

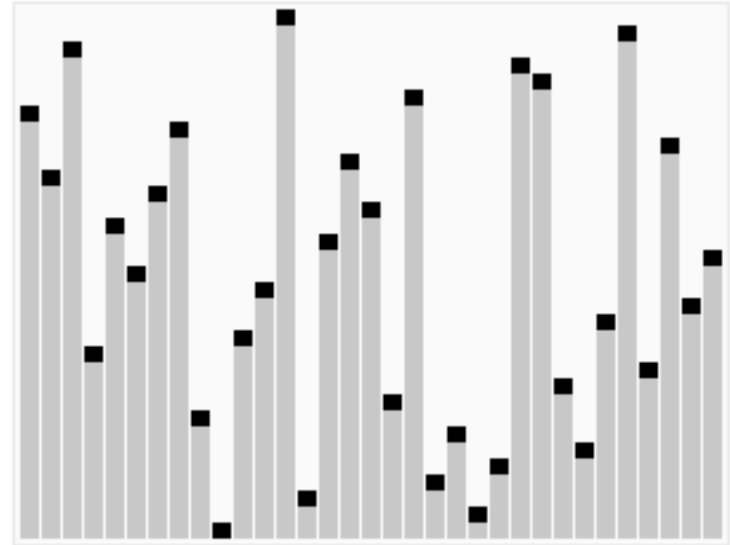
Introduction to Algorithms and Data Structures

Lecture 11: Sorting (2) Heap sort and Merge sort

Professor Ryuhei Uehara,
School of Information Science, JAIST, Japan.

uehara@jaist.ac.jp

<http://www.jaist.ac.jp/~uehara>



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.

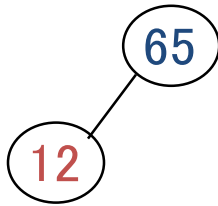
Example of heap sort @Step 1

Data = 65 12 46 97 56 33 75 53 21

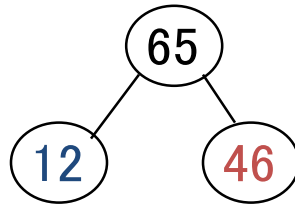
(1)add 65



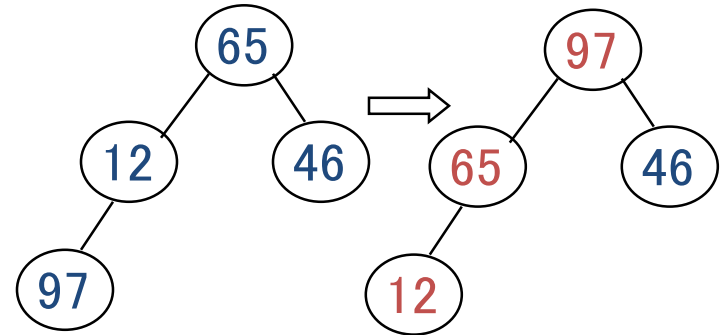
(2)add 12



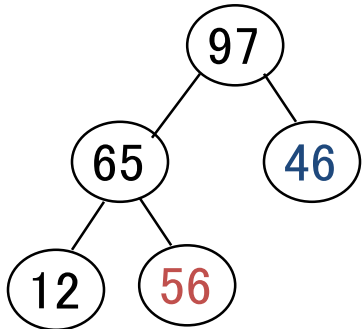
(3)add 46



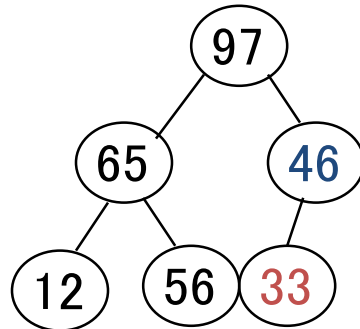
(4)add 97



(5)add 56



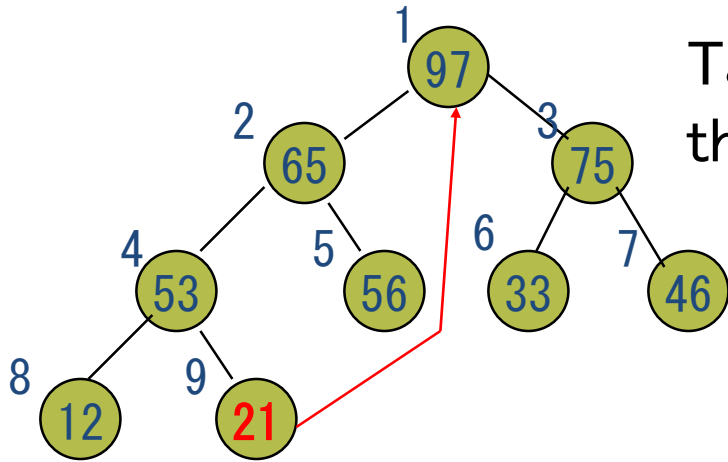
(6)add 33



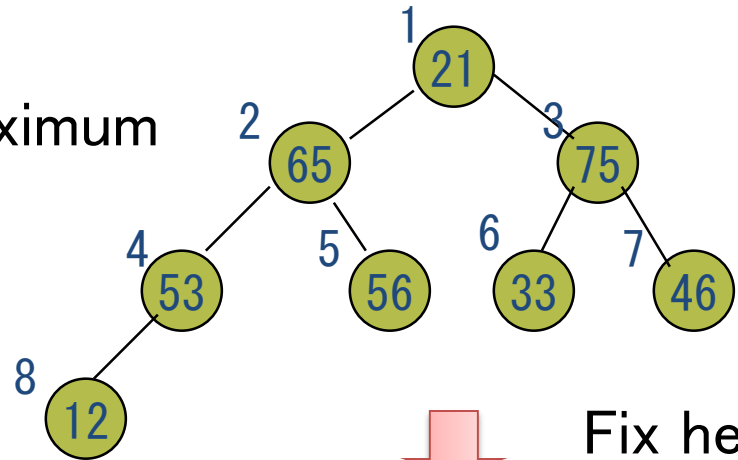
... in the same way, we can add data to heap one by one:

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

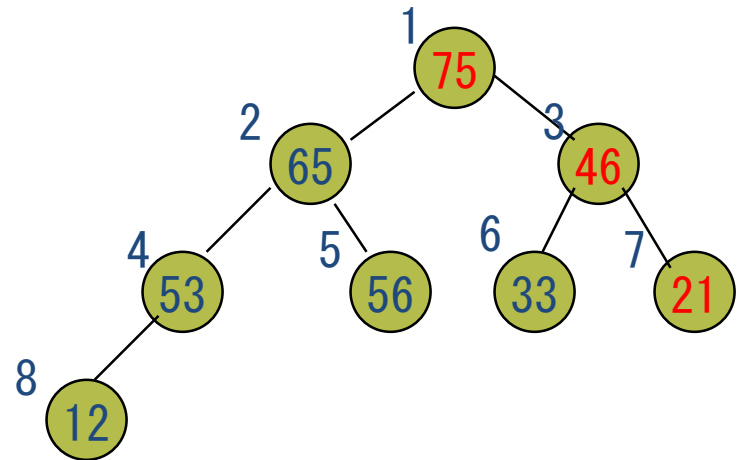
Example of heap sort @Step 2



Take
the maximum



Fix heap



75	65	46	53	56	33	21	12	97
----	----	----	----	----	----	----	----	----



Copy to the
rightmost
element

75	65	46	53	56	33	21	12	
----	----	----	----	----	----	----	----	--

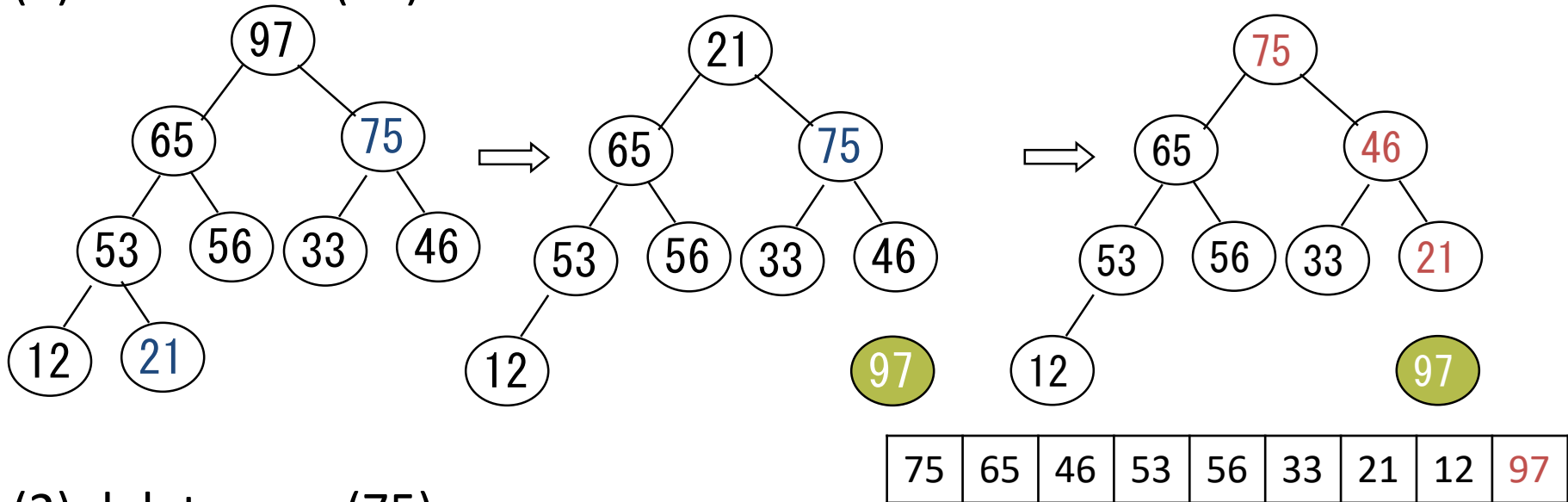
Example of heap sort @Step 2

array =

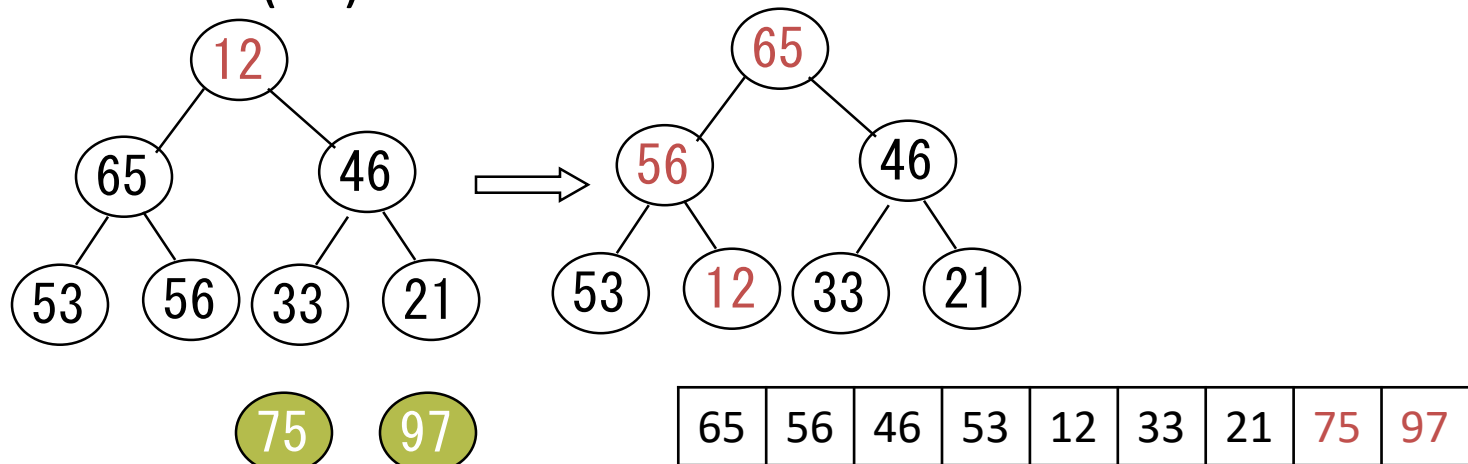
97	65	75	53	56	33	46	12	21
----	----	----	----	----	----	----	----	----

 :

(1) delete max (97)



(2) delete max (75)

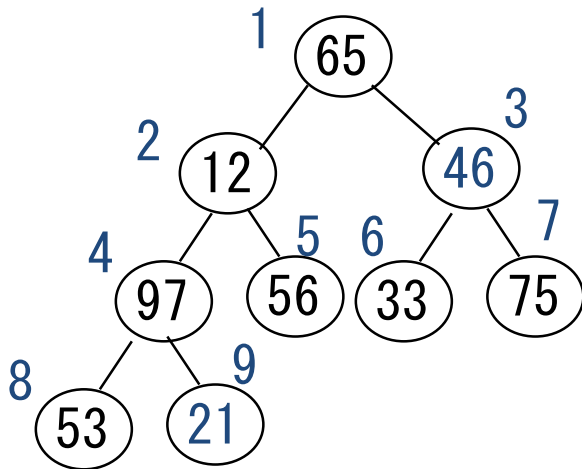


(Bit) improvement of heap sort

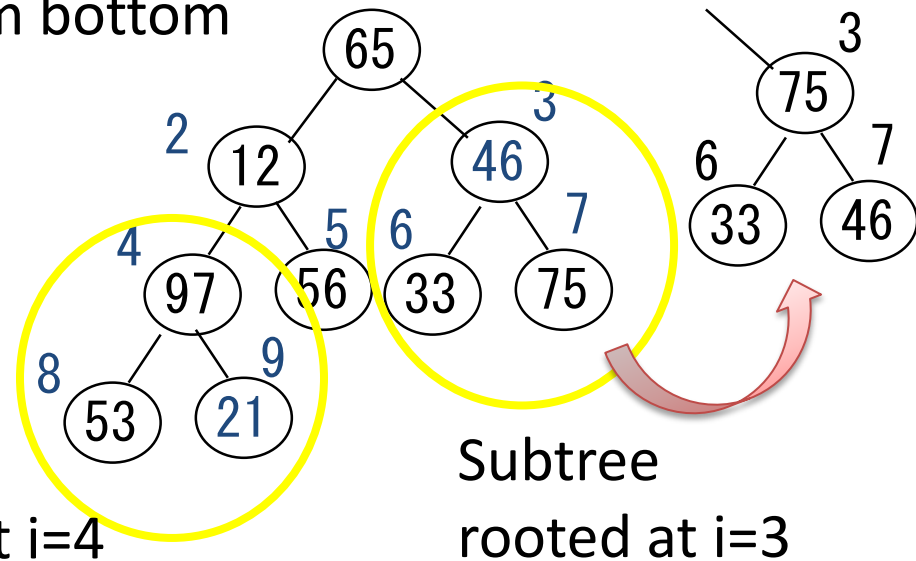
- We can make step 1 to run in $\Theta(n)$ time
 - Add all items into the array first
 - From bottom to top, exchange the parent/child

(1) Store data

(2) Exchange data in each parent/child from bottom



Subtree
rooted at $i=4$



Subtree
rooted at $i=3$



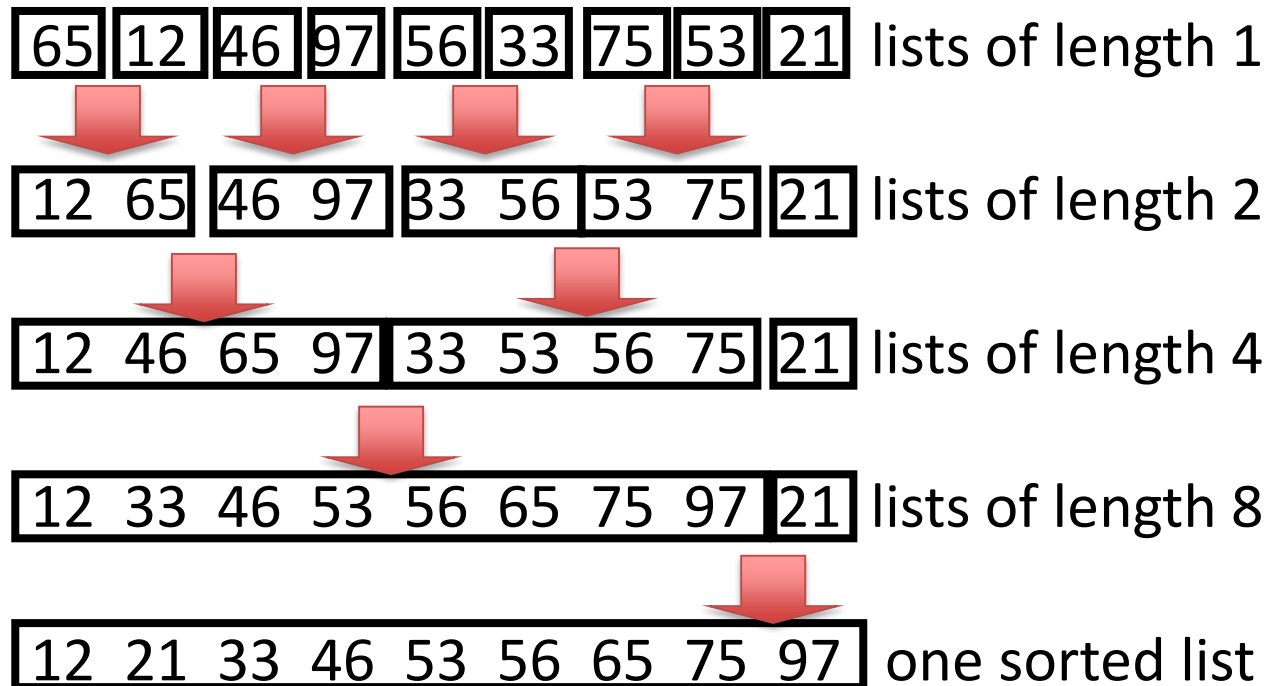
John von Neumann
1903–1957



MERGE SORT

Merge sort

- It repeats to merge two sorted lists into one (sorted) list



- First, it repeats to divide until all lists have length 1, and next, it merges each two of them.

Implementation of merge sort: Typical recursive calls

- The interval that will be sorted: [left, right]
- Find center $mid = (left + right)/2$



- $[left, right] \rightarrow [left, mid], [mid+1, right]$
- Perform merge sort for each of them, and merge these sorted lists into one sorted list.

Outline of merge sort

```
MergeSort(int left, int right){
    int mid;
    if(interval [left,right] is short)
        (sort by any other simple sort algorithm);
    else{
        mid = (left+right)/2;
        MergeSort(left, mid);
        MergeSort(mid+1, right);
        Merge [left, mid] and [mid+1, right];
    }
}
```

We can merge two lists of length p and q in $O(p + q)$ time.

Merge sort: the merge process

To merge [left, mid] and [mid+1, right]:

```
i=left; j=mid+1; k=left;
while(i<=mid && j<=right)
  if(a[i] <= a[j]) {
    b[k]=a[i]; k++; i++;
  } else {
    b[k]=a[j]; k++; j++;
  }
while(j<=right){ b[k]=a[j]; k++; j++; }
while(i<=mid){ b[k]=a[i]; k++; i++; }
for(i=left; i<=right; i++) a[i]=b[i];
```

Temporarily, it stores items in a[] to b[] to merge.

Write back b[] to a[]

$O(p + q)$
time

Merge sort: Time complexity

- $T(n)$: Time for merge sort on n data
 - $T(n) = 2T(n/2) + \text{“time to merge”}$
 $= 2T(n/2) + cn + d$ (c, d : some positive constant)

- To simplify, letting $n = 2^k$ for integer k ,

$$\begin{aligned}T(2^k) &= 2T(2^{k-1}) + c2^k + d \\&= 2(2T(2^{k-2}) + c2^{k-1} + d) + c2^k + d \\&= 2^2T(2^{k-2}) + 2c2^k + (1 + 2)d \\&= 2^2(2T(2^{k-3}) + c2^{k-2} + d) + 2c2^k + (1 + 2)d \\&= 2^3T(2^{k-3}) + 3c2^k + (1 + 2 + 4)d\end{aligned}$$

⋮

$$\begin{aligned}&= 2^i T(2^{k-i}) + ic2^k + (1 + 2 + \dots + 2^{i-1})d \\&= 2^k T(2^0) + kc2^k + (1 + 2 + \dots + 2^{k-1})d \\&= bn + cn \log n + (n - 1)d \in O(n \log n)\end{aligned}$$

Merge sort: Space complexity

- It is easy to implement by using two arrays `a[]` and `b[]`.
 - Thus space complexity is $\Theta(n)$, or we need n extra array for `b[]`.
 - It seems to be difficult to remove this “extra” space.
 - On the other hand, we can omit “Write back `b[]` to `a[]`” (in the 2 previous slides) when we use `a[]` and `b[]` alternately.

Maybe this “extra” space is the reason why merge sort is not used so often...


Monotone sequence merge sort

- Bit improved merge sort from the practical viewpoint.
- It first divides input into monotone sequences and merge them. (Original merge sort does not check the input)

Example: For 65, 12, 46, 97, 56, 33, 75, 53, 21;


65	12	46	97	56	33	75	53	21
----	----	----	----	----	----	----	----	----

 Divide into monotone sequences



12	46	65	97	21	33	53	56	75
----	----	----	----	----	----	----	----	----

 Merge neighbors



12	21	33	46	53	56	65	75	97
----	----	----	----	----	----	----	----	----

 Sorted!

Monotone sequence merge sort:

Time complexity

- We can merge in $O(p+q)$ time to merge two sequences of length p and q
- After merging, the number of sequences becomes in half.
 - When the number of monotone sequences is h , the number of recursion is $\log_2 h$ times.
- One recursion takes $O(n)$ time
 - $O(n \log h)$ time in total.
- When data is already sorted: $h = 1 \rightarrow O(n)$ time
- The maximum number of monotone sequences is $n/2$
 - $O(n \log n)$ time in total.