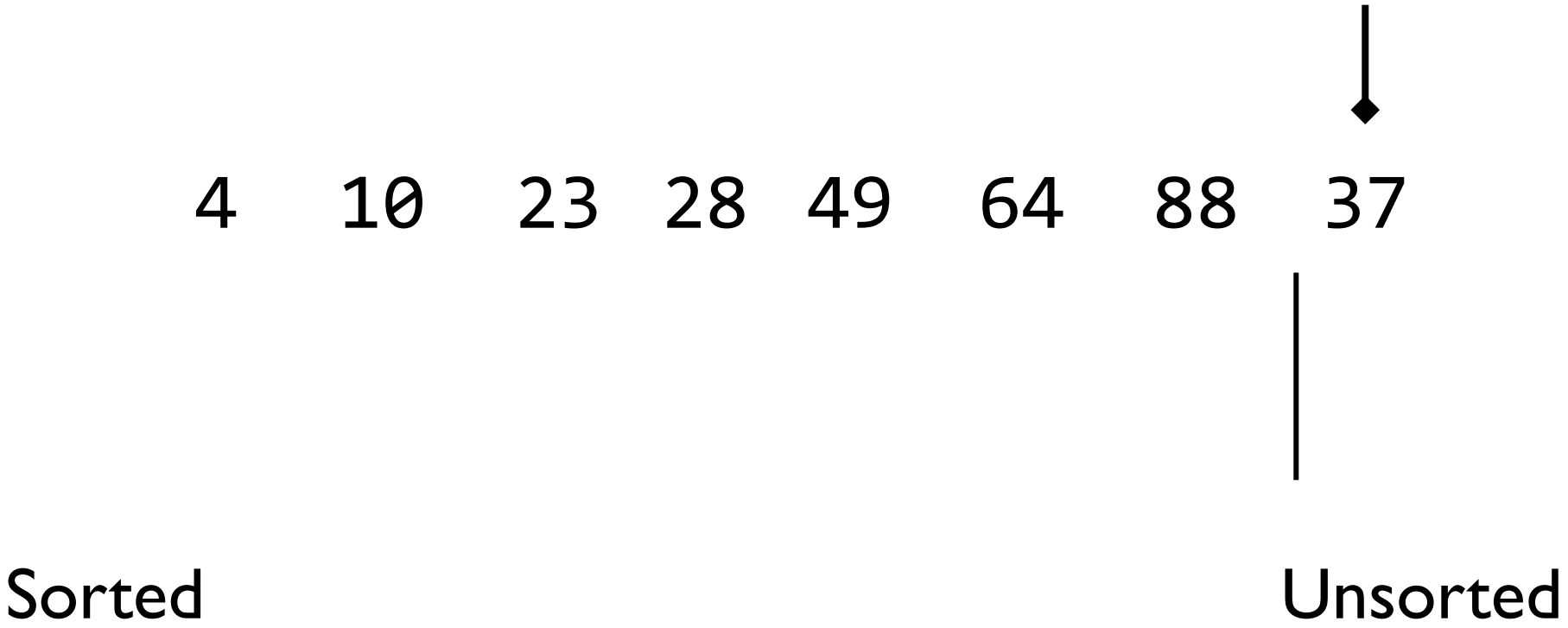


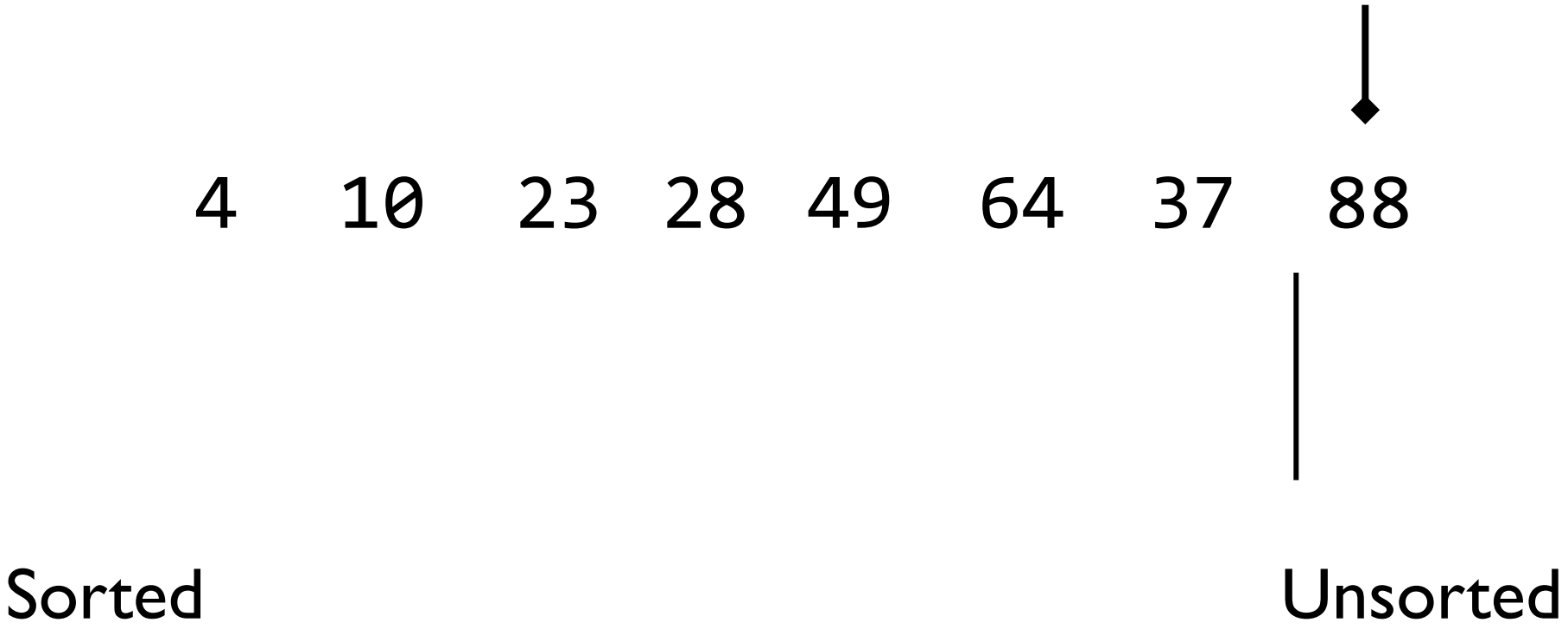
Sorted

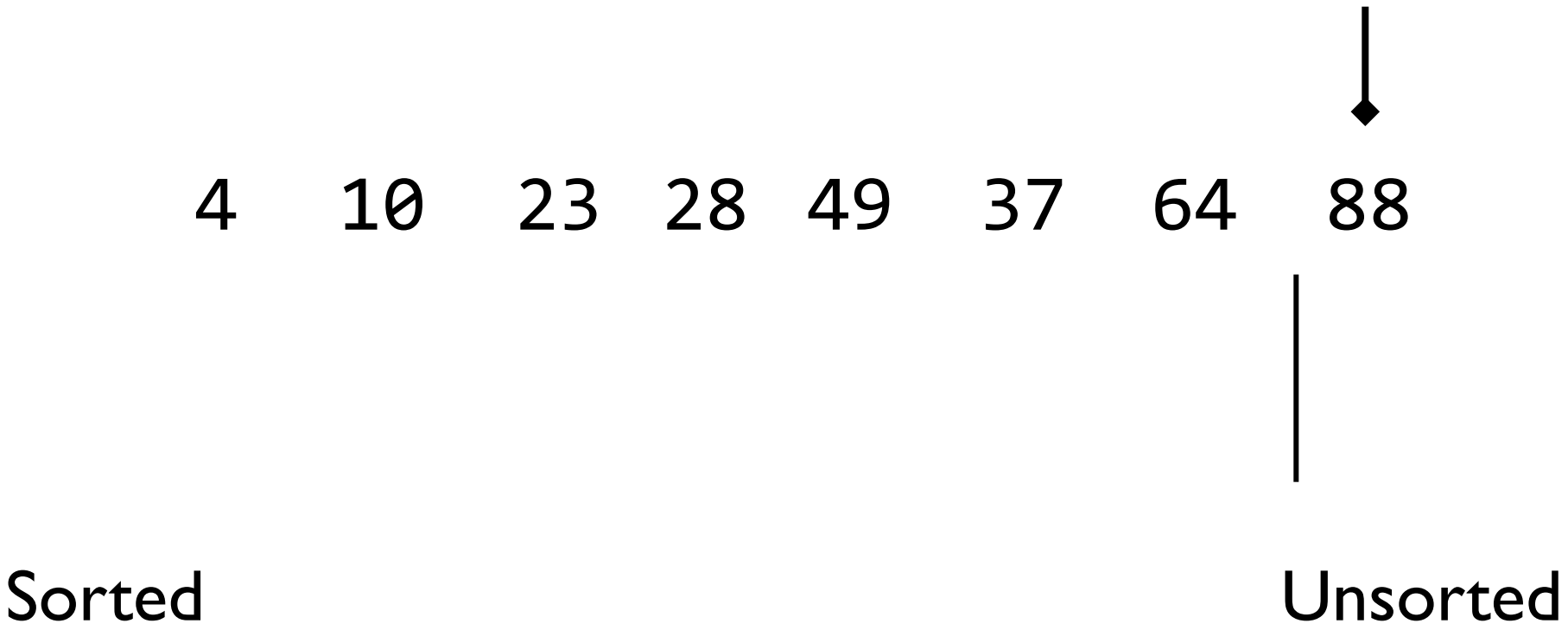
Unsorted


4 10 23 28 49 88 64 37
Sorted
Unsorted

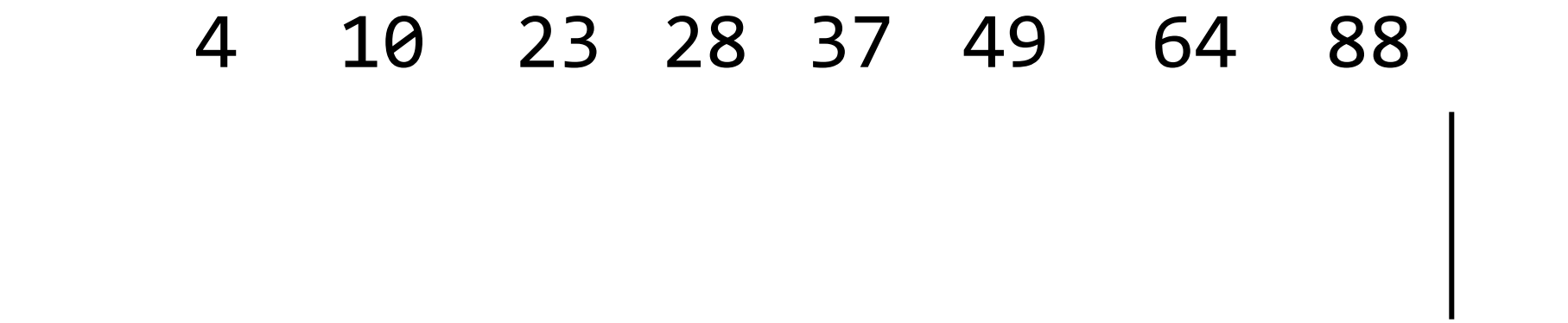

4 10 23 28 49 64 88 37
Sorted
Unsorted


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]);
    }
  }
}
```

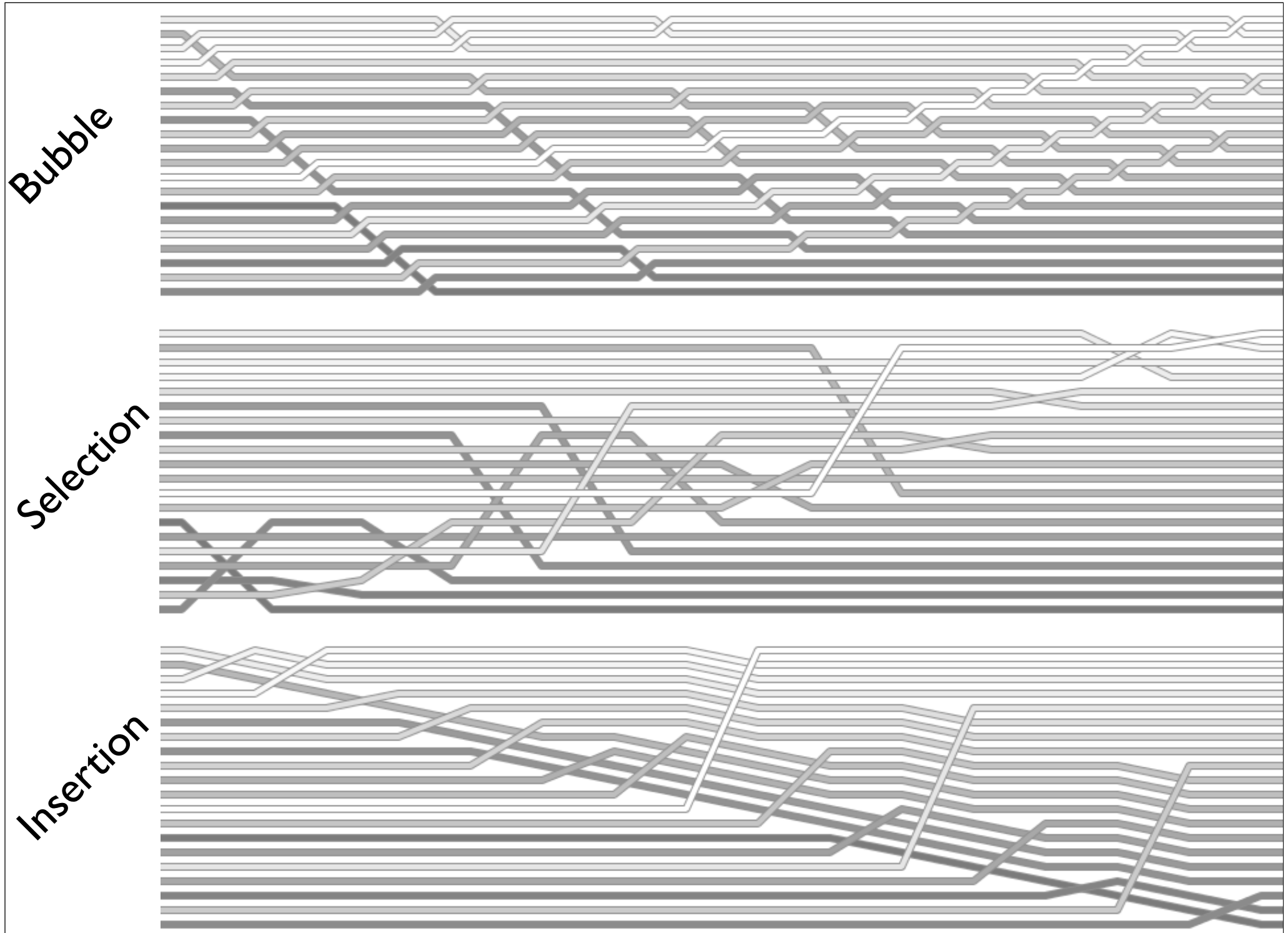
Insertion Sort Performance

- 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.
}
```



http://sortvis.org/