Sorting & Selection

Jeevani Goonetillake
University of Colombo School of Computing
Sri Lanka
Comparison Sort

• A type of sorting algorithm that only reads the list elements through "≤" operator in order to determine with respect to two elements in the list which element should occur first in the final sorted list.

• The requirement is that the operator obey the following two properties:
  • if $a \leq b$ and $b \leq c$ then $a \leq c$ (transitivity)
  • $\forall a \text{ and } b$, exactly one of $a < b$, $b < a$ and $a = b$ is true (trichotomy).
The ideal sorting algorithm would have the following key properties:

- **Stable**: Equal keys aren't reordered.
- **Operates in place**, requiring $O(1)$ extra space.
- **Worst-case** $O(n \log n)$ key comparisons.
- **Adaptive**: Speeds up to $O(n)$ when data is nearly sorted.
Comparison-based Sorting

A lower bound on the number of comparisons will be a lower bound on the complexity of any comparison-based sorting algorithm.

Comparison sort algorithms:

- Merge sort  $\Theta(n \lg n)$
- Heap sort  $\Theta(n \lg n)$
- Insertion sort  $O(n^2)$
- Quick sort  $O(n^2)$

Use decision tree model to prove that lower bound of any comparison sort is $\Omega(n \lg n)$. 

Jeevani Goonetillake (University of Colombo - Sri Lanka) 
Lecture3
A decision tree can model the execution of any comparison sort:

- One tree for each input size \( n \).
- View the algorithm as splitting whenever it compares two elements.
- The tree contains the comparisons along all possible instruction traces.
- The running time of the algorithm => the length of the path taken.
- Worst-case running time => height of tree.
Sort $\langle a_1, a_2, a_3 \rangle$

- The left subtree shows subsequent comparisons if $a_i \leq a_j$.
- The right subtree shows subsequent comparisons if $a_i > a_j$.
Decision-tree Example

Any comparison sort can be turned into a Decision tree

Sort $\langle 9, 4, 6 \rangle$ which are numbered respectively as element 1:2:3

Insertion sort for three elements as a decision tree

Jeevani Goonetilleke (University of Colombo - Sri Lanka)
Lecture3
Decision-tree Example

Sort $\langle 9, 4, 6 \rangle$
Decision-tree Example

Sort $\langle 9, 4, 6 \rangle$
Lower bound for decision-tree sorting

**Theorem.** Any decision tree that can sort $n$ elements must have height $\Omega(n \lg n)$.

**Proof.** Since there are $n!$ possible permutations the tree must contain $\geq n!$ leaves. A height-$h$ binary tree has $\leq 2^h$ leaves. Thus, $n! \leq 2^h$.

\[ \therefore h \geq \lg(n!) = \Omega(n \lg n). \]
Sorting in linear time

Is there a faster algorithm?
A different model of computation?

Counting sort: No comparisons between elements.

Input: $A[1 \ldots n]$, where $A[j] \in \{1, 2, \ldots, k\}$.
Output: $B[1 \ldots n]$, sorted.
Auxiliary storage: $C[1 \ldots k]$. 
Counting sort

for $i \leftarrow 1 \text{ to } k$
  do $C[i] \leftarrow 0$
for $j \leftarrow 1 \text{ to } n$
  do $C[A[j]] \leftarrow C[A[j]] + 1$
  \> $C[i] = |\{\text{key} = i\}|$
for $i \leftarrow 2 \text{ to } k$
  do $C[i] \leftarrow C[i] + C[i-1]$
for $j \leftarrow n \text{ down to } 1$
  do $B[C[A[j]]] \leftarrow A[j]$
  \> $C[A[j]] \leftarrow C[A[j]] - 1$
Counting-sort example

A: 5 1 3 4 1

B:  

C: 1 2 3 4 5

Jeevani Goonetillake (University of Colombo - Sri Lanka)
Lecture 3
Loop 1

\[
\text{for } i \leftarrow 1 \text{ to } k \\
\text{do } C[i] \leftarrow 0
\]
Loop 2

\[
\;
\]

for \( j \leftarrow 1 \) to \( n \)

\[
do \quad C[A[j]] \leftarrow C[A[j]] + 1 \quad \triangleright \quad C[i] = |\{\text{key} = i\}| \]

Jeevani Goonetillake (University of Colombo - Sri Lanka)
Lecture3
Loop 2

for $j \leftarrow 1$ to $n$
  do $C[A[j]] \leftarrow C[A[j]] + 1$

$\triangleright C[i] = |\{\text{key} = i\}|$
Loop 2

\[ A: \begin{array}{ccccc} 
1 & 2 & 3 & 4 & 5 \\
5 & 1 & 3 & 4 & 1 \\
\end{array} \] 

\[ B: \] 

\[ C: \begin{array}{ccccc} 
1 & 2 & 3 & 4 & 5 \\
1 & 0 & 1 & 0 & 1 \\
\end{array} \] 

\[ \text{for } j \leftarrow 1 \text{ to } n \] 
\[ \text{do } C[A[j]] \leftarrow C[A[j]] + 1 \quad \triangleright C[i] = |\{\text{key} = i\}| \]
Loop 2

for $j \leftarrow 1$ to $n$
    do $C[A[j]] \leftarrow C[A[j]] + 1$
      ▷ $C[i] = |\{\text{key} = i\}|$

A: 5 1 3 4 1
B:  
C: 1 0 1 1 1 1
Loop 2

\[ \text{for } j \leftarrow 1 \text{ to } n \]
\[ \text{do } C[A[j]] \leftarrow C[A[j]] + 1 \quad \triangleright C[i] = |\{\text{key} = i\}| \]
Loop 3

\[
\begin{array}{c|c|c|c|c|c}
\hline
& 1 & 2 & 3 & 4 & 5 \\
\hline
A: & 5 & 1 & 3 & 4 & 1 \\
\hline
B: & & & & & \\
\hline
C: & 2 & 0 & 1 & 1 & 1 \\
\hline
C': & 2 & 2 & 1 & 1 & 1 \\
\hline
\end{array}
\]

**for** \( i \leftarrow 2 \) **to** \( k \)

**do** \( C[i] \leftarrow C[i] + C[i-1] \)
Loop 3

\[
\begin{array}{cccccc}
A: & 5 & 1 & 3 & 4 & 1 \\
B: & & & & & \\
C: & 2 & 0 & 1 & 1 & 1 \\
C': & 2 & 2 & 3 & 1 & 1 \\
\end{array}
\]

\textbf{for } i \leftarrow 2 \textbf{ to } k \\
\textbf{do } C[i] \leftarrow C[i] + C[i-1] \quad \triangleright C[i] = |\{\text{key } \leq i\}|
for $i \leftarrow 2$ to $k$

$\text{do } C[i] \leftarrow C[i] + C[i-1]$
Loop 3

for $i \leftarrow 2$ to $k$
do $C[i] \leftarrow C[i] + C[i-1]$
for $j \leftarrow n$ downto 1
\begin{align*}
    & C[A[j]] \leftarrow C[A[j]] - 1
\end{align*}
for $j \leftarrow n$ downto 1

$B[C[A[j]]] \leftarrow A[j]$

$C[A[j]] \leftarrow C[A[j]] - 1$
Loop 4

\[ A: \begin{bmatrix} 5 & 1 & 3 & 4 & 1 \end{bmatrix} \]

\[ B: \begin{bmatrix} 1 & 3 & 4 \end{bmatrix} \]

\[ C: \begin{bmatrix} 1 & 2 & 3 & 3 & 5 \end{bmatrix} \]

\[ C': \begin{bmatrix} 1 & 2 & 2 & 3 & 5 \end{bmatrix} \]

\[ \text{for } j \leftarrow n \text{ downto } 1 \]
\[ \text{do } B[C[A[j]]] \leftarrow A[j] \]
\[ C[A[j]] \leftarrow C[A[j]] - 1 \]
Loop 4

for $j \leftarrow n$ downto 1

do $B[C[A[j]]] \leftarrow A[j]$

$C[A[j]] \leftarrow C[A[j]] - 1$
### Loop 4

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td><strong>A:</strong></td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td><strong>B:</strong></td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td><strong>C:</strong></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td><strong>C':</strong></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

#### Algorithm:

```plaintext
for j ← n downto 1 
do
    B[C[A[j]]] ← A[j]
    C[A[j]] ← C[A[j]] − 1 
```

Jeevani Goonetillake (University of Colombo - Sri Lanka)
Lecture3
Counting Sort - Analysis

\[\Theta(k)\] 
\[\text{for } i \leftarrow 1 \text{ to } k\]
\[\text{do } C[i] \leftarrow 0\]

\[\Theta(n)\] 
\[\text{for } j \leftarrow 1 \text{ to } n\]
\[\text{do } C[A[j]] \leftarrow C[A[j]] + 1\]

\[\Theta(k)\] 
\[\text{for } i \leftarrow 2 \text{ to } k\]
\[\text{do } C[i] \leftarrow C[i] + C[i-1]\]

\[\Theta(n)\] 
\[\text{for } j \leftarrow n \text{ downto } 1\]
\[\text{do } B[C[A[j]]] \leftarrow A[j]\]
\[C[A[j]] \leftarrow C[A[j]] - 1\]

\[\Theta(n + k)\]
Counting Sort

Counting sort is a *stable* sort: it preserves the input order among equal elements.

\[ A: 5 \ 1 \ 3 \ 4 \ 1 \]

\[ B: 1 \ 1 \ 3 \ 4 \ 5 \]
Radix Sort

• Radix sort is a non-comparative integer sorting algorithm.
• Sorts data with integer keys by grouping keys by the individual digits which share the same significant position and value.
• Sorting starts with the least significant bit and is stable.
Radix Sort - Example

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>219</td>
<td>710</td>
<td>710</td>
</tr>
<tr>
<td>458</td>
<td>255</td>
<td>219</td>
</tr>
<tr>
<td>658</td>
<td>436</td>
<td>436</td>
</tr>
<tr>
<td>839</td>
<td>458</td>
<td>839</td>
</tr>
<tr>
<td>436</td>
<td>658</td>
<td>658</td>
</tr>
<tr>
<td>710</td>
<td>219</td>
<td>710</td>
</tr>
<tr>
<td>255</td>
<td>839</td>
<td>839</td>
</tr>
</tbody>
</table>

Jeevani Goonetillake (University of Colombo - Sri Lanka)
Lecture3
Correctness of radix sort

**Induction on digit position**

- Assume that the numbers are sorted by their low-order \( t - 1 \) digits.
- Sort on digit \( t \)
Correctness of radix sort

**Induction on digit position**

- Assume that the numbers are sorted by their low-order \( t - 1 \) digits.

- Sort on digit \( t \)
  - Two numbers that differ in digit \( t \) are correctly sorted.
Correctness of radix sort

**Induction on digit position**

- Assume that the numbers are sorted by their low-order \( t - 1 \) digits.
- Sort on digit \( t \)
  - Two numbers that differ in digit \( t \) are correctly sorted.
  - Two numbers equal in digit \( t \) are put in the same order as the input \( \Rightarrow \) correct order.
Radix Sort Algorithm

• Radixsort\(a[1,\ldots,n]\) where \(a[i] \in \{0,\ldots,m\}\)
  
  for \(1 \leq i \leq m\)

  for \(i = 0,\ldots, \lceil \log m \rceil\) do

  CountingSort\(a\) using bit \(i\) as the key
  (least significant first)

• Note: \(m\) is the maximum possible value in the array.
Radix Sort Analysis

• If maximum possible value in the array => m
• Radix sort takes $O(d \times (n+b))$ time
  where $n$ – size of the array
    $b$ – base of number representation
    $d$ – $\log_b(m)$

Then run time => $O(\log_b(m) \times (n+b))$
  => $O(n \log m)$. 

Jeevani Goonetillake (University of Colombo - Sri Lanka)
Lecture3
Medians and Order Statistics

Jeevani Goonetillake
University of Colombo School of Computing
Sri Lanka
Order statistics

- The $i^{th}$ order statistic of a set of $n$ elements is the $i$th smallest element.
  - The minimum is the first order statistic ($i = 1$).
  - The maximum is the $n$th order statistic ($i = n$).
  - A median is the “halfway point” of the set.
Median

• When $n$ is odd, the median is unique, at $i = (n + 1)/2$.
• When $n$ is even, there are two medians:
  • The **lower median**: $i = \lfloor (n + 1)/2 \rfloor$, and
  • The **upper median**: $i = \lceil (n + 1)/2 \rceil$.
• The phrase “median” normally means the lower median.
Selection Problem

Input : A set $A$ of $n$ (distinct) number and a number $i$, with $1 \leq i \leq n$.

Output : The element $x \in A$ that is larger than exactly $i-1$ other elements of $A$.

• The selection problem can be solved in $O(n \log n)$ time. Sort the numbers using an $O(n \log n)$-time algorithm, such as heapsort or merge sort. Then return the $i$th element in the sorted array.

• Is there a faster algorithm - An $O(n)$-time algorithm?
Finding minimum

Possible to obtain an upper bound of $n-1$ comparisons for finding the minimum of a set of $n$ elements.

• Examine each element in turn and keep track of the smallest one.

• The algorithm is optimal, because each element, except the minimum, must be compared to a smaller element at least once.
Finding minimum

MINIMUM(A)

\[ min \leftarrow A[1] \]

for \( i \leftarrow 2 \) to \( \text{length}[A] \)

\[ \text{do if } min > A[i] \]

\[ \text{then } min \leftarrow A[i] \]

return \( min \)

• The maximum can be found in exactly the same way by replacing the \( > \) with \( < \) in the above algorithm.
Simultaneous minimum and maximum

Some applications need both the minimum and maximum.

**Method 1**
- Find the minimum and maximum independently, using $n-1$ comparisons for each, for a total of $2n-2$ comparisons.
Simultaneous minimum and maximum

Method 2

Can be performed with at most 3 \( \lceil n/2 \rceil \) comparisons

- Maintain the minimum and maximum of elements seen so far.
- Process elements in pairs.
- Compare the elements of a pair to each other.
- Compare the larger element to the maximum so far, and compare the smaller element to the minimum so far.

This gives only 3 comparisons for every 2 elements (which is an improvement to 2 comparisons for each element).
Simultaneous minimum and maximum

Method 2

compare each pair to find minimum and maximum
larger elements: compare to the current maximum
smaller elements: compare to the current minimum
Simultaneous minimum and maximum

Setting up the initial values for the min and max depends on whether \( n \) is odd or even.

- If \( n \) is even, compare the first two elements and assign the larger to max and the smaller to min.
- If \( n \) is odd, set both min and max to the first element.
Simultaneous minimum and maximum

• If $n$ is even, $\# \text{ of comparisons} = 3(n-2) + 1 = 3n - 2$

• If $n$ is odd, $\# \text{ of comparisons} = \frac{3(n-1)}{2} = 3 \left\lfloor \frac{n}{2} \right\rfloor$

• In either case, the $\# \text{ of comparisons} \leq 3 \left\lfloor \frac{n}{2} \right\rfloor$
Selection in expected linear time

Selection of the $i$th smallest element of the array $A$ can be done in $\Theta(n)$ time using a divide-and-conquer algorithm.

The function `RANDOMIZED-SELECT`:

- a divide-and-conquer algorithm,
- uses `RANDOMIZED-PARTITION` from the quicksort algorithm, and
- recurse on one side of the partition only.
RANDOMIZED-SELECT Procedure

1. RANDOMIZED-SELECT(A, p, r, i)
2. \[\text{if } p = r\]
3. \[\text{then return } A[p]\]
4. \[q \leftarrow \text{RANDOMIZED-PARTITION}(A, p, r)\]
5. \[k \leftarrow q - p + 1\]
6. \[\text{if } i = k \quad /* \text{the pivot value is the answer */}\]
7. \[\text{then return } A[q]\]
8. \[\text{elseif } i < k\]
9. \[\text{then return } \text{RANDOMIZED-SELECT}(A, p, q - 1, i)\]
10. \[\text{else return } \text{RANDOMIZED-SELECT}(A, q, r, i - k)\]
RANDOMIZED-SELECT Procedure

To find the $i$th order statistic in $A[p…q-1]$

To find the $(i-k)$th order statistic in $A[q+1…r]$
RANDOMIZED-SELECT

• Find the $i^{th}$ (let $i = 6$ here) smallest element from the given array.

7 11 16 6 14 4 3 12

3 6 4 7 14 16 11 12

In sorted array the 6$^{th}$ element is 12

First element in the array is considered as the pivot here only for illustration purpose.

Required element in this sub-array is at $i-k = 2$, new value of $i = 2$

Required element is in this sub-array is at $i = 2$

Return 12 as the $i = k = 2$
Algorithm analysis

The **worst case**: always recurse on a subarray that is only one element smaller than the previous subarray.

\[ T(n) = T(n - 1) + \Theta(n) \]
\[ = \Theta(n^2) \]

The **best case**: always recurse on a subarray that has half of the elements smaller than the previous subarray.

\[ T(n) = T(n/2) + \Theta(n) \]
\[ = \Theta(n) \]
The average case:

- It is necessary to show that $T(n) = \Theta(n)$.
- For $1 \leq k \leq n$, the probability that the subarray $A[p .. q]$ has $k$ elements is $1/n$.
- To obtain an upper bound, it is assumed that $T(n)$ is monotonically increasing and that the $i$th smallest element is always in the larger subarray.
- So, we get

$$T(n) \leq \frac{1}{n} \sum_{k=1}^{n} (T(\max(k-1, n-k) + O(n))).$$
Algorithm analysis

\[ T(n) \leq \frac{1}{n} \sum_{k=1}^{n} (T(\max(k-1, n-k) + O(n)) = \frac{1}{n} \sum_{k=1}^{n} (T(\max(k-1, n-k))) + O(n) \]

\[ \leq \frac{2}{n} \sum_{k=n/2}^{n-1} T(n-k) + O(n). \]

\[ \because \max(k-1, n-k) = \begin{cases} 
  k & \text{if } k > \lceil n/2 \rceil \\
  n-k & \text{if } k \leq \lceil n/2 \rceil 
\end{cases} \]

<table>
<thead>
<tr>
<th>k</th>
<th>1</th>
<th>2</th>
<th>...</th>
<th>\lceil n/2 \rceil</th>
<th>\lceil n/2 \rceil + 1</th>
<th>...</th>
<th>n-1</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>\max(k-1, n-k)</td>
<td>n-1</td>
<td>n-2</td>
<td>...</td>
<td>n- \lceil n/2 \rceil</td>
<td>\lceil n/2 \rceil</td>
<td>...</td>
<td>n-2</td>
<td>n-1</td>
</tr>
</tbody>
</table>

Each term from \( T(\lceil n/2 \rceil) \) to \( T(n-1) \) appears twice.

**Conclusion:** The expected running time of the selection algorithm is \( O(n) \) when using random partitioning.