Bucket-Sort

Have seen lower bound of $\Omega(n \log n)$ for comparison-based sorting algs
Some “cheating” algorithms achieve $O(n)$, given certain assumptions re input

One example: **bucket sort**

Assumption: input numbers to be sorted are drawn from **uniform distribution** on $[0, 1)$
In this case, **expected** running time of bucket sort is $O(n)$
Alg maintains “buckets” (linked lists). Basic idea:

- if you have $n$ input elements, then we need $n$ buckets
- divide $[0, 1)$ evenly into $n$ consecutive sub-intervals
  $[0, 1/n), [1/n, 2/n), \ldots, [(n-1)/n, 1)$ (that's them buckets)
- given some element $A[i] \in [0, 1)$, throw it into bucket with index $[n \cdot A[i]]$
- hope that input is distributed evenly among buckets
- sort buckets separately and concatenate results
Input $A = A[1], \ldots, A[n]$ with $A[i] \in [0, 1)$ drawn uniformly at random

Need auxiliary array $B[0], \ldots, B[n-1]$ of linked lists

**Bucket-Sort**($A$)

1. $n \leftarrow \text{length}(A)$
2. \textbf{for} $i \leftarrow 1$ \textbf{to} $n$ \textbf{do}
3. \hspace{1em} insert $A[i]$ into list $B[\lfloor n \cdot A[i] \rfloor ]$
4. \textbf{end for}
5. \textbf{for} $i \leftarrow 0$ \textbf{to} $n - 1$ \textbf{do}
6. \hspace{1em} sort list $B[i]$ with insertion sort
7. \textbf{end for}
8. concatenate lists $B[0], \ldots, B[n-1]$ together in order

**Claim:** expected running time is $O(n)$
Example

10 inputs elements, thus buckets
[0, 1/10), [1/10, 2/10), ... [9/10, 1)

After sorting buckets:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.32</td>
<td>0.02</td>
</tr>
<tr>
<td>0.12</td>
<td>0.12</td>
</tr>
<tr>
<td>0.78</td>
<td>0.22</td>
</tr>
<tr>
<td>0.55</td>
<td>0.32</td>
</tr>
<tr>
<td>0.91</td>
<td>0.41</td>
</tr>
<tr>
<td>0.22</td>
<td>0.55 → 0.59</td>
</tr>
<tr>
<td>0.41</td>
<td>/</td>
</tr>
<tr>
<td>0.59</td>
<td>0.72 → 0.78</td>
</tr>
<tr>
<td>0.72</td>
<td>/</td>
</tr>
<tr>
<td>0.02</td>
<td>0.91</td>
</tr>
</tbody>
</table>
Correctness obvious

Why expected running time $O(n)$?

Certainly depends on size of buckets (# of elements in linked lists)

Let $n_i$ be random variable denoting size of $i$-th bucket, $B_i$

Insertion sort is $O(n^2)$ alg, thus overall running time

$$T(n) = \Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)$$

Take expectations and do some stuff:

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

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

$$= \Theta(n) + \sum_{i=0}^{n-1} E[O(n_i^2)]$$

$$= \Theta(n) + \sum_{i=0}^{n-1} O(E[n_i^2])$$
So, what’s $E[n_i^2]$?

Claim: $E[n_i^2] = 2 - 1/n$ for $0 \leq i \leq n - 1$

Clearly same expectations for all buckets since input is drawn from uniform distribution on $[0, 1)$: each value is equally likely to fall into any bucket.

Define r.v. $X_{ij}$ for $i = 0, \ldots, n - 1$ and $j = 1, \ldots, n$:

$$X_{ij} = \begin{cases} 
1 & A[j] \text{ falls into bucket } i \\
0 & \text{otherwise}
\end{cases}$$

Clearly,

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

because $X_{ij}$ is equal to 1 for each element that falls into $i$-th bucket.
\[ E[n_i^2] = E \left[ \left( \sum_{j=1}^{n} X_{ij} \right)^2 \right] \]

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

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

\[
= \sum_{j=1}^{n} E[X_{ij}^2] + \sum_{1 \leq k \leq n} E[X_{ij} X_{ik}] \\
\]

(*) is because

\[
\left( \sum_{j=1}^{n} X_{ij} \right)^2 \\
= (X_{i1} + X_{i2} + \cdots + X_{i,n-1})^2 \\
= X_{i1}X_{i1} + X_{i1}X_{i2} + \cdots + X_{i1}X_{i,n-1} + \\
X_{i2}X_{i1} + X_{i2}X_{i2} + \cdots + X_{i2}X_{i,n-1} + \cdots + \\
X_{i,n-1}X_{i1} + X_{i,n-1}X_{i2} + \cdots + X_{i,n-1}X_{i,n-1}
\]
By definition of expectation,

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

and when \( k \neq j \), \( X_{ij} \) and \( X_{ik} \) are independent, and thus

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

This gives

\[
E[n_i^2] = \sum_{j=1}^{n} \frac{1}{n} + \sum_{1 \leq k \leq n \atop k \neq j} \frac{1}{n^2} \\
= n \cdot \frac{1}{n} + n(n-1) \cdot \frac{1}{n^2} \\
= 1 + \frac{n-1}{n} \\
= 1 + \frac{n-1}{n} - \frac{1}{n} \\
= 2 - \frac{1}{n}
\]

and therefore

\[
E[T(n)] = \Theta(n) + \sum_{i=0}^{n-1} O(E[n_i^2]) \\
= \Theta(n) + \sum_{i=0}^{n-1} \left(2 - \frac{1}{n}\right) = \Theta(n)
\]
**Note:** bucket sort may have linear running time even when input is not drawn from uniform distribution on \([0, 1)\):

We’re fine whenever sum of squares of bucket sizes is linear in \# of elements (that’s the insertion sort)

Or if we use, say, merge sort, then whenever
\[
\sum_{i=0}^{n-1} n_i \log n_i = O(n)
\]
And now for something completely different

**Sorting** yields complete information re order of input elements

But what if we don’t really need all this information, but perhaps just want to know the value of the $k$-th smallest element?

Sorting clearly solves this problem, but is there perhaps something faster?

Simple for smallest, 2nd-smallest, $k$-th smallest for, say, constant $k$

But what about $n/2$-th smallest? $\sqrt{n}$-th smallest?

Exact problem formulation:

**Input:** Set $A$ of $n$ (distinct) numbers and a number $i \in \{1, \ldots, n\}$

**Output:** Element $x \in A$ that is larger than exactly $i - 1$ other elements of $A$
We’re going to see $\text{Select}(A, i)$ with \textbf{linear worst-case running time}.

Idea is D&C:

1. Divide $n$ elements into $\lfloor n/5 \rfloor$ groups of 5 elements each, and at most one group containing the remaining $n \mod 5 < 5$ elements.

2. Find median of each of the $\lceil n/5 \rceil$ groups by sorting each one, and then picking median from sorted group elements.

3. Call $\text{Select}$ recursively on set of $\lceil n/5 \rceil$ medians found above, giving median-of-medians $x$.

4. Partition input around $x$. Let $k$ be $\#$ of elements on low side plus one, so $x$ is $k$-th smallest element and there are $n - k$ elements on high side of partition.

5. If $i = k$, return $x$. Otherwise use $\text{Select}$ recursively to find $i$-th smallest element of low side if $i < k$, or $(i - k)$-th smallest on high side if $i > k$. 
First question: what’s wrong with basically running Quicksort and disregarding one sub-problem at each recursion step?

Answer: might be that we partition in a bad way and always follow the large sub-problem: $\Omega(n^2)$

Perfect splits result in $\Theta(n)$ (thus randomised version would have expected running time of $O(n)$)

Our somewhat more complicated algorithms guarantees $O(n)$
Observations:

- $n = 5 \cdot 5 + 3 = 28$ elements are circles
- groups are columns
- white circles are medians of groups
- $x$ is median of medians
- arrows from greater to smaller elements: three out of every full group to right of $x$ are greater than $x$, and three out of every group to left of $x$ are smaller than $x$ (simply because the corresponding medians are greater/smaller than $x$).
- elements on shaded background are guaranteed to be greater than $x$
We want lower-bound $\#$ elements greater than $x$

Know: at least half of medians found in step 2 are greater than $x$

Thus at least half of $\lceil n/5 \rceil$ groups contribute 3 elements greater than $x$ (excepts for incomplete group, and group containing $x$)

Disregard these two, and we get

$$3 \left( \left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil - 2 \right) \geq \frac{3n}{10} - 6$$

Same is true for $\#$ elements smaller than $x$

Thus, in worst case, Select is called recursively on at most

$$n - \left( \frac{3n}{10} - 6 \right) = \frac{7n}{10} + 6$$

elements (step 5): we have just established lower bounds for each partition, and thereby also upper bound.
Steps 1, 2, and 4 take $O(n)$ time each (step 2: $O(n)$ calls to insertion sort on sets of size $O(1)$)

Step 3: time $T(\lceil n/5 \rceil)$

Step 5: time $T(7n/10 + 6)$

Altogether:

$$T(n) \leq \begin{cases} 
\Theta(1) & n \leq 140 \\
T(\lceil n/5 \rceil) + T(7n/10 + 6) + O(n) & n > 140
\end{cases}$$

The “140” is black magic...:-)
Want to show that $T(n) \leq cn$ for some constant $c$

Assume $T(n) \leq cn$ for $c$ large enough and $n \leq 140$ (no problem)

Also, pick constant $a$ s.t. the $O(n)$ term is at most $an$ (non-recursive component)

Now go forth and substitute:

$$T(n) \leq c\lceil n/5 \rceil + c(7n/10 + 6) + an$$
$$\leq cn/5 + c + 7cn/10 + 6c + an$$
$$= 9cn/10 + 7c + an$$
$$= cn + (-cn/10 + 7c + an)$$

Now $T(n) \leq cn$ if and only if

$$-cn/10 + 7c + an \leq 0$$

equivalent to

$$c \geq 10a \frac{n}{n - 70}$$

when $n \geq 70$
Assumption $n \geq 140$, thus $\frac{n}{n-70} \leq 2$, thus choosing $c \geq 20a$ satisfies inequality

Note: for recursion end, any value greater than 70 would to

OK, that's it: $T(n) \leq cn$ for $c \geq 20a$, thus linear running time