Algorithm Design and Analysis

วิชาบังคับก่อน: 204251 และ 206281

ผู้สอน: ตอน 1 ผศ. เบญจมาศ ปัญญางาม
ตอน 2 ผศ. ดร. จักริน ชวชาติ

บทที่ 6
ต้นไม้การตัดสินใจและขอบเขตล่าง
(Decision Tree and Lower Bound)

Lower bound

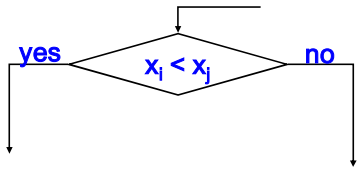
- Def : A lower bound of a problem is the least time complexity required for any algorithm which can be used to solve this problem
- Today : sorting lower bound

- Challenge. How to prove a lower bound for all conceivable algorithms?
- Model of computation. Decision trees.
  - Can access the elements only through pairwise comparisons.
  - All other operations (control, data movement, etc.) are free.
- Cost model. Number of compares.
- Q: Realistic model?
  - A1: Yes. Java, Python, C++, ...
  - A2: Yes. Mergesort, insertion sort, quicksort, heapsort, ...
  - A3: No. Bucket sort, radix sorts, ...

Comparison–Based Sorting Algorithms

- Elements are rearranged by comparing values of keys
- Comparisons performed at given stage will depend on outcomes of previous comparisons and previous rearrangements of keys.

```
Insertion sort
    for j = 2 to n to
      key = a[j]
      i = j-1
      while i > 0 and a[i] > key do
        a[i]+1 = a[i]
        i = i -1
      endwhile
      a[i+1] := key
    endfor
```



- Many sorting algorithms are comparison based :
- Bubble-sort, selection-sort, heap-sort, merge-sort, quick-sort

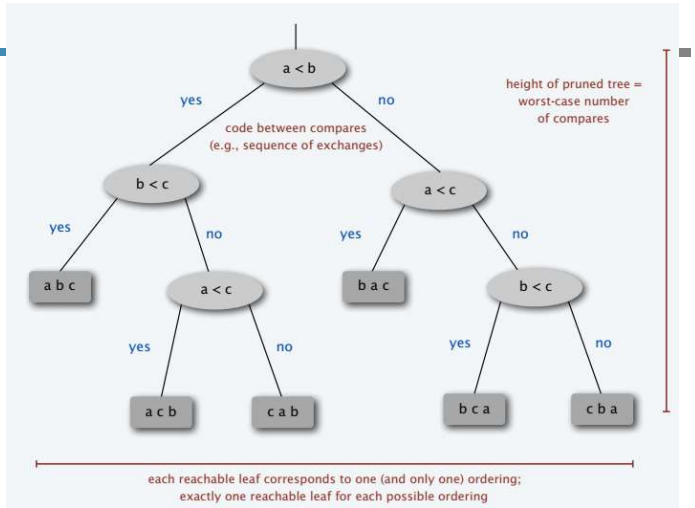
Algorithm	Worst case	Average	Space Usage
Insertion	$n^2/2$	$\theta(n^2)$	in place
Quicksort	$n^2/2$	$\theta(n \log n)$	log n
Mergesort	n lg n	$\theta(n \log n)$	n
Heapsort	2n lg n	$\theta(n \log n)$	in place

Lower Bound for Sorting

- Are there better algorithms?
- Goal: Prove that any sorting algorithm based on only comparisons takes $\Omega(N \log N)$ comparisons in the worst case (worse-case input) to sort N elements
- How many possible orderings do we have for sorting N distinct elements?
  - e.g.,Sorting an array of 3 elements, a, b and c produces 6 or 3! distinct orderings
  - the sorted output for a, b, c can be a b c, b a c, a c b, c a b, c b a, b c a.
  - N! possible orderings

Decision tree & Lower Bound for Sorting Problem

- Decision tree example: Insertion sort for 3 element
  - a b c
- Any comparison-based sorting process can be represented as a binary decision tree
  - True, False
- Each node represents a set of possible orderings, consistent with all the comparisons that have been made
- The tree edges are results of the comparisons



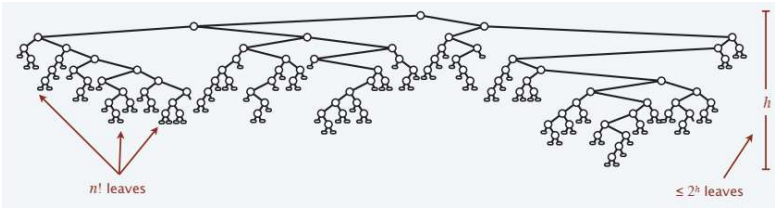
Theorem. Any deterministic compare-based sorting algorithm must make $\Omega(n \log n)$ compares in the worst-case.

Proof.

Assume array consists of n distinct values a_1 through a_n.

Worst-case number of compares = height h of pruned decision tree.

Binary tree of height h has ≤ $2h$ leaves.
$n!$ different orderings ⇒ $n!$ reachable leaves.



Minimum (lower bound) height (h) is $\log_2(n!)$

$\log_2(n!) = \log_2(n*(n-1)*(n-2)*....*(n/2)*....2*1)$

$= \log_2(n)+\log_2(n-1)+ ...+ \log_2(n/2)+....+ \log_2(2)+\log_2(1)$

$\geq \log_2(n)+\log_2(n-1)+ ...+ \log_2(n/2)$

$\geq n/2 \log_2(n/2) = n/2 (\log_2(n)-\log 22) = n/2 \log_2(n) - n/2$

So, $\log_2(n!) = \Omega(n \log n)$

Stirling 's approximation says that $n! > \sqrt{2n\pi}\left(\frac{n}{e}\right)^n > \left(\frac{n}{e}\right)^n$

$\log ((n / e)^n) \leq h$

$n \log (n / e) \leq h$

$n(\log n - \log e) \leq h$

$n \log n - n \log e \leq h$

So, $\log_2(n!) = \Omega(n \log n)$

ผศ. ดร. จักรินทร์ ชวชาติ
ผศ. เบญจมาศ ปัญญางาม

Theorem. Any deterministic compare-based sorting algorithm must make $\Omega(n \log n)$ compares in the worst-case.

Proof.
Assume array consists of n distinct values a_1 through a_n.
Worst-case number of compares = height h of pruned decision tree.
Binary tree of height h has ≤ 2^h leaves.
$n!$ different orderings ⇒ $n!$ reachable leaves.

$2^h \geq$ #leaves $\geq n!$

$h \geq \log_2(n!)$

$\geq n\log_2 n - n/\ln 2$

stirling's formula

ผศ. ดร. จักรินทร์ ชวชาติ
ผศ. เบญจมาศ ปัญญางาม

- The height of a binary tree which has n! leaves is at least $\Omega(n \log n)$
- worst-case running time is at least $\Omega(n \log n)$

- A lower bound for any comparison based sorting algorithm to sort a list of n distinct elements in the worst-case
- Does this mean that we can not do any better?? NO
  - Some types of sorting in linear time by using additional properties of input elements
  - Counting sort , Radix sort

ผศ. ดร. จักรินทร์ ชวชาติ
ผศ. เบญจมาศ ปัญญางาม

Sorting In Linear Time: Counting sort

- No comparisons between elements!
- But...depends on assumption about the numbers being sorted
- We assume numbers are in the range 1.. k
- Can be made in place by computing the cumulative frequencies.
- It is stable.

Counting Sort

	1	2	3	4	5	6	7	8
Data	3	6	4	1	3	4	1	4

Count the number of occurrence

	1	2	3	4	5	6
Count	2	0	2	3	0	1

Find position

	1	2	3	4	5	6
Count	2	2	4	7	7	8

```
for i=1 to k do C[i] = 0
for j=1 to n do
        C[D[j]] =C[D[j]] + 1
for i=2 to k do
        C[i] =C[i] + C[i–1]
```

Input: D[1..n], where D[j]∈ {1, 2, 3, ..., k} for j = 1 to n

Output: S[1..n], sorted (notice: not sorting in place)

Also: Array C[1..k] for auxiliary storage (constant for constant k)

	1	2	3	4	5	6	7	8
Data	3	6	4	1	3	4	1	4

Find position

	1	2	3	4	5	6
Count	2	2	4	7	7	8

Sort

```
For k =n downto 1 do
  S[C[D[k]]] =D[k]
  C[D[k]]--
```

	1	2	3	4	5	6	7	8
sortedData								4

	1	2	3	4	5	6
Count	2	2	4	6	7	8

	1	2	3	4	5	6	7	8
sortedData	1							4

	1	2	3	4	5	6
Count	1	2	4	6	7	8

	1	2	3	4	5	6	7	8
sortedData	1						4	4

	1	2	3	4	5	6
Count	1	2	4	5	7	8

	1	2	3	4	5	6	7	8
sortedData	1	1	3	3	4	4	4	6

Radix Sort

- ใช้แต่ละตำแหน่งตัวเลข โดยเริ่มตั้งแต่ตำแหน่งนัยสำคัญน้อยที่สุด (least significant digit)

329 457 657 839 436 720 355

	ผลลัพธ์การเรียงรอบที่ 1
ชุดที่ 0 720	720
ชุดที่ 1	355
ชุดที่ 2	436
ชุดที่ 3	457
ชุดที่ 4	657
ชุดที่ 5 335	329
ชุดที่ 6 436	839
ชุดที่ 7 457 657	
ชุดที่ 8	
ชุดที่ 9 329 839	

Radix Sort

- จากผลลัพธ์ของรอบที่ 1

720 355 436 457 657 329 839

ผลลัพธ์การเรียงรอบที่ 2

ชุดที่ 0
ชุดที่ 1 — 720
ชุดที่ 2 — 329
ชุดที่ 3 — 436
ชุดที่ 4
ชุดที่ 5 — 839
ชุดที่ 6 — 355
ชุดที่ 7
ชุดที่ 8 — 457
ชุดที่ 9 — 657

Radix Sort

- The running time of radix sort?
  - $\Theta(kn)$, where k represents the number of digitals in the keys.
- If we assume k is a constant, the running time of radix sort is $\Theta(n)$
  - a linear time sorting algorithm! Radix sort is NOT a comparison sort

329	720	720	329
457	355	329	355
657	436	436	436
839	457	839	457
436	657	355	657
720	329	457	839
355	839	657	720

Counting Sort & Radix Sort

- Use Counting sort to sort on digits :
- Sort n numbers on digits that range from 1..k
- Time: O(n + k)
- Each pass over n numbers with d digits takes time O(n+k), so total time O(dn+dk)
- When d is constant and k=O(n), takes O(n) time

Bucket Sort

BUCKET_SORT (A)

n =size of array A,

k = size of bucket B

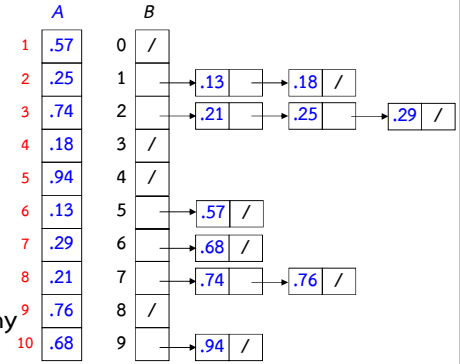
1. For i = 1 to n do

   Insert A[i] into an appropriate bucket.

2. For i = 1 to k do

   Sort ith bucket using any reasonable comparison sort.

3 Concatenate the buckets together in order



Bucket Sort

- Let S(m) denote the number of comparisons for a bucket with m keys and n_i be the no of keys in the i-th bucket.
- Total number of comparisons (all buckets) = $\sum k_{i=1}\ S(n_i)$
- If the keys are uniformly distributed the expected size of each bucket = n/k
- Let, $S(m)=\Theta(m\ (\log\ m))$
- Total number of comparisons
- $= k(n/k)\ \log(n/k)$
- $= n\ \log(n/k)$
- If k=n/10 , then n log(10) comparisons would be done and running time would be linear in n.

Summary of Sorting

Sorting choices:

- $O(N^2)$ – Bubble sort, Insertion Sort
- O(N log N) average case running time
  - Heapsort: In-place, not stable.
  - Mergesort: O(N) extra space, stable.
  - Quicksort: Claimed fastest in practice but, $O(N^2)$ worst case. Needs extra storage for recursion. Not stable.
- O(N) – Radix Sort: fast and stable. Not comparison based. Not in-place.