CSE 202 - Algorithms

Sorting-related topics
1. Lower bound on comparison sorting
2. Beating the lower bound
3. Finding medians and order statistics
   (chapters 8 & 9)

The game of “20 questions”

- Suppose I choose one of k objects.
  - We both know the set of objects, e.g. \{1,2,...,k\}.
- You ask me yes-no questions.
  - I answer truthfully.
- How many questions do you need to ask (worst case)?

A binary decision tree for \{1,2,3,4,5\}
How many comparisons for sorting?

- Comparison sorts asks only yes-no questions.
  - “Is x(i) > x(j)”
- A sorting algorithm must get a different sequence of answers on each distinct input.
- For n elements, there are n! possible inputs.
- Thus, we need at least \( \lg(n!) \) comparisons.

Estimating \( \lg(n!) \)

- Direct computation:
  - For \( n>1 \), \( n! < n^n \), so \( \lg(n!) < n \lg n \).
    - so \( \lg(n!) \) is \( O(n \lg n) \).
  - For \( n>1 \), \( n! > (n/2)^{n/2} \).
    - Obvious for \( n \) even.
    - Hand waving for \( n \) odd.
    - Thus, \( \lg(n!) > (n/2) \lg(n/2) = \frac{1}{2} n (\lg n - 1) \).
    - For \( n>4 \), \( (\lg n - 1) > \lg n - (\lg n)/2 = \lg n /2 \).
    - Thus, \( \lg(n!) > \frac{1}{4} n \lg n \), proving \( \lg(n!) \) is \( \Omega(n \lg n) \).
- Or use Stirling's formula: \( n! \approx (2\pi n)^{1/2} (n/e)^n \).
  - Yadda, yadda, yadda ... (Gives a tighter bound).
Best known comparison sort

<table>
<thead>
<tr>
<th>n</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>⌈lg n⌉</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>10</td>
<td>13</td>
<td>16</td>
<td>19</td>
<td>22</td>
<td>26</td>
<td>29</td>
<td>33</td>
<td>37</td>
</tr>
<tr>
<td>Merge sort</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>10</td>
<td>13</td>
<td>16</td>
<td>19</td>
<td>22</td>
<td>26</td>
<td>30</td>
<td>34</td>
<td>38</td>
</tr>
<tr>
<td>Best possible</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>10</td>
<td>13</td>
<td>16</td>
<td>19</td>
<td>22</td>
<td>26</td>
<td>30</td>
<td>34</td>
<td>?</td>
</tr>
</tbody>
</table>

Source: Sloan’s “Encyclopedia of Integer Sequences” (try Google on “sloane sequence”)

Radix Sort (not a comparison sort)

Given a list of \( n \) k-digit numbers,

For \( i = 1 \) to \( k \) {
    partition data into bins according to the \( i \)-th digit;
    reassemble bins into one list;
}

At each iteration, keep the data in each bin in the same order as it was in the list.

Result: you’ll sort the entire list.

Practical considerations:

How do you manage storage?
How do you “reassemble”?
Analysis of Radix Sort

• Assuming “digit” means “base 10 digit” ...
  - What is the complexity?
  - Have we accomplished anything?

• What if one used some other base??

• Is this a linear time algorithm??
  - One “random access” step (with b possible choices) may be worth \( \lg b \) Yes-No questions.
  - If you can arrange things right.

Bucket Sort

Given N data items, uniformly distributed in [0,1].
  - A “reason 2” scenario.

Initialize N Buckets to “empty”;
For I = 1 to N
  Put \( A[I] \) into Bucket \( \lfloor N \times A[I] \rfloor \);
For I = 1 to N
  Sort Bucket I; /* N^2 method is OK */
Concatenate Buckets;

Analysis

Let \( X_{ij} = 1 \) if \( A[i] \) and \( A[j] \) end up in same bucket, 0 otherwise.
\( X_{ij} \) is a random variable. (What is the sample space??)

Let \( T(N) = \sum_{i=1}^{N} \sum_{j=1}^{N} X_{ij} \). \( T(N) \) is upper bound on comparisons needed.
\( E(X_{ij}) = 1/N \), so \( E(T(N)) = \sum_{i=1}^{N} \sum_{j=1}^{N} 1/N = N. \) (Other steps are \( \Theta(N) \).)

Why ??

Fine print: This is all messed up for \( i=j \). Can you fix it?
Sorting small numbers

• Suppose you have N numbers in the range of 0 to K-1.
  - N may be larger than K (repeats are possible)

• What does Radix Sort look like?

• What does Bucket Sort look like?

Both are closely related to Counting Sort

```c
for i = 1 to N, Count[A[i]]++;
for j = 2 to K, Count[j] += Count[j-1];
for i = 1 to N, B[Count[A[i]]--] = A[i];
```
Summary

- Radix sort and bucket sort are linear time under certain assumptions
  - Radix sort – numbers aren’t too long.
    For instance, n numbers in \(1, 2, \ldots, n^2\)
  - Bucket sort – average case; must know distribution.

- Sorting \(n\) \(n\)-bit long numbers in linear time is an open problem.

  There’s a \(O(n \ lg \ lg \ n \ lg \ lg \ lg \ n)\) technique known.

  “Linear for all reasonable values of \(n\), but unlikely to be used in practice.

  Consider \(n = 2^{100}\)

Stability

A sorting algorithm is **stable** if elements with equal keys stay in the same order.

If \(\text{Sort}([5, 8, 3, 10, 8])\) returned \([10, 8, 8, 5, 3]\), it wouldn’t be stable (since 8 and 8 got swapped).

Stable sorting is useful; e.g. you might want to first sort by one key field, then by another.

- Is Heapsort stable?
- Is Quicksort stable?
Order statistics

Select(A,k) - returns k\(^{th}\) smallest from n-element set A.

Median(A) = Select (A, \lfloor n/2 \rfloor).

Consider only comparison-based methods.

Select(A,1) - needs exactly n-1 comparisons.

Tree-based tournament or single pass needs only n-1.
Can't do better - every element except minimum must "lose".

Select(A,2) - can be done with n + \lceil \lg n \rceil comparisons.

Double elimination tournament.

Select(A,k) - can be done with n + k^2 \lg n

What about linear-time Select?
(from now on, assume no duplicates in A)

• Given x, in n-1 comparisons, you can find its rank and partition A into A\(_{lo}\) (items smaller than x) and A\(_{hi}\).

• If rank of x is i, and A = A\(_{lo}\) \cup \{x\} \cup A\(_{hi}\), then
  - if j<i, Select(A, j) = Select(A\(_{lo}\), j) ... or ...
  - if j>i Select(A, j) = Select(A\(_{hi}\), j-i).

• This suggests using divide and conquer
  - Find some x "near" the median "quickly".
  - Partition A into A\(_{lo}\) \cup \{x\} \cup A\(_{hi}\) using n-1 comparisons.
  - Reduce problem to "about" half the size.
  - "Almost" gives recurrence T(n) < T(n/2) + c n.
    which implies T(n) is O(n).
Does this really work??

1. Let $B =$ half of $A$;  free
2. Let $x =$ Median($B$);  $T(n/2)$
3. Find $i=$rank($x$), $A = A_{lo} \cup \{x\} \cup A_{hi}$;  $< n$
4. If ($k<i$) Select ($A_{lo}$, $k$);  $T(3n/4)$
   
   else Select ($A_{hi}$, $k-i$);  (in worst case)

Gives recurrence, $T(n) < T(n/2) + T(3n/4) + cn$

Hmmm ... need to try something different

---

Does this really work (attempt #2)

1. Let $B1, B2, B3$ be thirds of $A$;  free
2. Let $x_j =$ Median($B_j$); $x =$ Median($\{x_j\}$);  $3T(n/3)+3$
3. Find $i=$rank($x$), $A = A_{lo} \cup \{x\} \cup A_{hi}$;  $< n$
4. If ($k<i$) Select ($A_{lo}$, $k$);  $T(??)$
   
   else Select ($A_{hi}$, $k-i$);  (in worst case)

Gives recurrence, $T(n) < 3T(n/3) + T(??) + cn$

Not particularly better

... need to try something different
Does this really work (attempt #3)

1. Let $B_1, B_2, \ldots, B_{n/3}$ each have size 3; free
2. Let $x_j = \text{Median}(B_j)$; $n/3 \times 3 = n$
3. $x = \text{Median}([x_i])$; $T(n/3)$
4. $i = \text{rank}(x)$, $A = A_{lo} \cup \{x\} \cup A_{hi}$; $< n$
5. If $(k<i)$ Select $(A_{lo}, k)$; $T(??)$
   else Select $(A_{hi}, k-i)$; (in worst case)

Gives recurrence, $T(n) < T(n/3) + T(??) + cn$

Are we getting anywhere??
Don't give up!! One more idea and it can be done.

Does this really work (attempt #4)

1. Let $B_1, B_2, \ldots, B(n/5)$ each have size 5; free
2. Let $x_i = \text{Median}(B_i)$; $n/5 \times 7 < 2n$
3. $x = \text{Median}([x_i])$; $T(n/5)$
4. $i = \text{rank}(x)$, $A = A_{lo} \cup \{x\} \cup A_{hi}$; $< n$
5. If $(k<i)$ Select $(A_{lo}, k)$; $T(7n/10)$
   else Select $(A_{hi}, k-i)$; (in worst case)

Gives recurrence, $T(n) < T(n/5) + T(7n/10) + cn$

Yes!!

Best known results: can find median in $3n$ comparisons, lower bound is $2n$. 
Proof that recursion for median algorithm is $O(n)$

Given $T(n) = T(\lfloor n/5 \rfloor) + T(\lceil 7n/10 \rceil) + f(n)$, $T(0)=0$, and $f(n)$ is $O(n)$.
We know $\exists n_0, c_0$ s.t. $\forall n>n_0, f(n) \leq c_0 \cdot n$. (Call this equation [1].)

Let $c = \max (10c_0, \max \{T(n)/n\})$. So $c_0 \leq c/10$ [2] and $\forall n \leq n_0, cn \geq T(n)$. [3]

Claim: $\forall n > 0, T(n) \leq cn$.

Proof by induction on $n$. Let $P(n)$ be the statement, "$T(n) \leq cn$".

Bases cases ($P(n)$ for $n = 1, \ldots, n_0$) : These all follow from [3].

Inductive step: Given $n > n_0$, assume $\forall k < n P(k)$ (i.e. assume $\forall k < n T(k) \leq c k$).
In particular, since $\lfloor n/5 \rfloor < n$, $T(\lfloor n/5 \rfloor) \leq c \lfloor n/5 \rfloor$, which is $\leq cn/5$. [4]

Similarly, $T(\lceil 7n/10 \rceil) \leq c \lceil 7n/10 \rceil \leq 7cn/10$, [5]

Then $T(n) = T(\lfloor n/5 \rfloor) + T(\lceil 7n/10 \rceil) + f(n)$ (definition of $T(n)$.)

\[
\leq cn/5 + 7cn/10 + c_0 n \quad \text{(from [4], [5], and [1].)}
\]

\[
\leq cn/5 + 7cn/10 + cn/10 \quad \text{(from [2].)}
\]

\[
\leq cn(1/5 + 7/10 + 1/10) = cn. \quad \text{Q.E.D.}
\]

What happens if we change floors to ceilings??

Given $T(n) = T(\lceil n/5 \rceil) + T(\lceil 7n/10 \rceil) + f(n)$, $T(0)=0$, and $f(n)$ is $O(n)$.

We could argue that for $n > 100$, $\lfloor n/5 \rfloor < .21n$ and $\lceil 7n/10 \rceil < .71n$.

We'd also can change definition of $c$ to ensure $c_0 \leq .08c$.

To do so, we'd say, "Let $c = \max (c_0/08, \max \{T(n)/n\})$." 

Then, when we get to ...

"Then $T(n) = T(\lceil n/5 \rceil) + T(\lceil 7n/10 \rceil) + f(n)$"

we'll be able to argue that

"$T(n) \leq .21cn + .71cn + .08cn = cn.$"

and be done.