C Version

I111E Algorithms & Data Structures
8. Sorting(1):
Bubble, insertion, and heap Sort

School of Information Science

Ryuhei Uehara & Giovanni Viglietta

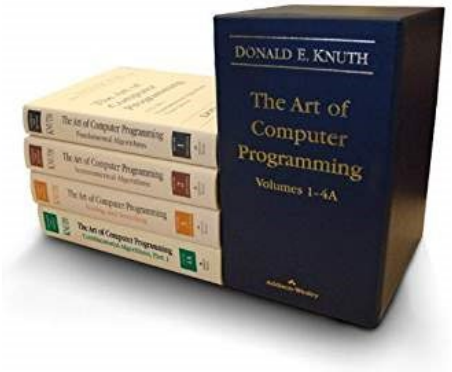
uehara@jaist.ac.jp & johnny@jaist.ac.jp

2019-11-13

All materials are available at

http://www.jaist.ac.jp/~uehara/couse/2019/i111e

Sorting

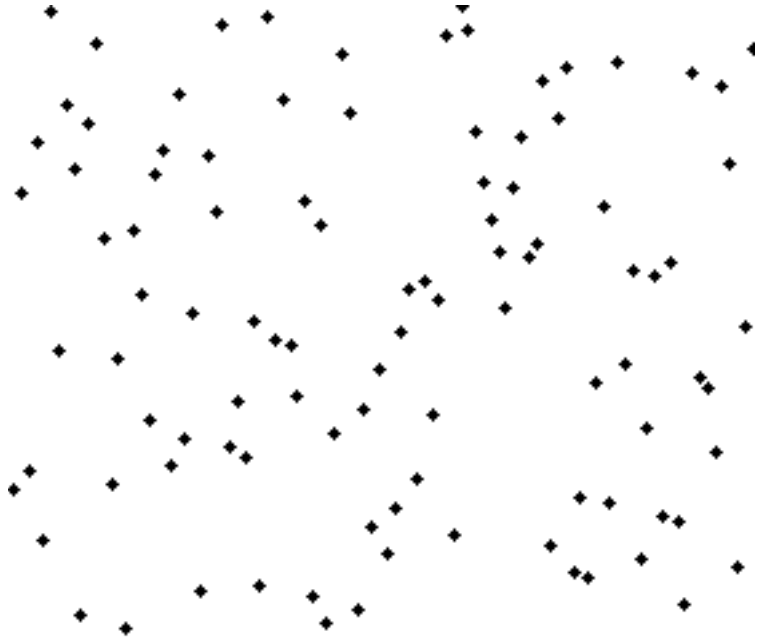


- Make given set of data in order
  - Numerical data: increasing/decreasing order

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…

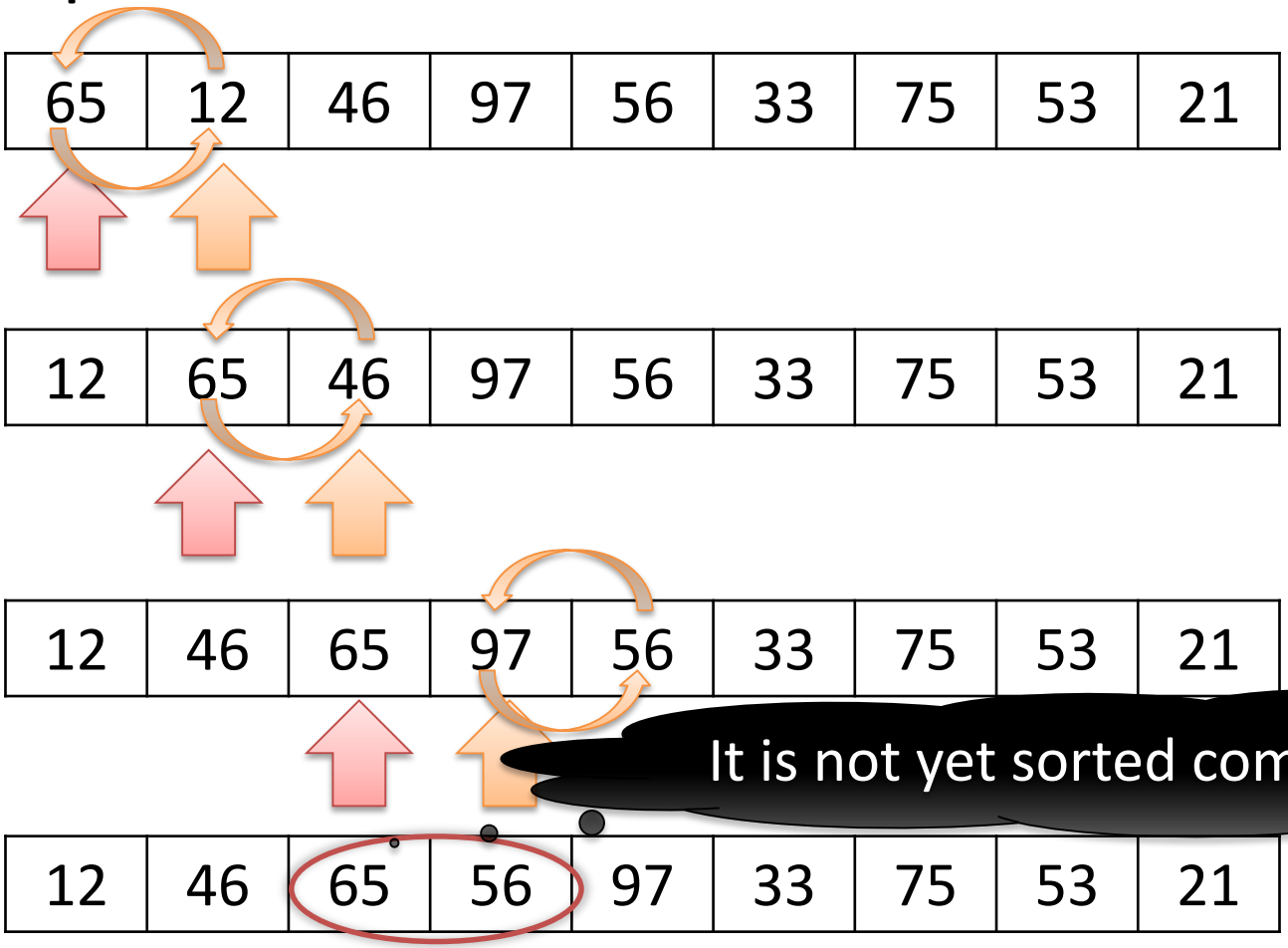
See the volume 3 of TAOCP by D. Knuth for more algorithms…



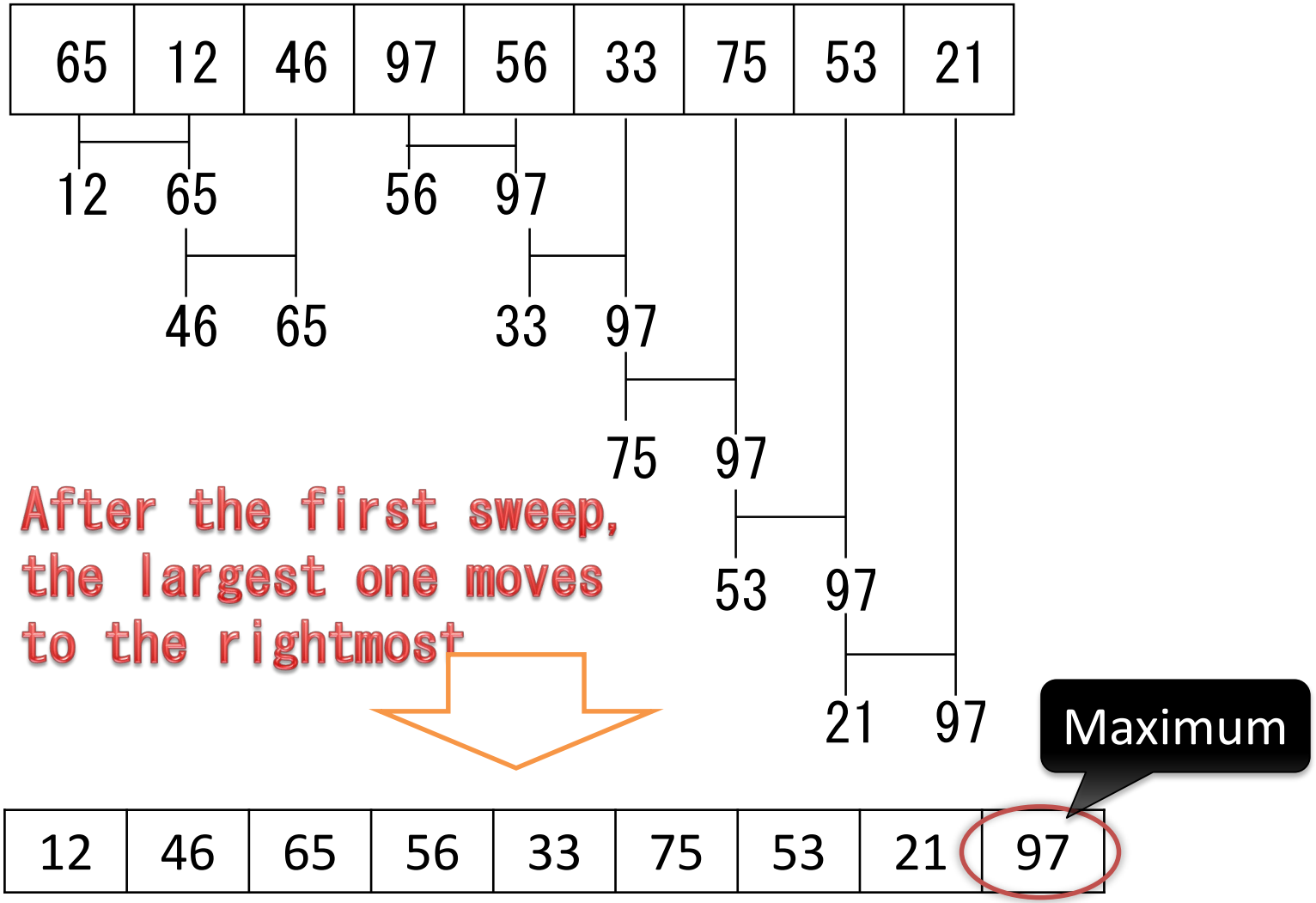
BUBBLE SORT

Bubble sort

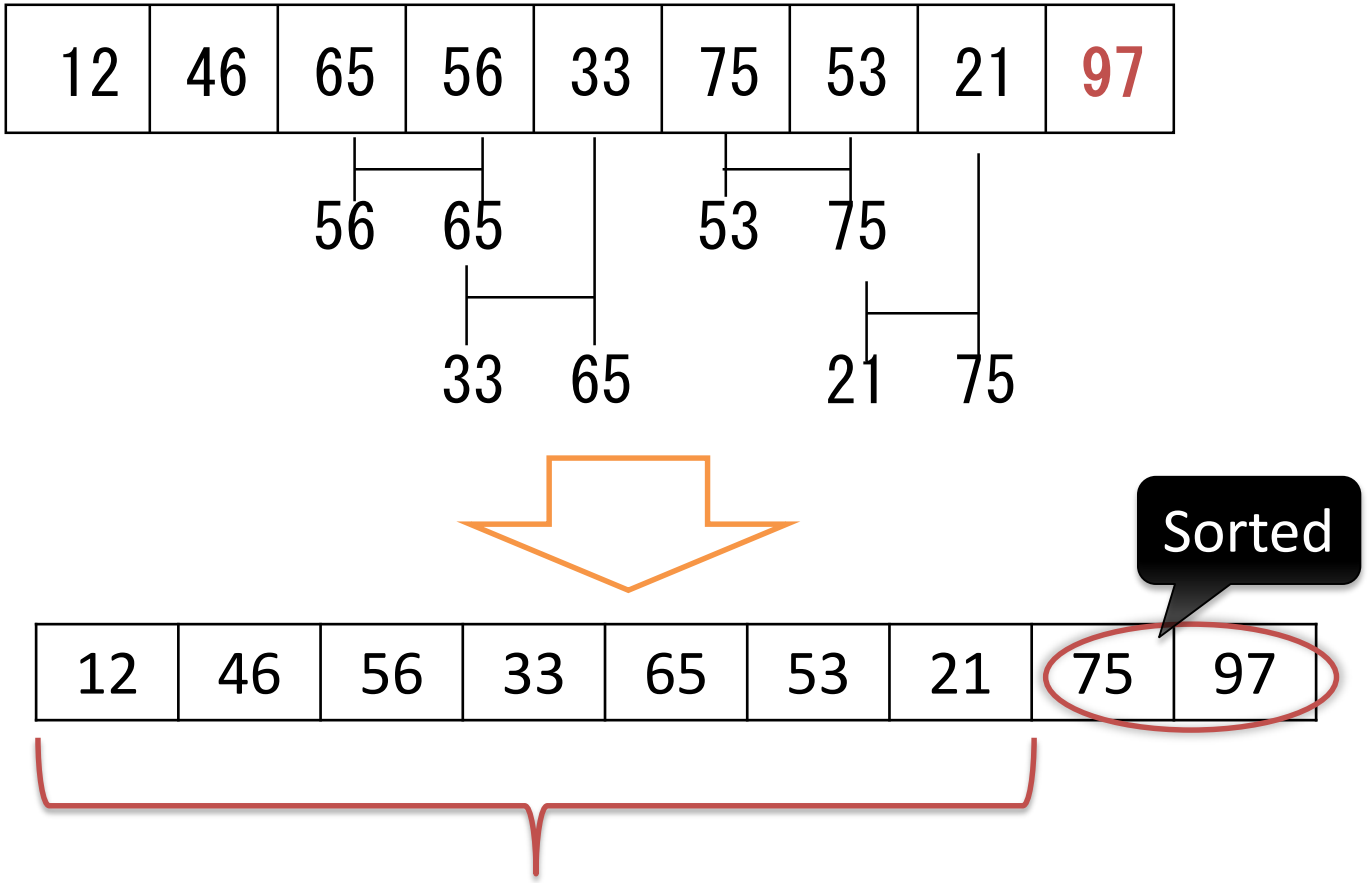
- From left to right, if they are not in order, swap them



Bubble sort: 1st sweep



Bubble sort: 2nd sweep

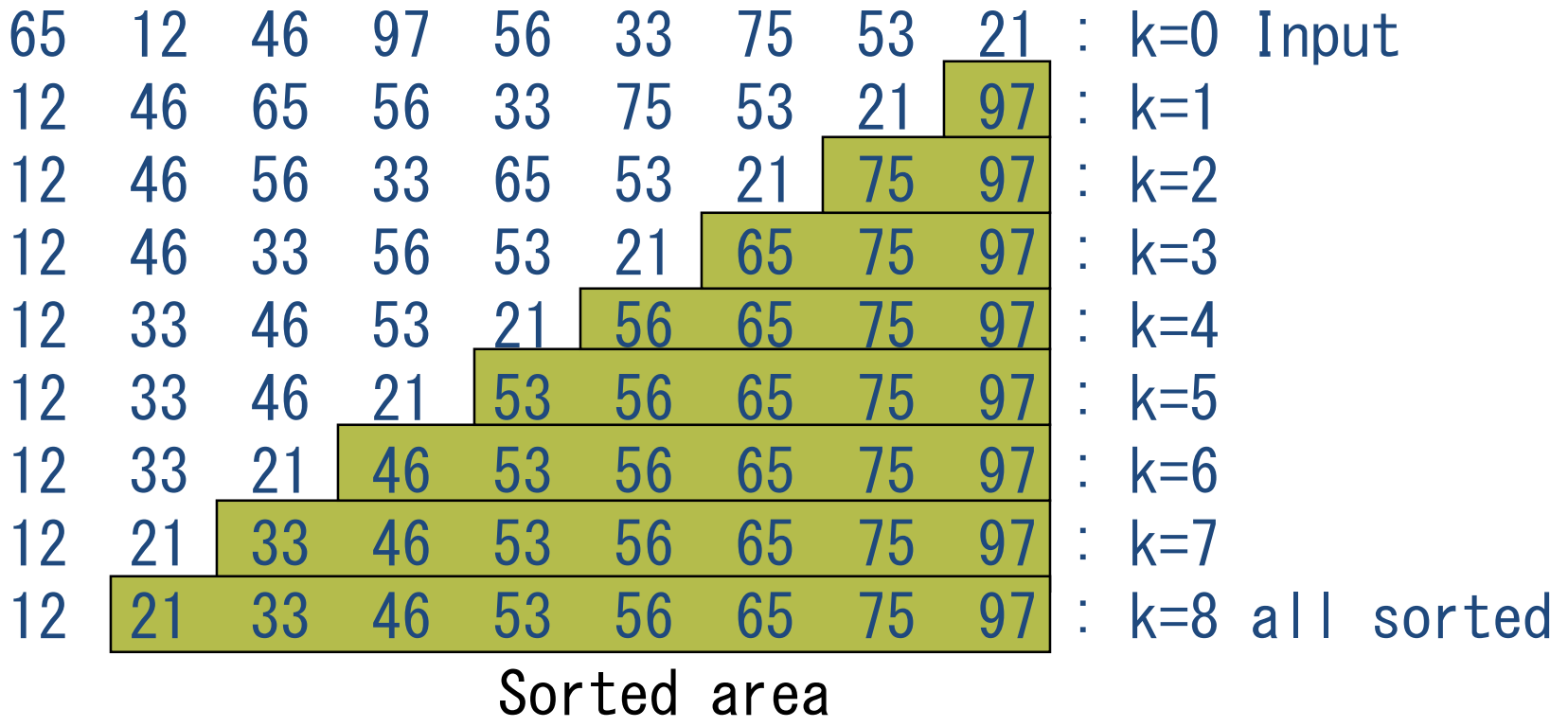


They are not sorted yet ➔ next sweep

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]);
```

※Use swap!

- # 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 to compare
  - For data in reverse order, the number of swaps is also $\Theta(n^2)$

Example: Implementation of Bubble Sort

```
public class i111_09_p9 {
    public static void Main(){
        int[] data = new int[]{65,12,46,97,56,33,75,53,21};
        print(data);

        int n = data.Length;
        for (int k=1; k<n; k++) {
            for (int i=0; i<n-k; i++) {
                if (data[i]>data[i+1]) {
                    int t=data[i]; data[i]=data[i+1]; data[i+1]=t;
                }
            }
            print(data);
        }
    }

    public static void print(int[] ar) {
        for (int i=0; i<ar.Length; i++) System.Console.Write(ar[i]+" ");
        System.Console.WriteLine();
    }
}
```

Body

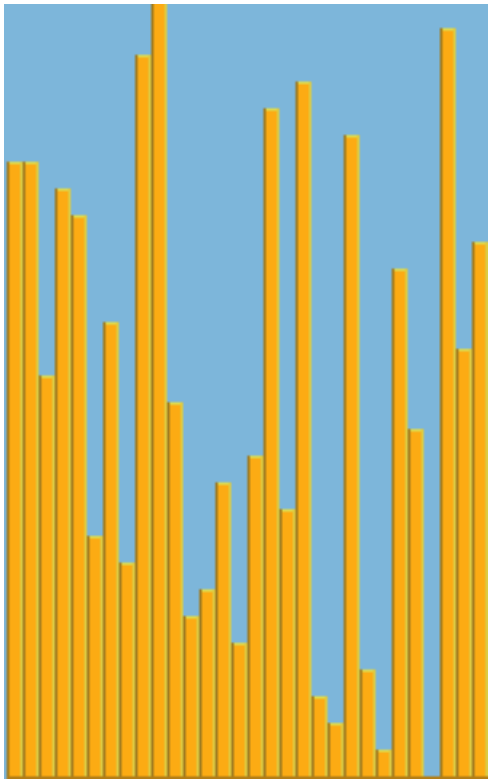
Initialize data

swap

Output

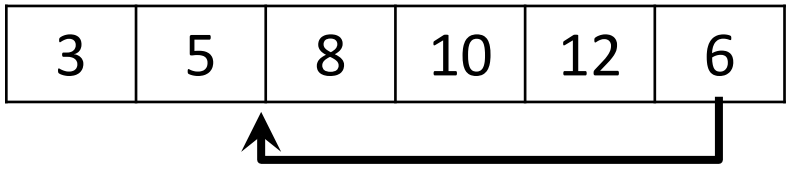
←※

INSERTION SORT

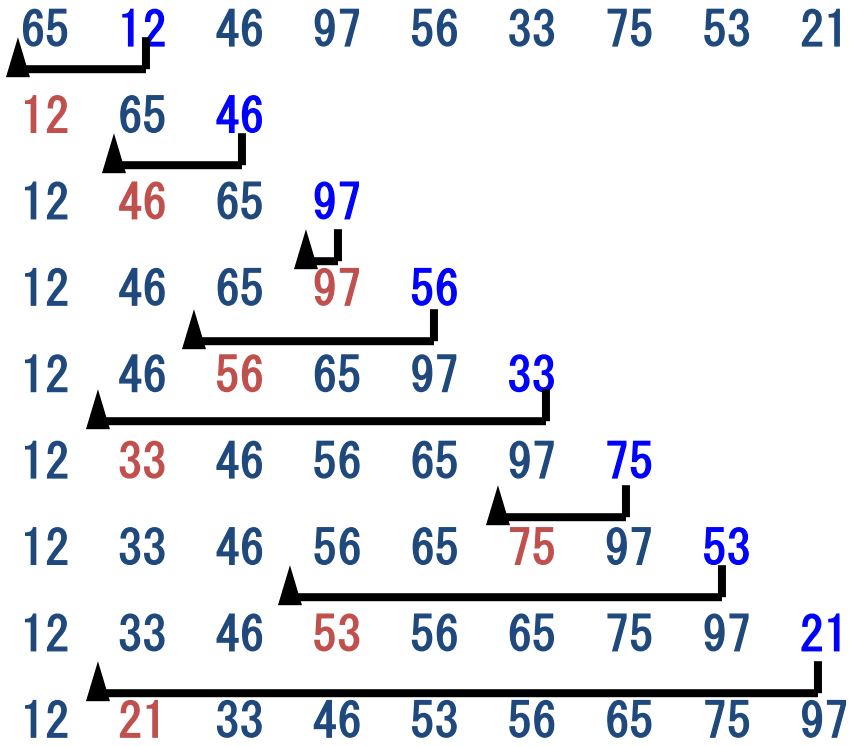


Insertion sort

- 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;
}
```

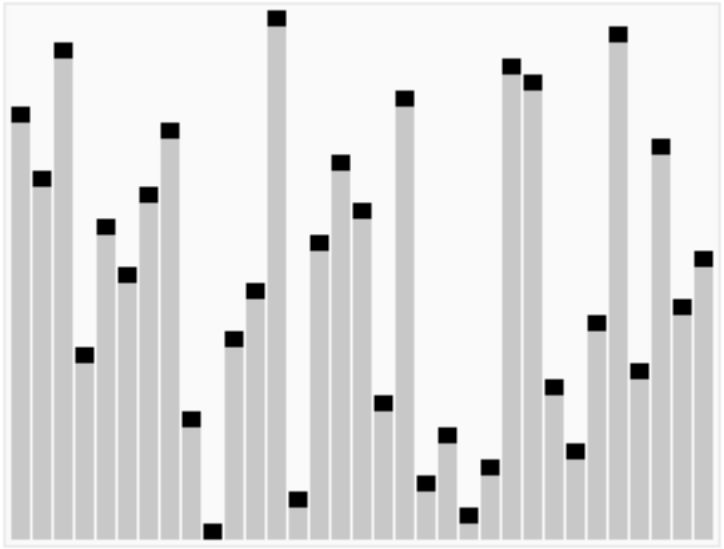
Until smaller than x

Move larger one than x to right

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



HEAP SORT

Heap sort

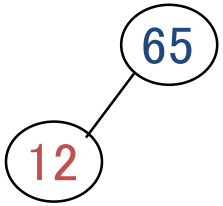
- Data structure heap
  - Insertion of data: $\Theta(\log n)$ time
  - Take the maximum element: $\Theta(\log n)$ time
- How to sort by heap
  - Step 1: Put n elements into heap
  - Step 2: Repeat to take the maximum element from heap, and copy it to the rightmost element
- Computational Complexity:
  - Both of steps 1 and 2 take $\Theta(n \log n)$ time.

Example of heap sort @Step 1

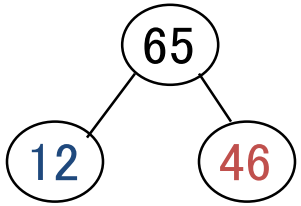
Data = 65 12 46 97 56 33 75 53 21

(1)add 65

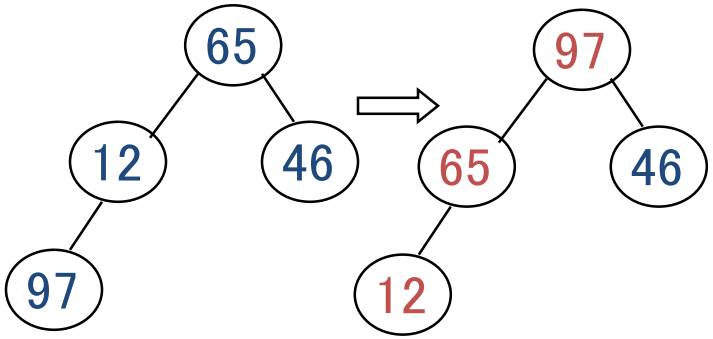
(2)add 12



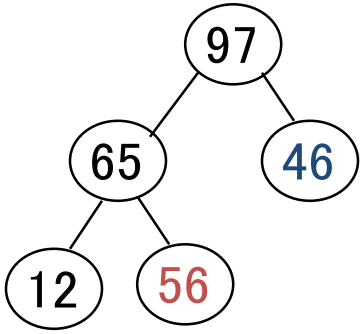
(3)add 46



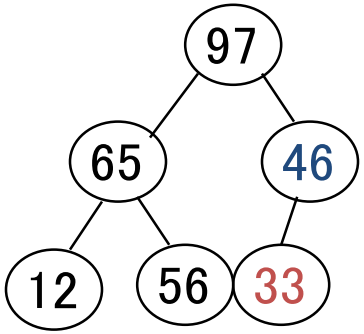
(4)add 97



(5)add 56



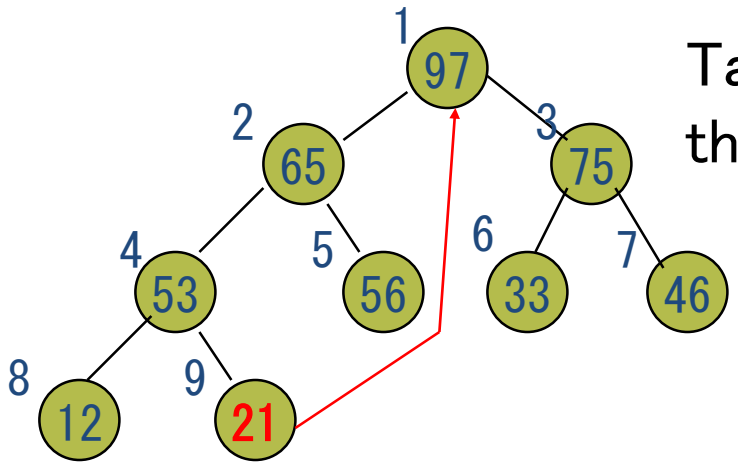
(6)add 33



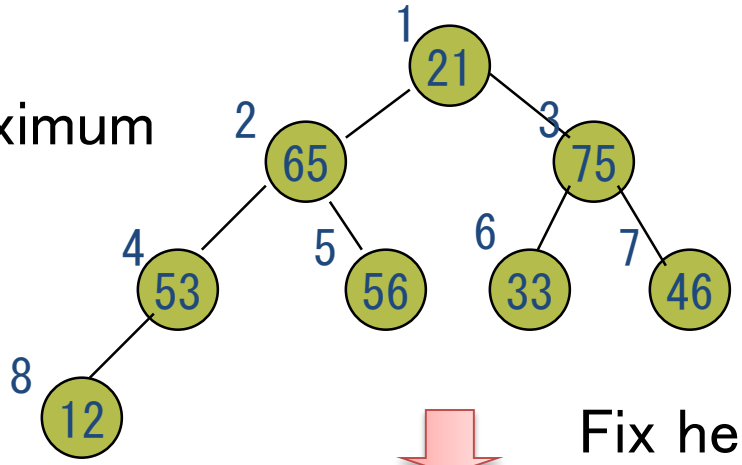
… in the same way, we can add data to heap one by one:

1	2	3	4	5	6	7	8	9
97	65	75	53	56	33	46	12	21

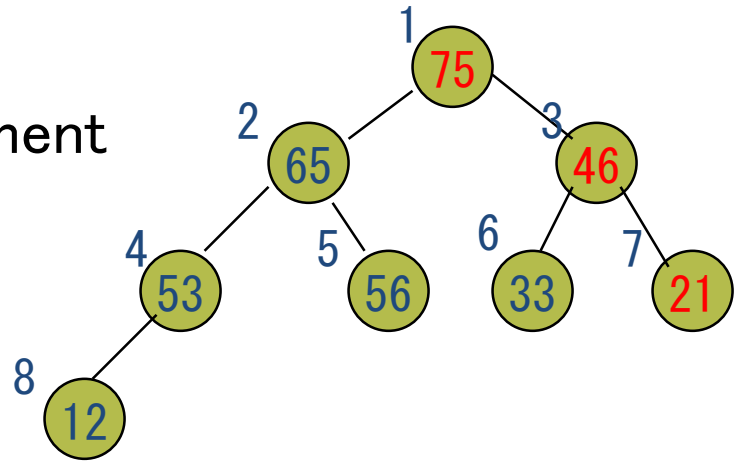
Example of heap sort @Step 2



Take
the maximum



Fix heap



Copy to the
rightmost element

75	65	46	53	56	33	21	12	97

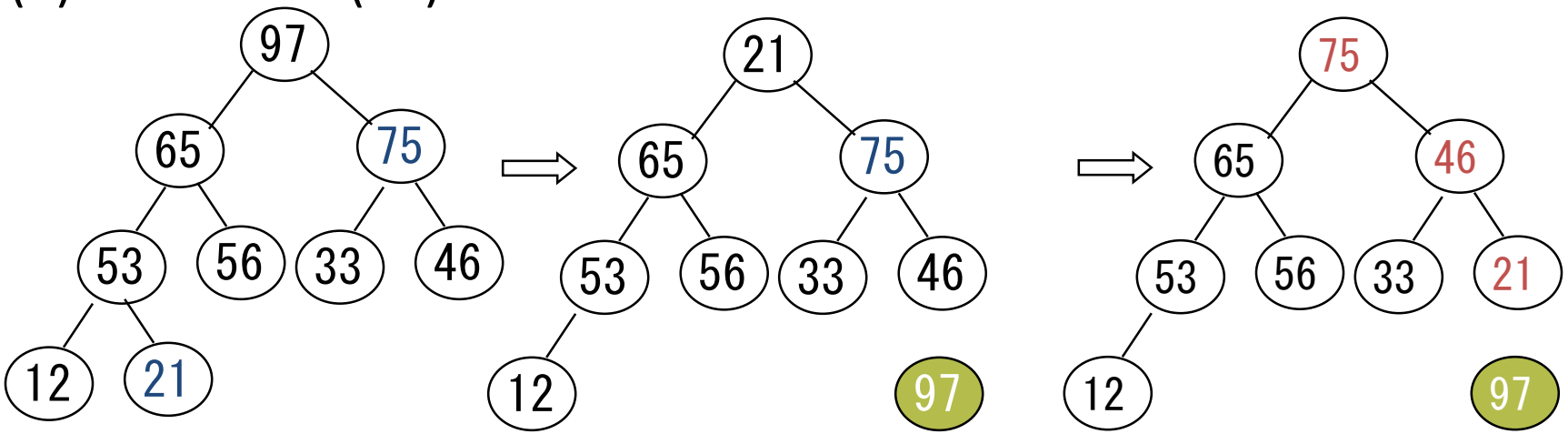
Note: We can sort them “in place”.

75	65	46	53	56	33	21	12	

Example of heap sort @Step 2

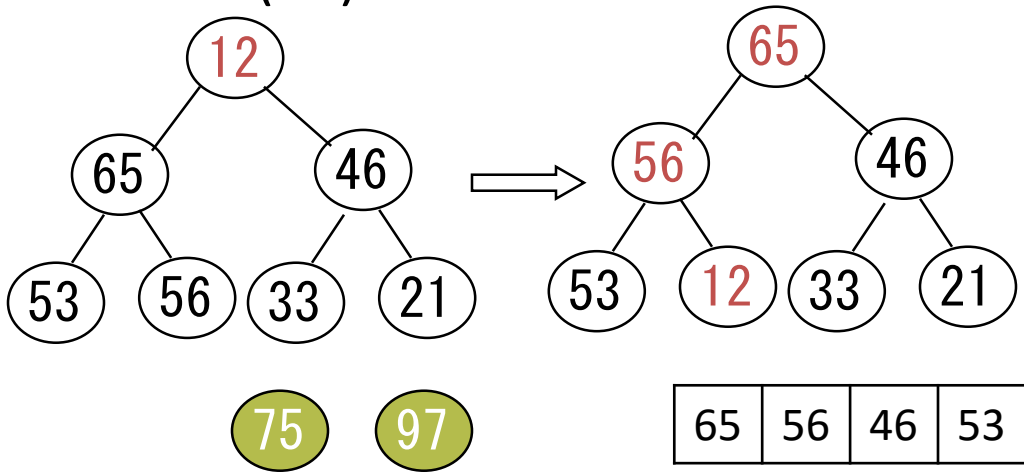
array = | 97 | 65 | 75 | 53 | 56 | 33 | 46 | 12 | 21 | :

(1)delete max (97)



75	65	46	53	56	33	21	12	97

(2) delete max (75)



65	56	46	53	12	33	21	75	97

Example: Implementation of heap

(We use an extra array and copy data from smallest one)

```
public class i111_09_p23 {
    public static void Main(){
        int[] data =
            new int[]{65,12,46,97,56,33,75,53,21};
        print(data);

        int n = data.Length;
        HeapArray heap = new HeapArray(n);
        for (int i=0; i<n; i++) heap.push(data[i]);
        for (int i=0; i<n; i++) data[i] = heap.pop();
        print(data);
    }

    public static void print(int[] ar) {
        for (int i=0; i<ar.Length; i++)
            System.Console.Write(ar[i]+" ");
        System.Console.WriteLine();
    }
}
```

Body

Put them all, and take them all

Implementation of HeapArray

In this implementation, parent is smaller than children

```
class HeapArray {
    private int[] heap;
    private int n;
    public HeapArray(int size) {
        heap = new int[size];
        n = 0;
    }
    public void push(int x){
        heap[n] = x;
        int child = n;
        int parent = (n-1) /2;
        while (child != 0 && x<heap[parent]) {
            heap[child] = heap[parent];
            child = parent;
            parent = (child-1)/2;
        }
        heap[child] = x;
        n++;
    }
    public int pop() {
        int minValue = heap[0];
        heap[0] = heap[n-1];
        n--;
        int parent = 0;
        int child = parent*2+1;
        while (child <n) {
            if (child+1<n && heap[child]>heap[child+1]) child++;
            if (heap[parent]<=heap[child]) break;
            int t=heap[child]; heap[child]=heap[parent]; heap[parent]=t;
            parent = child;
            child = parent*2+1;
        }
        return minValue;
    }
}
```

←※

←※