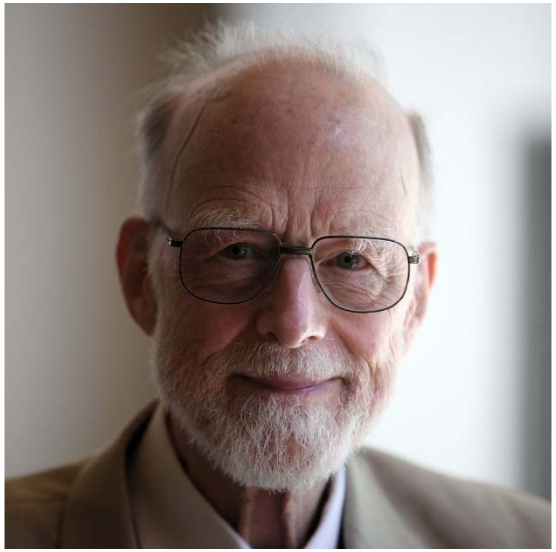
Introduction to Algorithms and Data Structures

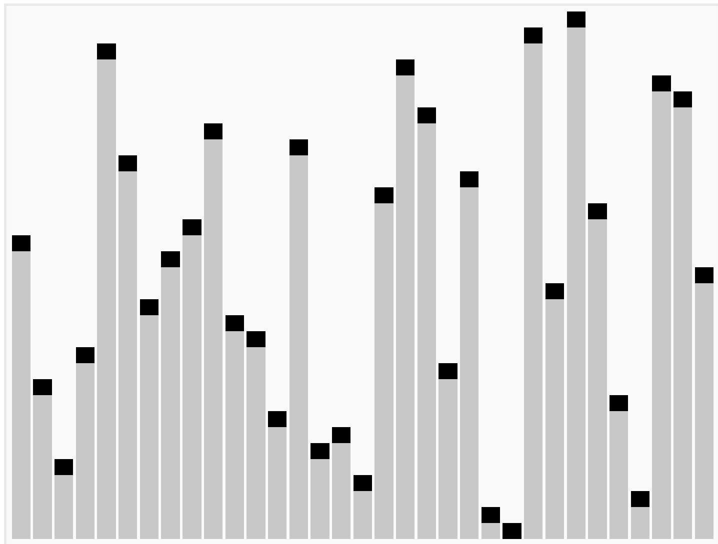
Lecture 12: Sorting (3) Quick sort, complexity of sort algorithms, and counting sort

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



Tony Hoare
1934–

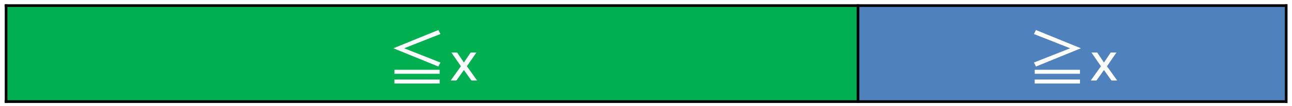
QUICK SORT



C.A.R. Hoare, "Algorithm 64: Quicksort".
Communications of the ACM 4 (7): 321 (1961)

Quick sort

- Main property: On average, the fastest sort!
- Outline of quick sort:
  - Step 1: Choose an element x (which is called pivot)
  - Step 2: Move all elements ≦ x to left
    Move all elements ≧ x to right



  - Step 3: Sort left and right sequences <u>independently</u> and <u>recursively</u>
    - (When sequence is short enough, sort by any simple sorting)

Quick sort: Example
Step 1. Choose an element x

- Sort the following array by quick sort:

65	12	46	97	56	33	75	53	21

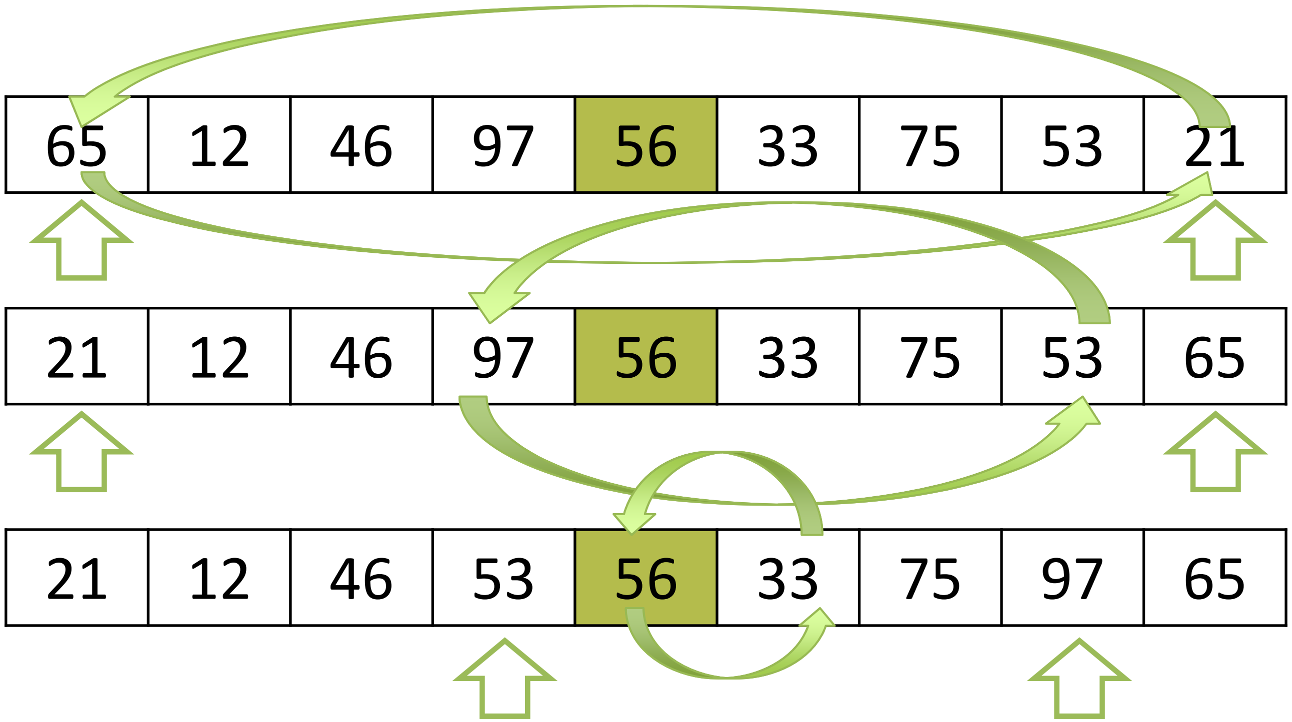
- Choose x=56, for example;

65	12	46	97	56	33	75	53	21

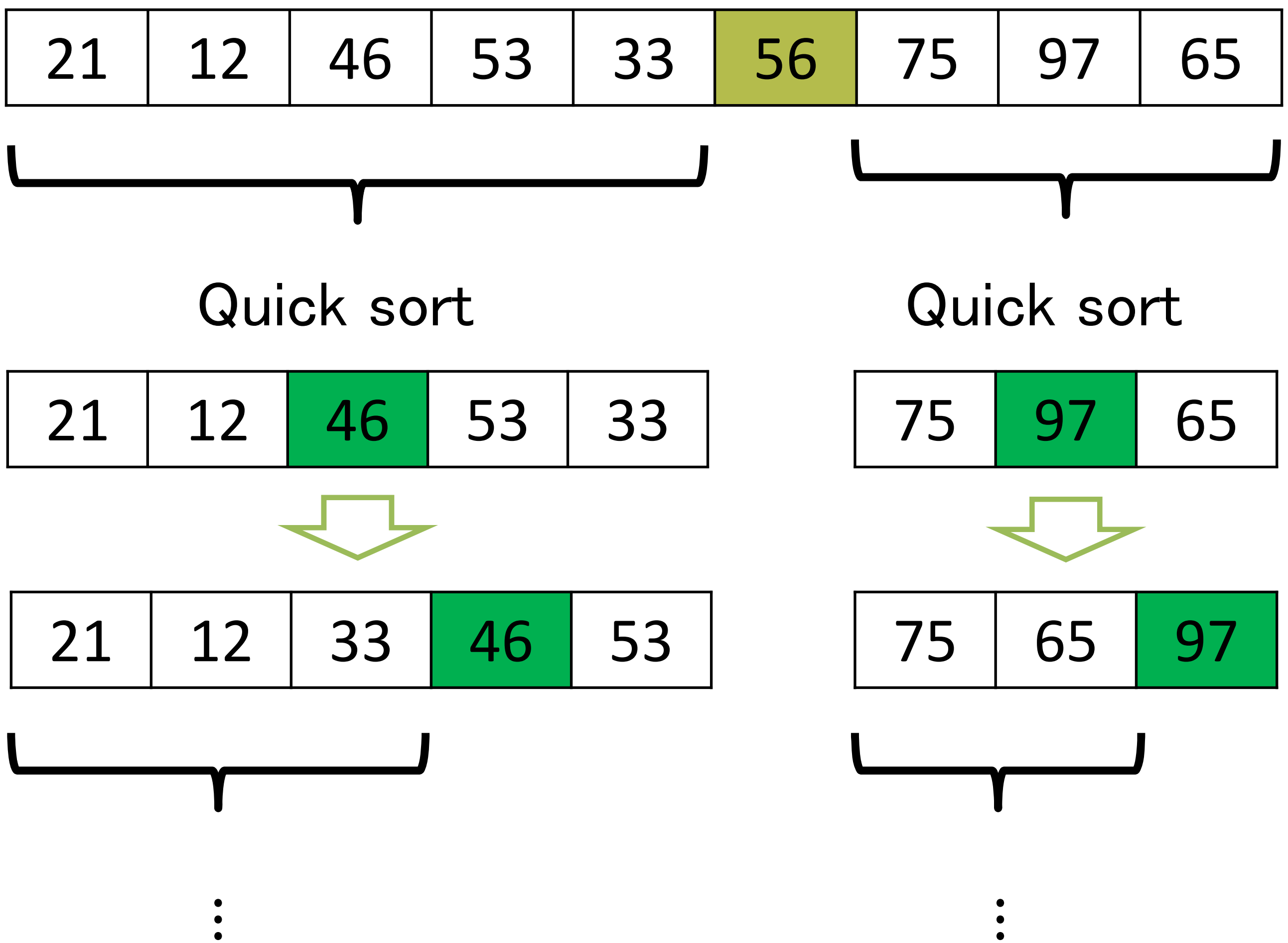
Quick sort: Example
Step 2. Move element w.r.t x:

- 

- Start from [l, r] = [0,n-1], move l and r,
  Swap a[l] and a[r] when a[l] >= x && a[r] < x



Quick sort: Example
Step 3. Sort left and right sequences recursively



Quick sort: Program

```
qsort(int a[], int left, int right){
  int i, j, x;
  if(right <= left) return;
  i = left; j = right; x = a[(i+j)/2];
  while(i<=j){
    while(a[i]<x) i=i+1;
      while(a[j]>x) j=j-1;
       if(i<=j){
      swap(&a[i], &a[j]); i=i+1; j=j-1;
    }
  }
  qsort(a, left, j); qsort(a, i, right);
}
```

Note: In MIT textbook, there is another implementation.

Quick sort: Time complexity (1/3) Worst case

- When the pivot x is the maximum or minimum element, we divide
  length n → length 1 + length n-1
- This repeats until the longer one becomes 2
- The number of comparisons; $\sum_{k=2}^{n} k \in \Theta(n^2)$

Almost as same as the bubble sort…

Quick sort: Time complexity (2/3)
Average case

- Pick up x randomly from n elements.
- For each k, x is the k-th element in n elements with probability 1/n
- When x is the k-th element;
  length n → length k + length n-k

Quick sort: Time complexity (3/3) Average case

- When x is the k-th element;
  length n → length k + length n-k
- Total number $C(n)$ of comparisons

$$C(n) = \sum_{k=1}^{n} \frac{1}{n}(n + C(k) + C(n-k))$$

$$\approx n + \frac{2}{n}\sum_{k=1}^{n} C(k)$$

$$\Longrightarrow \quad C(n) = 1.36 n \log n + O(n)$$

Quick quiz

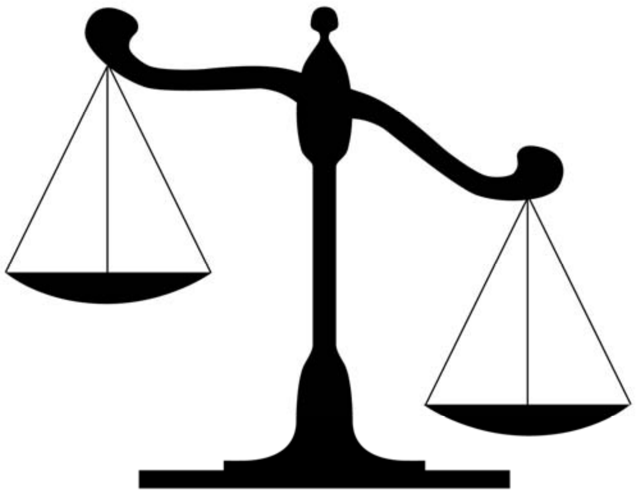
- For the qsort, construct a bad input that gives the worst case.

- When you fix the way of choice of pivot, there are some inputs that give the worst case. However, using randomization, we can avoid that scenario.

COMPUTATIONAL COMPLEXITY OF THE SORTING PROBLEM

Sort on Comparison model

- Sort on comparison model: Sorting algorithms that only use the “ordering” of data
  - It only uses the property of “a > b, a = b, or a < b”; in other words, the value of variable is not used.



Computational complexity of sort on comparison model

- Upper bound: $O(n \log n)$
  There exist sort algorithms that run in time proportional to $n \log n$ (e.g., merge sort, heap sort, …).
- Lower bound: $\Omega(n \log n)$
  For any comparison sort, there exists an input such that the algorithm runs in time proportional to $n \log n$.

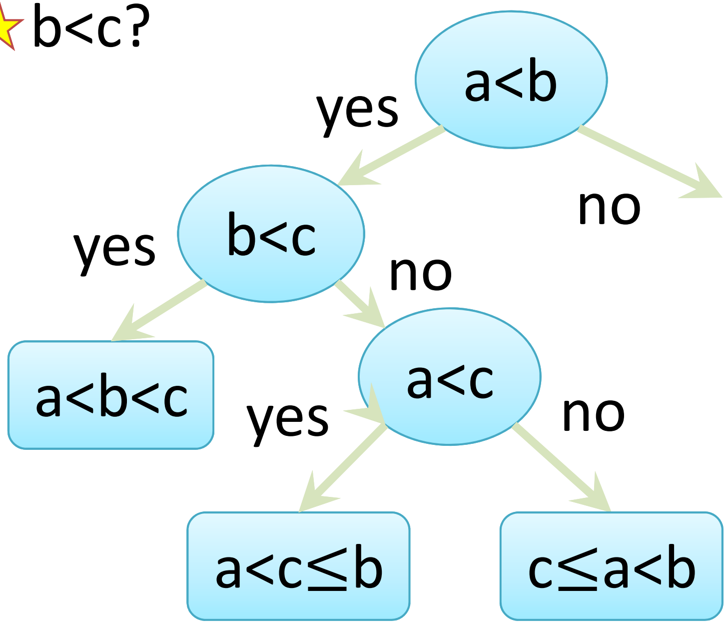
We consider the lower bound of comparison sorting.

Computational complexity of comparison sort: lower bound

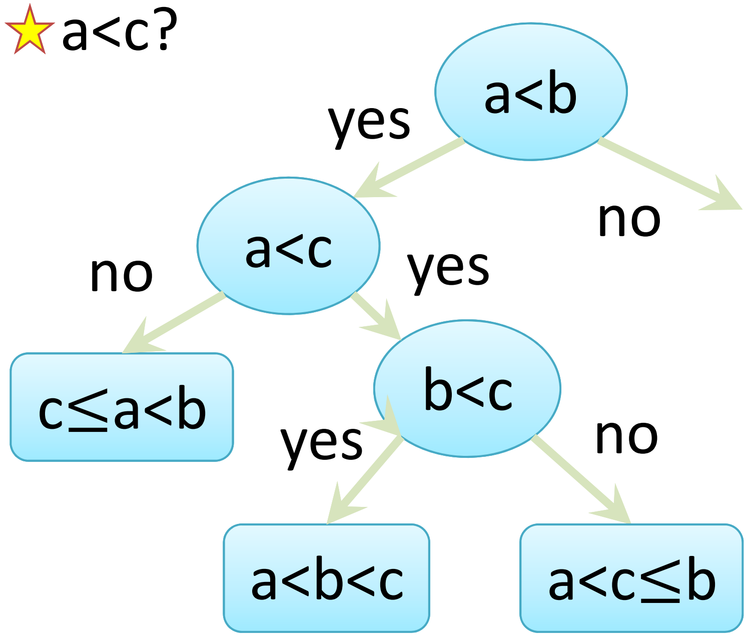
- Simple example; sort 3 data a, b, c:

  First, compare (a,b), (b,c), or (c, a). Without loss of generality, we assume that (a,b) is compared; then the next pair is (b,c) or (c,a):

☆b<c?



☆a<c?



Computational complexity of comparison sort: lower bound

- What we know from sorting of {a, b, c}:
  - For any input, we obtain the solution <u>at most </u>3 comparison operators.
  - There are some input that we have to compare at least 3 comparison operations.

  = maximum length of a path from root to a leaf is 3,

  which gives us the lower bound.

When we build a decision tree such that "the longest path from root to a leaf is shortest," that length of the longest path gives us a lower bound of sorting problem.

Computational complexity of comparison sort: lower bound

The case when *n* data are sorted

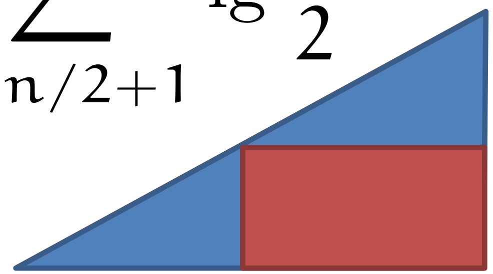
- Let k be the length of the longest path in an optimal decision tree T. Then,

  The number of leaves of T ≦ 2^k

- Since all possible permutations of *n* items should appear as leaves, $n! \leqq 2^k$
- By taking logarithm,

$$k = \lg 2^k \geq \lg n! = \sum_{i=1}^{n} \lg i \geq \sum_{i=n/2+1}^{n} \lg \frac{n}{2}$$

$$= \frac{n}{2} \lg \frac{n}{2} \in \Omega(n \log n)$$



Non-comparison sort: Counting sort

- We need some assumption:

  $data[i] \in \{1,\ldots,k\}$ for $1 \leqq i \leqq n$, $k \in O(n)$

  (For example, scores of many students)
- Using values of data, it sorts in $\Theta(n)$ time.

Counting sort

Input: data[i]∈{1,…,k} for 1≦i≦n, k∈O(n)

Idea: Decide the position of element x

– Count the number of element less than x

➔That number indicates the position of x

Example:

3	7	4	1	2	5

1	2	3	4	5	6	7
1	1	1	1	1	0	1

➔

1	2	3	4	5	6	7
0	1	2	3	4	5	5

⬇

1	2	3	4	5	7

Counting sort

Q. When array contains many data of same values?

A. Use 3 arrays a[], b[], c[] as follows;
(a[]: input, b[]: sorted data, c: counter)

- c[a[i]] counts the number of data equal to a[i]

- For each j with 0≦j≦k,
let c'[j] := c[0] + … + c[j-1] + c[j], then
c'[j] indicates the number of data whose value is less than j

- Copy a[i] to certain b[] according to the value of c'[]

Counting sort: program

```
CountingSort(a, b, k){
  for i=0 to k
     c[i] = 0;

  for j=0 to n-1
     c[ a[j] ] = c[ a[j] ] + 1;

  for i=1 to k
     c[i] = c[i] + c[i-1];

  for j=n-1 downto 0
    b[ c[a[j]]-1 ] = a[j];
    c[a[j]] = c[a[j]] - 1;
}
```

Initialize counter c[]

Count the number of the value in a[i]

Compute c’[] from c[] In an efficient way!

Copy a[] to b[]

Counting sort: Example
Sort integers (3,6,4,1,3,4,1,4)

- After (2);
  c[]=(0,2,0,2,3,0,1)
- After (3);
  c[]=(0,2,2,4,7,7,8)

a[7]=4 => b[c[4]-1] = b[6], c[4]=6
a[6]=1 => b[c[1]-1] = b[1], c[1]=1
a[5]=4 => b[c[4]-1] = b[5], c[4]=5
a[4]=3 => b[c[3]-1] = b[3], c[3]=3
a[3]=1 => b[c[1]-1] = b[0], c[1]=0
a[2]=4 => b[c[4]-1] = b[4], c[4]=4
a[1]=6 => b[c[6]-1] = b[7], c[6]=7
a[0]=3 => b[c[3]-1] = b[2], c[3]=2

```
CountingSort(a, b, k){
  for i=0 to k
     c[i] = 0;

(2)for j=0 to n-1
     c[ a[j] ] = c[ a[j] ] + 1;

(3)for i=1 to k
     c[i] = c[i] + c[i-1];

  for j=n-1 to downto 0
    b[ c[a[j]]-1 ] = a[j];
    c[a[j]] = c[a[j]] - 1;
}
```

Counti...

Sort is said to be “<u>stable</u>” when two variables of the same value in order after sorting.

- After...

  c[]=(0,2, ...,7,7,8)

a[7]=4 => b[c[4]-1] = b[6], c[4]=6
a[6]=1 => b[c[1]-1] = b[1], c[1]=1
a[5]=4 => b[c[4]-1] = b[5], c[4]=5
a[4]=3 => b[c[3]-1] = b[3], c[3]=3
a[3]=1 => b[c[1]-1] = b[0], c[1]=0
a[2]=4 => b[c[4]-1] = b[4], c[4]=4
a[1]=6 => b[c[6]-1] = b[7], c[6]=7
a[0]=3 => b[c[3]-1] = b[2], c[3]=2

```
      ...j] ] = c[ a[j] ] + 1;

(3)for i=1 to k
      c[i] = c[i] + c[i-1];

   for j=n-1 to downto 0
     b[ c[a[j]]-1 ] = a[j];
     c[a[j]] = c[a[j]] - 1;
 }
```

Short (and advanced) exercises

- Among sort algorithms; bubble sort, insertion sort, heap sort, merge sort, quick sort, counting sort,
  - Which are stable?
  - Which is not comparison sort?
  - Investigate more sort algorithms!

- Investigate "Harmonic number," which is defined by $H(n) = \sum_{i=1}^{n} \frac{1}{i}$
  (It appears in analysis of lower bound of comparison sort.)