Introduction to Algorithms and Data Structures

Lecture 10: Sorting (1)
Bubble sort and Insertion sort

Professor Ryuhei Uehara,
School of Information Science, JAIST, Japan.
uehara@jaist.ac.jp
http://www.jaist.ac.jp/~uehara

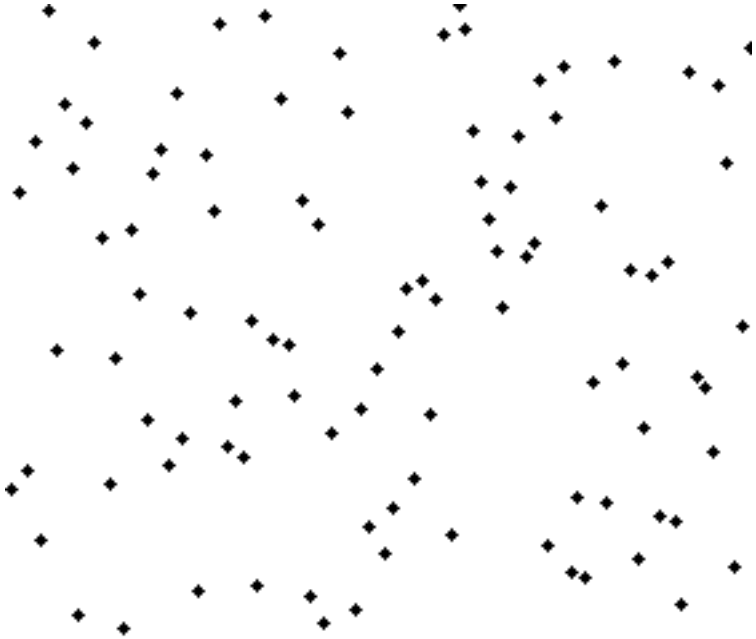
Sorting

- Make given set of data in order
  - Numerical data: increasing/decreasing order

65	12	46	97	56	33	75	53	21	Input
12	21	33	46	53	56	65	75	93	Increasing
93	75	65	56	53	46	33	21	12	Decreasing

  - String data: lexicographical ordering
    e.g., aaa, aab, aba, abb, baa, bab, bbc, bcb
- Tons of sorting algorithms
  - Bubble sort, insertion sort, heap sort, merge sort, quick sort, and counting sort, and so on…

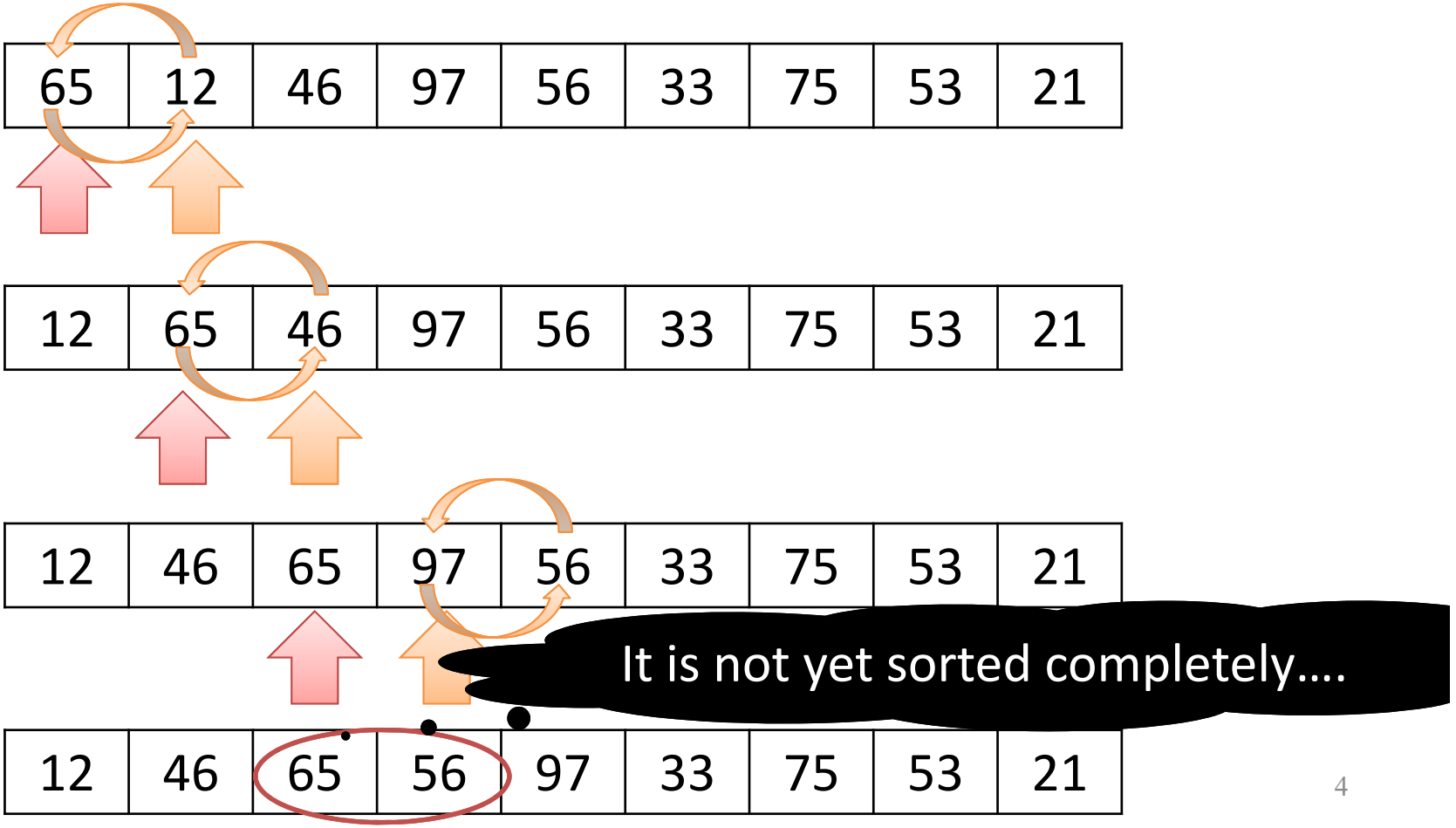
See the volume 3 of TAOCP by D. Knuth for more algorithms…



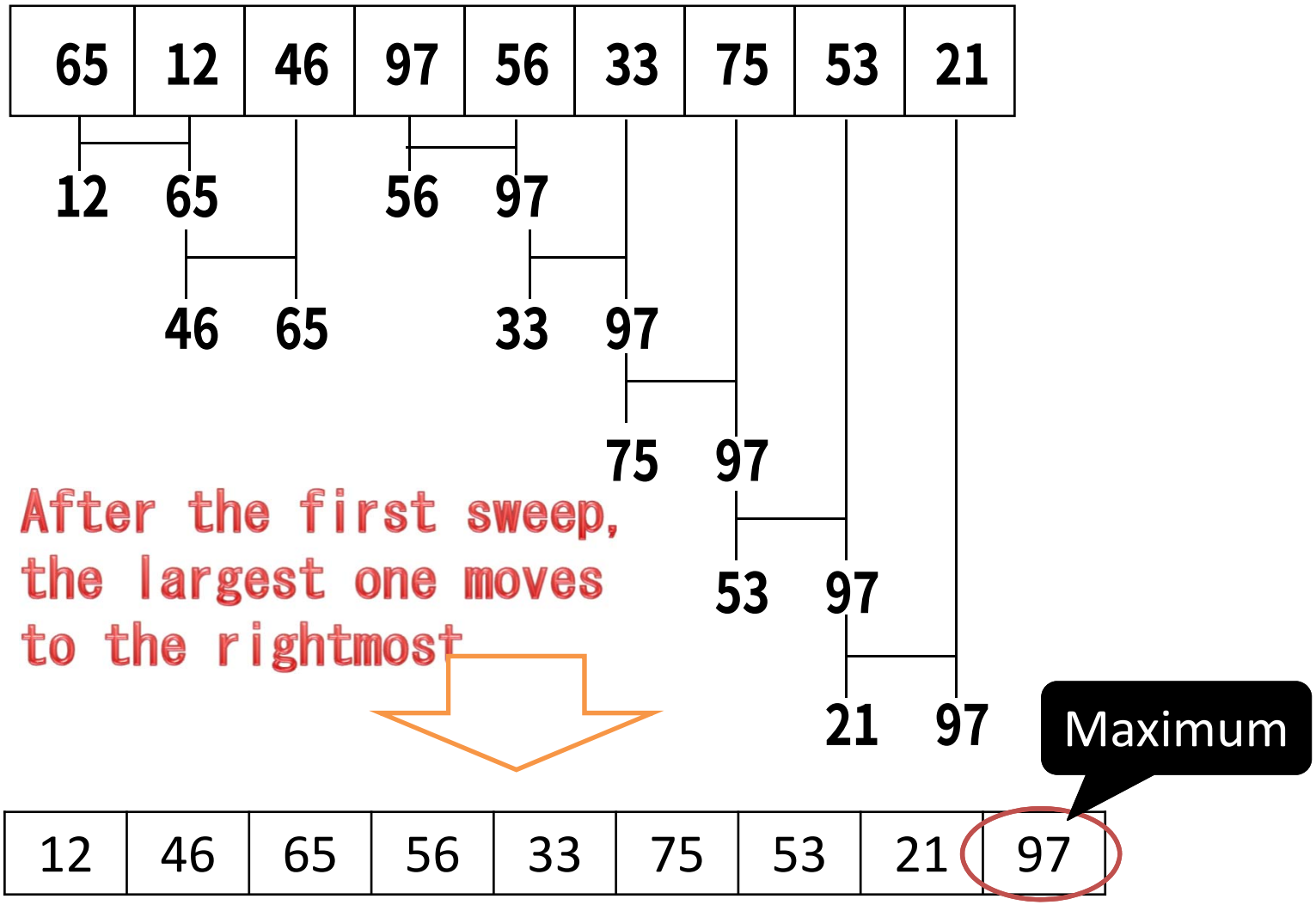
BUBBLE SORT

Bubble sort

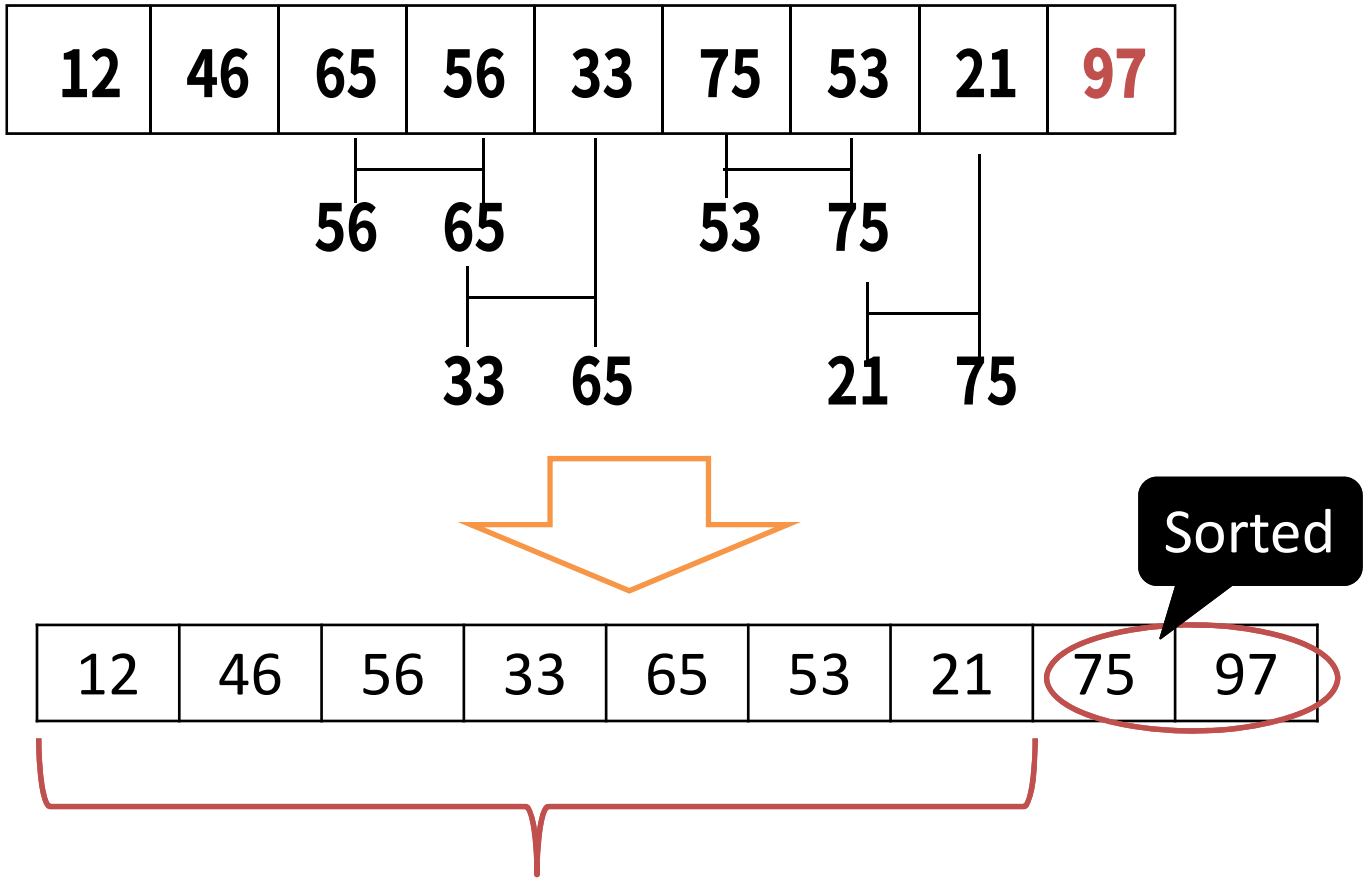
- From left to right, if they are not in order, swap them



Bubble sort: 1st sweep



Bubble sort: 2nd sweep

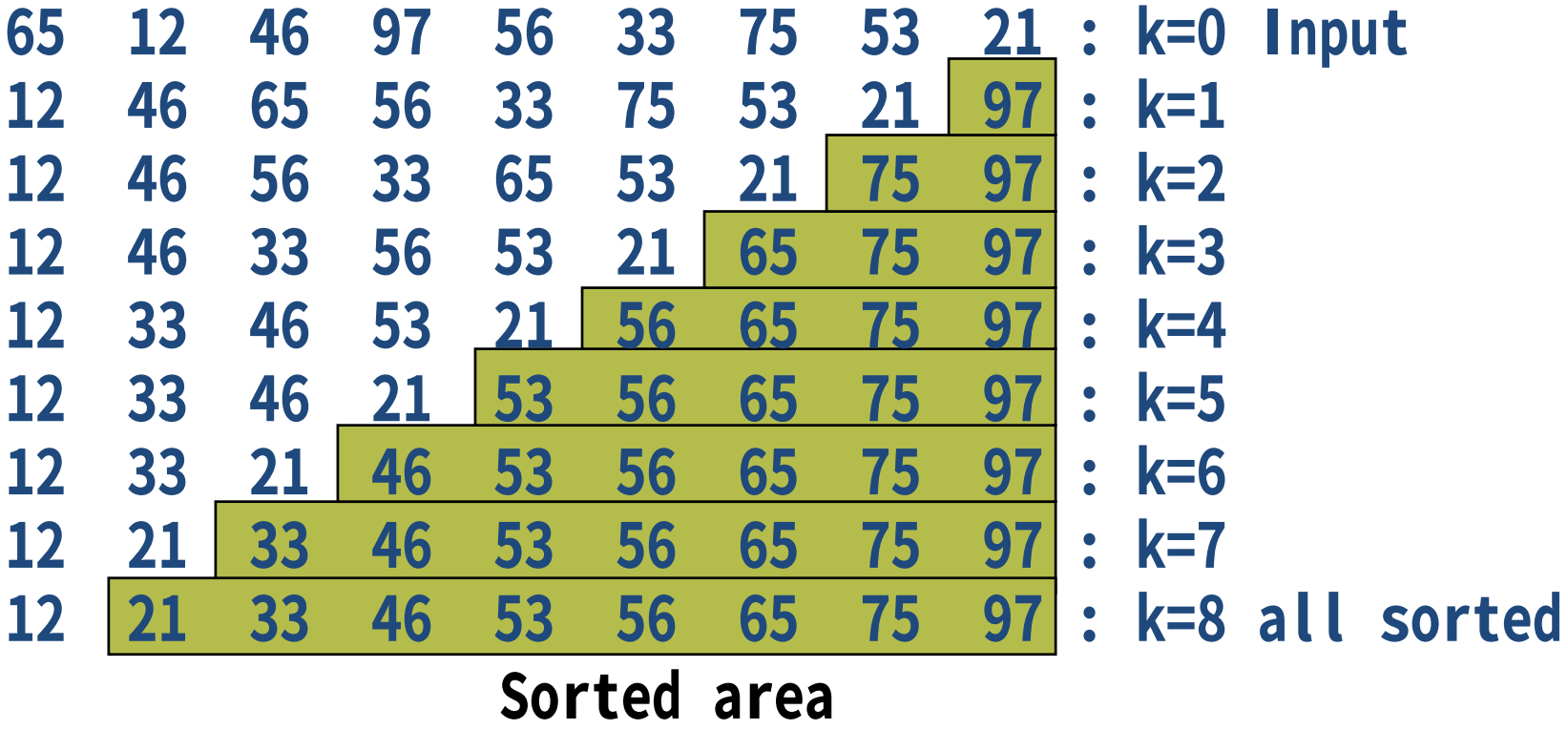


They are not sorted yet ➔ next sweep

Bubble sort: Number of sweeps

- By k sweeps, k data are sorted →

  the number of sweeps to sort all is n-1 times.



Bubble sort: Time complexity

- Program

```
for(k=1; k<n; k=k+1)
   for(i=0; i<n-k; i=i+1)
      if(data[i] > data[i+1])
         swap(&data[i], &data[i+1]);
```

- # of comparisons: $\sum_{k=1}^{n-1}(n-k) = n(n-1)/2 \in \Theta(n^2)$
  - It takes $\Theta(n^2)$ time even for sorted input
  - For data in reverse order, the number of swaps is also $\Theta(n^2)$

Bubble sort: some tips
Decrease the number of swaps

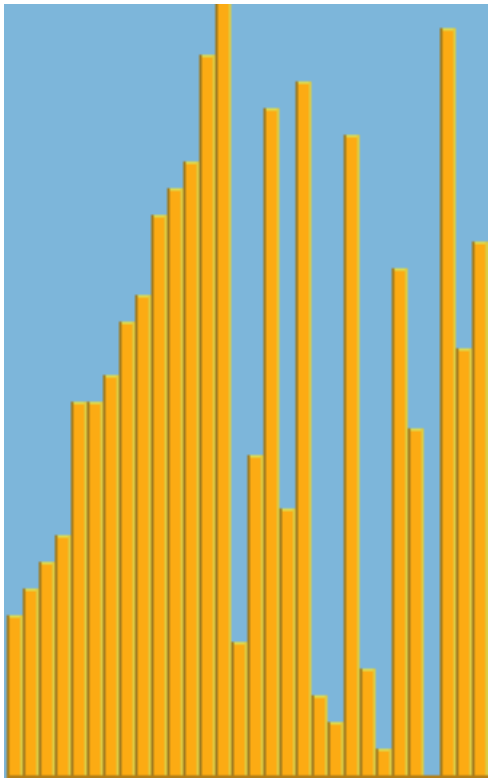
Q. Can we decrease the number of swaps to n?

```
for(k=1; k<n; k=k+1)
   for(i=0; i<n-k; i=i+1)
      if(data[i] > data[i+1])
         swap(&data[i], &data[i+1]);
```

A. Swap the rightmost one and the maximum one

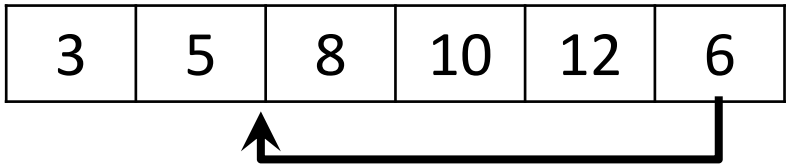
```
for(k=n-1; k>0; k=k-1){
  m=0;
  for(i=1; i<=k; i=i+1)
    if(data[i] > data[m]) m=i;
  swap(&data[k], &data[m]);
}
```

INSERTION SORT

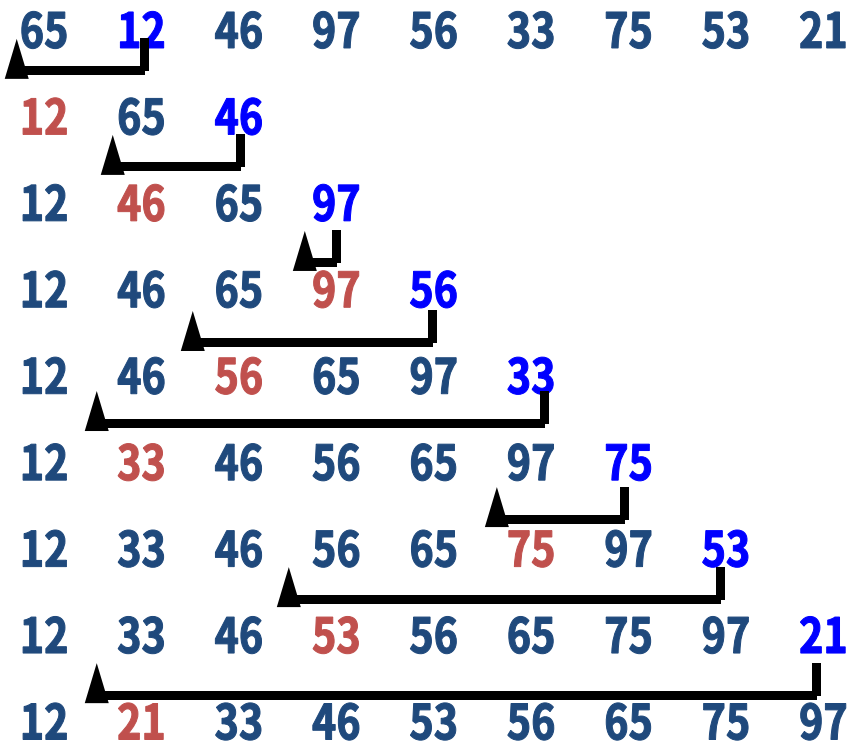


Insertion sort

- Sorted elements grows one by one



We need to shift the elements larger than the current item




```
for(i=1; i<n; i=i+1){
 x = data[i]; j=i;
 while(data[j-1]>x && j>0){
   data[j] = data[j-1];
   j=j-1;
 }
 data[j] = x;
}
```

Q. Why we do not use binary search for finding the point?

Insertion sort: time complexity

- Best case: $\Theta(n)$
  - When input data were already sorted
- Worst case: $\Theta(n^2)$
  - When data are in reverse order
  - Move all items for each item
- On average: $\Theta(n^2)$
  - When the data is the k-th one among m sorted data; In this case, we need k comparisons

Short exercise

In page 2, we consider the following case;

- String data: lexicographical ordering
  e.g., aaa, aab, aba, abb, baa, bab, bbc, bcb

For any two binary strings s=s[1]…s[n] and t=t[1]…t[m], describe exact condition if and only if s<t

(More exercise; can you make a dictionary that has all binary strings in your lexicographical ordering, and any finite length word has finite index? How can you avoid the potential problem?)