Linear Sorting

Curs 2016
A problem has a time upper bound $T_U(n)$ if there is an algorithm $A$ such that for any input 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 every input 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: \( \exists A, \forall e \text{ time } A(e) \leq T_U(|e|) \),

- Lower bound: \( \forall A, \exists e \text{ time } A(e) \geq T_L(|e|) \),

To prove an upper bound: produce an \( A \) which works for any \( e \).

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 $(a_{\pi(1)}, a_{\pi(2)}, \ldots, a_{\pi(n)})$. Each of the $n$ permutations must appear as one of the leaves of the tree
**Theorem**

Any comparison sort that sorts $n$ elements must performe $\Omega(n \lg n)$ 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,

$n! \leq 2^h \Rightarrow h \geq \lg(n!) > \lg\left(\frac{n}{e}\right)^n = \Omega(n \lg n)$. 

$\square$
When randomization helps: Bucket sort

Assume the input $A[1 \ldots n], 0 \leq A[i] < 1$ is generated by a random process that distributes elements uniformly over $[0, 1)$.

Bucket sort is a deterministic algorithm, which sorts $n$ keys in expected time $O(n)$, over the choice of random input.

Divide $[0, 1)$ into $n$ buckets and distribute the $n$ elements to sort into the $n$ buckets.

Using insertion, sort the elements in each bucket and go through the buckets in order.

Use an auxiliary Data Structure: $B[0 \ldots n − 1]$ of linked lists.
Algorithm

**Bucket** \((A[1 \ldots n])\)

for \(i = 1\) to \(n\) do
  do insert \(A[i]\) into \(B[\lfloor nA[i]\rfloor]\)
end for

for \(j = 0\) to \(n - 1\) do
  do use insertion to sort \(B[j]\)
end for

concatenate \(B[0], B[1], B[2], \ldots, B[n - 1]\)
Example bucket sort

**Input data A:**

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>.76</td>
<td>.16</td>
<td>.78</td>
<td>.26</td>
<td>.11</td>
<td>.67</td>
<td>.21</td>
<td>.31</td>
</tr>
</tbody>
</table>

**Diagram:**

- B[7]
- B[7]
- B[7]
Correctness

Given $A[i]$ and $A[j]$ must prove they are appear in the proper relative order.

If they fell in the same bucket, as insertion is correct, they will be output in the same order.


If not $k = \lfloor nA[i] \rfloor \geq \lfloor nA[j] \rfloor = l$ contradiction to $k < l$. 
Average complexity

Let $n_i$ a random variable counting the elements in the $B[i]$ bucket.
Let $T(n)$ a rv counting the running time of bucket.
As insertion sort is quadratic, $T(n) = \Theta(n) + \sum_{i=0}^{n-1} n_i^2$.

\[
\therefore \mathbb{E}[T(n)] = \mathbb{E}\left[\Theta(n) + \sum_{i=0}^{n-1} n_i^2\right] = \Theta(n) + \sum_{i=0}^{n-1} \mathbb{E}[n_i^2]
\]

Claim: $\mathbb{E}[n_i^2] = 2 - \frac{1}{n}$
For $0 \leq i \leq n - 1$, $1 \leq j \leq n$, define irv:

\[
X_{ij} = \begin{cases} 
1 & \text{if } A[j] \text{ falls in bucket } B[i], \\
0 & \text{otherwise.}
\end{cases}
\]

Then $n_i = \sum_{j=1}^{n} X_{ij}$
Average complexity

\[ \mathbb{E} \left[ n_i^2 \right] = \mathbb{E} \left[ \left( \sum_{j=1}^{n} X_{ij} \right)^2 \right] = \mathbb{E} \left[ \sum_{j=1}^{n} \sum_{k=1}^{n} X_{ij} X_{ik} \right] \]

\[ = \mathbb{E} \left[ \sum_{j=1}^{n} X_{ij}^2 + \sum_{j=1}^{n} \sum_{1 \leq k \leq n, k \neq j} X_{ij} X_{ik} \right] \]

\[ = \sum_{j=1}^{n} \mathbb{E} \left[ X_{ij}^2 \right] + \sum_{j=1}^{n} \sum_{1 \leq k \leq n, k \neq j} \mathbb{E} \left[ X_{ij} X_{ik} \right] . \]

As \( \Pr [X_{ij} = 1] = \frac{1}{n} \), then

\[ \mathbb{E} \left[ X_{ij}^2 \right] = 1^2 \cdot \frac{1}{n} + 0(1 - \frac{1}{n}) = \frac{1}{n} \]
Average complexity

When $k \neq j$ then $X_{ij}$ and $X_{ik}$ are independent, hence

$$E[X_{ij}X_{ik}] = \frac{1}{n} \frac{1}{n} = \frac{1}{n^2}$$

Substituting above:

$$E\left[n_i^2\right] = \sum_{j=1}^{n} \frac{1}{n} + \sum_{j=1}^{n} \sum_{1 \leq k \leq n, k \neq j} \frac{1}{n^2}$$

$$= \frac{n}{n} + \frac{n(n-1)}{n^2} = 1 + \frac{n-1}{n} = 2 - \frac{1}{n},$$

proving our claim.

Therefore, $E[T(n)] = \Theta(n) + n(2 - \frac{1}{n}) = \Theta(n)$
To sort in linear time we cannot use comparisons!

What else?

Restricting keys to be numbers?
**Linear sorting: Counting sort**

Assume the input $A[1\ldots n]$, is an array of integers between 0 and $k$. Need: an array $B[1\ldots n]$ as the output and an array $C[1\ldots k + 1]$ as scratch.

**Counting** ($A, k$)

```plaintext
for $i = 0$ to $k$ do
  do $C[i] := 0$
end for

for $i = 1$ to $n$ do
  do $C[A[i]] := C[A[i]] + 1$
end for

for $i = 1$ to $k$ do
  do $C[i] := C[i] + C[i - 1]$
end for

for $i = n$ downto 1 do
      do $C[A[i]] := C[A[i]] - 1$
end for
```
Counting \((A, k)\)
for \(i = 0\) to \(k\) do
  do \(C[i] := 0\) \{ \(O(k)\)\}
end for
for \(i = 1\) to \(n\) do
  do \(C[A[i]] := C[A[i]] + 1\) \{ \(O(n)\)\}
end for
for \(i = 1\) to \(k\) do
  do \(C[i] := C[i] + C[i - 1]\) \{ \(O(k)\)\}
end for
for \(i = n\) downto 1 do
  do \(B[C[A[i]]] := A[i]\) \{ \(O(n)\)\}
  \(C[A[i]] := C[A[i]] - 1\) \{ \(O(k)\)\}
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
Example

<table>
<thead>
<tr>
<th>329</th>
<th>720</th>
<th>720</th>
<th>329</th>
</tr>
</thead>
<tbody>
<tr>
<td>475</td>
<td>475</td>
<td>329</td>
<td>355</td>
</tr>
<tr>
<td>657</td>
<td>355</td>
<td>436</td>
<td>436</td>
</tr>
<tr>
<td>839</td>
<td>436</td>
<td>839</td>
<td>457</td>
</tr>
<tr>
<td>436</td>
<td>657</td>
<td>355</td>
<td>657</td>
</tr>
<tr>
<td>720</td>
<td>329</td>
<td>657</td>
<td>720</td>
</tr>
<tr>
<td>355</td>
<td>839</td>
<td>475</td>
<td>839</td>
</tr>
</tbody>
</table>
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.
Complexity

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

Given $n$ words of $b$ bits each. View each word as having $d = b/r$ digits of $r$ bits

```
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
```

In this case $d = 4$

Notice $r$ bits can take value between 0 and $2^r$, so each pass of counting sort will take:

$\Theta(n + 2^r) \Rightarrow T(n, b) = \Theta(d(n + 2^r)) = \Theta((b/r)(n + 2^r))$.

Given $b$ and $n$, how to choose $r$?

- If we take $r \gg \log n$, then $T(n, b)$ is exponential,
- If $r \sim \log n \Rightarrow T(n, b) = \Theta(bn/\log n)$. 
Comparing radix and counting:

- Counting sort is $\Theta(k)$, which for $k \gg O(n)$ could be bad. If $k = O(n^2)$ then counting is $\Theta(n^2)$.
- Radix with the choice of of $\lg n$-bit digits can sort $n$ $d$-digit numbers (from 0 to $n^d - 1$) in $d\Theta(n)$. Therefore for $k = O(n^2)$, Radix can sort it in $\Theta(2n)$. 
Comparing radix and quicksort:

Consider 2000 integers of 32 bits each:

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

Empirically, radix is better than others for $n > 2000$. 
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.