Linear Sorting

Curs Fall 2018

Upper and lower bounds on time complexity of a problem.

A problem has a time upper bound $T_U(n)$ if there is an algorithm A such that for any input e of size n:
$A(e)$ gives the correct answer in $\leq T_U(n)$ steps.

A problem has a time lower bound $T_L(n)$ if there is NO algorithm which solves the problem if time $< T_L(n)$, for any input e of size n.

It may be that an algorithm solves the problem faster that $T_L(n)$ for a specific input.

Lower bounds are hard to prove, as we have to consider every possible algorithm.

Upper and lower bounds on time complexity of a problem.

- Upper bound: $\exists A, \forall e$ time $A(e) \leq T_U(|e|)$,
- Lower bound: $\forall A, \exists e$ time $A(e) \geq T_L(|e|)$,

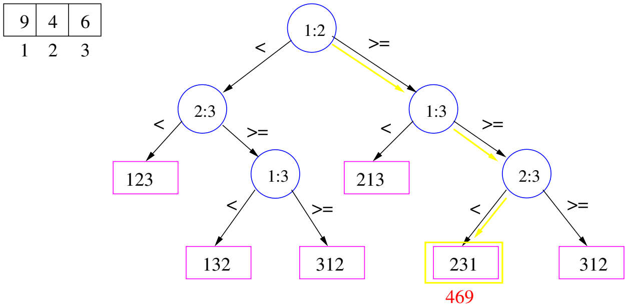
To prove an upper bound: produce an A which works for any e, $|e| = n$.

To prove a lower bound , show that for any possible algorithm, the time on an input is greater than the lower bound.

Lower bound for **comparison based** sorting algorithm.

Use a decision tree: A binary tree where,

- each internal node represents a comparison $a_i : a_j$, the left subtree represents the case $a_i \leq a_j$ and the right subtree represents the case $a_i > a_j$
- each leaf represents one of the n! possible permutations $(a_{\pi(1)}, a_{\pi(2)}, \dots, a_{\pi(n)})$. Each of the n permutations must appear as one of the leaves of the tree

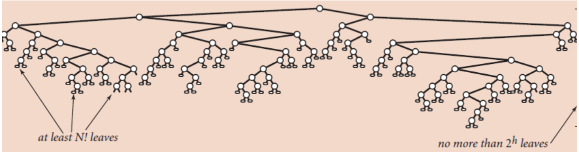


Theorem

Any comparison sort that sorts n elements must perform $\Omega(n \lg n)$ comparisons.

Proof.

Equivalent to prove: Any decision tree that sorts n elements must have height $\Omega(n \lg n)$.
Let h the height of a decision tree with $n!$ leaves,
$n! \leq 2^h \Rightarrow h \geq \lg(n!) > \lg(\frac{n}{e})^n = \Omega(n \lg n)$. □



Linear sorting: Counting sort

Assume the input $A[0 \dots n]$, is an array of integers betwen 0 and k.
Need: an array $B[0 \dots n]$ as the output
and an array $C[0 \dots k]$ as scratch.

Counting (A, k)
for $i = 0$ to k **do**
  do $C[i] := 0$
end for
for $i = 0$ to n **do**
  do $C[A[i]] := C[A[i]] + 1$
end for
for $i = 0$ to k **do**
  do $C[i] := C[i] + C[i-1]$
end for
for $i = n$ downto 0 **do**
  do $B[C[A[i]]] := A[i]$
  $C[A[i]] := C[A[i]] - 1$
end for

Counting (A, k)

for $i = 0$ to k **do**

do $C[i] := 0$ { $O(k)$}

end for

for $i = 0$ to n **do**

do $C[A[i]] := C[A[i]] + 1$ { $O(n)$}

end for

for $i = 0$ to k **do**

do $C[i] := C[i] + C[i-1]$ { $O(k)$}

end for

for $i = n$ down to 0 **do**

$B[C[A[i]]] := A[i]$ { $O(n)$}

$C[A[i]] := C[A[i]] - 1$

end for

Time complexity: $T(n) = O(n+k)$ if $k = O(n)$, then $T(n) = O(n)$.

Linear sorting: Radix sort

An important property of counting sort is that it is stable, numbers with the same value, appear in the output in the same order as they do in the input.
For instance Heap sort is not stable.

Given an array A with n keys, each one with d digits, the Radix (Least Significant Digit),

Radix LSD (A, d)
for $i = 1$ to d **do**
  Use stable sorting to sort the i-th digit of A.
end for

What does it mean Radix?

Radix means base

Radix 10=Decimal; Radix 2= Binary; Radix 16=Hexadecimal.

To transform an integer Radix 16 to a decimal integer
$(4CF5)_{16} = (4 \times 16^3 + 12 \times 16^2 + 15 \times 16^1 + 5 \times 16^0) = 19701$

Binary	Hex	Decimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Example

329		720		720		329
475		475		329		355
657		355		436		436
839	$\Rightarrow$	436	$\Rightarrow$	839	$\Rightarrow$	457
436		657		355		657
720		329		657		720
355		839		475		839

Theorem (Correctness of Radix)

The previous algorithm sort correctly n keys.

Induction on d.

If $d = 1$ the stable sorting works. Assume it is true for $d - 1$, to sort the d-th digit,
if $a_d < b_d$ then a will be placed before b,
if $b_d < a_d$ then b will be placed before a,
if $b_d = a_d$ then as we are using a stable sorting a and b will remain in the same order, which by hypothesis was already the correct one. $\square$

Complexity

Given n integers each with d digits, each digit in the range 0 to 9, if we use counting sorting: $T(n,d) = \Theta(d(n+9))$.

Faster?
Given n b-bits integers, we can view each as having $d = b/r$ "digits" of r bits each.
Each bit is an integer in the range 0 to $2^r - 1$.
So we can use counting sort with $k = 2^r - 1$.

For ex., if we have words of $b = 32$ bits, which we split in $d = 4$ $r = 8$-bit digits:
1 1 0 0 1 0 1 0 0 0 1 1 0 1 0 0 1 1 1 0 1 0 0 1 1 1 0 0 1 0 0 0

Each pass of counting sort takes $\Theta(n+k) = \Theta(n+2^r)$, as there are d passes
$\Rightarrow T(n,b) = \Theta(d(n+2^r)) = \Theta((b/r)(n+2^r))$.

Given b and n, how to choose r?

- If we take $r >> \log n$, then $T(n,b)$ is exponential,
- If $r \sim \lg n \Rightarrow T(n,b) = \Theta(\frac{b}{\lg n}(2n))$.

Complexity of RADIX sorting: Given n positive integers, each integer with a maximal value of $f(n)$ and with at most d digits necessaries to represent each integer in RADIX b.
Notice that $d = \log_b f(n)$.
Then the complexity $T(n)$ of using RADIX to sort the n integers is:

$$T(n) = O((n+b)d) = O(n+b)\log_b f(n).$$

Example

Given n integers, each one with a value between 0 and $n^2 - 1$, to sort the n integers in linear time,
Using $T(n) = O((n+b)d) = O(n+b)\log_b f(n)$,
where $f(n) \leq n^2 - 1$.

We want $T(n) = \Theta(n)$, i.e. $d = \Theta(1)$,
so we need $d = \log_b(n^2 - 1) = \Theta(1)$.

taking $b = n$ then $d = \Theta(1)$ and we have $T(n) = \Theta(1)$.

Comparing radix and counting:

For n integers, each integer with at most d digits, where each digit is in the range $[0, 9]$:

- Counting sort is $\Theta(9dn)$,
- Radix with the choice of $r = \log_9 n$-digits can sort n d-digit numbers in $\frac{d}{\log_9 n}\Theta(9n)$.

Comparing radix and quicksort:

Consider 2000 integers of 32 bits each:

- Quicksort needs to do $\lg 2000 = 11$ passes over the data,
- Radix sort with digits of 11-bits, takes 3 passes (at each one counting sort makes 2 passes).

Empirically, when dealing with natural numbers, radix is better than other sorting methods for values of $n > 2000$.

A bit of history.

Radix and all counting sort are due to Herman Hollerith.
In 1890 he invented the card sorter that allowed to shorten the US census to 5 weeks, using punching cards.