Searching and Sorting Algorithms, Complexity Analysis

Searching Algorithms

– General definition
  • Locate an element $x$ in a list of distinct elements $a_1, a_2, \ldots, a_n$, or determine that it is not in the list

– Linear search

**Algorithm 2: Linear search**

*Input*: unsorted sequence $a_1, a_2, \ldots, a_n$

*position of target value $x$*

*Output*: subscript of entry equal to target value; 0 if not found

*Initialize*: $i \leftarrow 1$

*while* $(i \leq n$ and $x \neq a_i)$

  $i \leftarrow i + 1$

*if* $i \leq n$ then

  *location* $\leftarrow i$

*else*

  *location* $\leftarrow 0$


Searching Algorithms

– Binary search

  • Requires input sequence to be sorted

**General Idea:**

Find 9 in the sequence: $<1, 4, 6, 9, 10, 14>$

left mid right

$\downarrow \downarrow \downarrow$

$<1, 4, 6, 9, 10, 14>$

9 > $a_{\text{mid}}$?

Yes ($9 > 6$)

9 > $a_{\text{mid}}$?

No ($9 < 10$)

$\downarrow \downarrow \downarrow$

$<1, 4, 6, 9, 10, 14>$

left mid right

$\downarrow \downarrow \downarrow$

$<1, 4, 6, 9, 10, 14>$

Searching Algorithms

– Binary search

**Algorithm 3: Binary search**

*Input*: sorted sequence $a_1, a_2, \ldots, a_n$

*position of target value $x$*

*Output*: subscript of entry equal to target value; 0 if not found

*Initialize*: $left \leftarrow 1$; $right \leftarrow n$

*while* $(left < right)$

  $mid \leftarrow \lfloor (left + right) / 2 \rfloor$

  *if* $x > a_{\text{mid}}$ then

    $left \leftarrow mid + 1$

  *else* $right \leftarrow mid$

*if* $x = a_{\text{left}}$ then

  *location* $\leftarrow left$

*else* $location \leftarrow 0$
Complexity Analysis

- Usually time complexity considered
- Space complexity can also be considered
- RAM Model
  - Constant time basic operations (add, sub, load, store…)
- Worst-case complexity measure
  - Estimates the time required for the most time-consuming input of each size
- Average-case complexity measure
  - Estimates the average time required for input of each size
  - Not always easy to see what is the average-case

Algorithm 2: Linear search

Initialize: \( i \leftarrow 1 \)

while \((i \leq n \text{ and } x \neq a_i)\)

\( i \leftarrow i + 1 \)

if \( i \leq n \) then location \( \leftarrow i \) else location \( \leftarrow 0 \)

Worst-case: element is at the end of the sequence or is not present

\( n \) iterations of loop \( c_1 + c_2 \Rightarrow \Theta(n) \)

Avrg-case: element is at the middle of the sequence

\( n/2 \) iterations of loop \( c_1 + c_2 \Rightarrow \Theta(n) \)

Algorithm 3: Binary search

Initialize: \( \text{left} \leftarrow 1; \text{right} \leftarrow n \)

while \((\text{left} < \text{right})\)

\( \text{mid} \leftarrow (\text{left} + \text{right}) / 2 \)

if \( x > a_{\text{mid}} \) then \( \text{left} \leftarrow \text{mid} + 1 \) else \( \text{right} \leftarrow \text{mid} \)

if \( x = a_{\text{mid}} \) then location \( \leftarrow \text{left} \) else location \( \leftarrow 0 \)

Average & worst-case analysis:

\( k \) iteration of while loop \( c_1 + c_2 \Rightarrow \Theta(k) = \Theta(\log_2 n) \)

Sorting Algorithms

- General definition
  - Putting a number of elements into a list in which the elements are in increasing order
- Input:
  - A sequence of \( n \) numbers \(<a_1, a_2, \ldots, a_n>\)
- Output:
  - A permutation (reordering) \(<a'_1, a'_2, \ldots, a'_n>\) of the input sequence such that \( a'_1 \leq a'_2 \leq \ldots \leq a'_n \)
Insertion sort

**General idea:**
- Same idea as what you do when cards are distributed
- Your left hand is initially empty
- Until all cards have been distributed
  - Take a card with right hand and insert it at the right position in left hand

**Algorithm 3: Insertion sort**

```plaintext
Input: unsorted sequence \(a_1, a_2, \ldots, a_n\)
Output: sorted sequence \(a_1, a_2, \ldots, a_n\)

for \(j \leftarrow 2\) to \(n\)
  distribute all cards
  key \(\leftarrow a_j\)
  \(i \leftarrow j - 1\)
  while \(i > 0\) and \(a_i > a_j\)
    insert at right position in left hand
    \(a_{i+1} \leftarrow a_i\)
    \(i \leftarrow i - 1\)
  \(a_j \leftarrow key\)
```

**Insertion Sort - example**

1. \(<3, 18, 4, 10, 7>\)
2. \(<3, 4, 18, 10, 7>\)
3. \(<3, 4, 7, 10, 18>\)

**Correctness proof of insertion sort**

- Must prove that:
  - The algorithm terminates
  - The algorithm sorts the input sequence

- Termination:
  - All operations of the algorithm take a finite amount of time
  - The algorithm executes a bounded number of loop iterations
Correctness proof of insertion sort

– The algorithm sorts the input sequence

Algorithm 3: Insertion sort
for \( j \leftarrow 2 \) to \( n \)
  \[
  \begin{align*}
  \text{key} &\leftarrow a_j \\
  i &\leftarrow j - 1 \\
  \text{while } i > 0 \text{ and } a_i > a_j \\
  &\quad a_{i+1} \leftarrow a_i \\
  &\quad i \leftarrow i - 1 \\
  a_i &\leftarrow \text{key}
  \end{align*}
  \]

• Prove a loop invariant:
  – \( a_1 \ldots a_j \) sorted at end of iteration \( j \)

Correctness proof of insertion sort

– Base step \((j = 2)\): \( a_1 \ldots a_2 \)
  • Property holds by condition of while loop and last line

Correctness proof of insertion sort

– Prove loop invariant by induction
  • Base step: \( j = 2 \)
  • Induction step: if true for \( 2 \leq k < j \) then true for \( j \)

Correctness proof of insertion sort

– Induction step:
  • Property holds by condition of while loop and last line
Correctness proof of insertion sort

– We proved that \( a_1 \ldots a_j \) is sorted at end of iteration \( j \)

– The last iteration is when \( j = n \)

– Consequently, when the algorithm terminates, \( a_1 \ldots a_j \) is sorted

Sorting Algorithms

– Bubble Sort

Algorithm 4: Bubble sort

\[\text{Input:} \text{ unsorted sequence} \quad a_1, a_2, \ldots, a_n \]
\[\text{Output:} \text{ sorted sequence} \quad a_1, a_2, \ldots, a_n \]

\[\text{for } i \leftarrow 1 \text{ to } n \]
\[\text{for } j \leftarrow n \text{ to } i + 1 \]
\[\text{if } a_j < a_{j-1} \text{ then interchange } a_j \text{ and } a_{j-1} \]

Bubble sort - example

\[\begin{align*}
&\text{i} \\
&\uparrow \\
&\quad <18, 4, 10, 7> \\
&\quad \downarrow \\
&\quad <4, 18, 7, 10> \\
&\quad \downarrow \\
&\quad <4, 7, 18, 10> \\
&\quad \downarrow \\
&\quad <4, 7, 10, 18> \\
&\quad \downarrow \\
&\quad <4, 7, 10, 18>
\end{align*}\]

Complexity Analysis

Algorithm 3: Insertion sort

\[\text{for } j \leftarrow 2 \text{ to } n \]
\[\text{key} \leftarrow a_j \]
\[j \leftarrow j - 1 \]
\[\text{while } i > 0 \text{ and } a_i > a_{i+1} \]
\[a_{i+1} \leftarrow a_i \]
\[i \leftarrow i - 1 \]
\[a_i \leftarrow \text{key} \]

Worst-case: input is sorted in reverse order

\[\sum_{j=1}^{n} \sum_{i=1}^{j} c = c + 2c + \ldots + (n - 1)c = \frac{(n - 1)c}{2} \]
\[=> \Theta(n^2)\]
Complexity Analysis

Algorithm 3: Insertion sort

```
for j ← 2 to n
    key ← a[j]
    i ← j - 1
    while i > 0 and a[i] > a[j]
        a[i+1] ← a[i]
        i ← i - 1
    a[i] ← key
```

Average-case: half of the elements in $a_1$..$a_{j-1}$ are greater than $a_j$

\[
\sum_{i=2}^{\lfloor \frac{j}{2}\rfloor} \sum_{c=1}^{i} (c + c + 2c + ...) \approx \left( \frac{n-1}{2} \right) \left( \frac{n+1}{2} \right) \frac{n-1}{2} \Rightarrow \Theta(n^2)
\]

Complexity Analysis

Algorithm 4: Bubble sort

```
for i ← 1 to n
    for j ← n to i+1
        if $a_j < a_{j-1}$ then interchange $a_j$ and $a_{j-1}$
```

Average & worst-case:

\[
\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c = (n-1)c + (n-2)c + ... + c = \frac{(n-1)^2}{2} c \Rightarrow \Theta(n^2)
\]

Summary

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Avrg-case</th>
<th>Worst-case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear search</td>
<td>$\Theta(n)$</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td>Binary search</td>
<td>$\Theta(\log_2(n))$</td>
<td>$\Theta(\log_2(n))$</td>
</tr>
<tr>
<td>Insertion sort</td>
<td>$\Theta(n^2)$</td>
<td>$\Theta(n^2)$</td>
</tr>
<tr>
<td>Bubble sort</td>
<td>$\Theta(n^2)$</td>
<td>$\Theta(n^2)$</td>
</tr>
</tbody>
</table>

Complexity of algorithms

– Common terminology

<table>
<thead>
<tr>
<th>Complexity</th>
<th>Terminology</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(1)$</td>
<td>Constant complexity</td>
</tr>
<tr>
<td>$O(\log n)$</td>
<td>Logarithmic complexity</td>
</tr>
<tr>
<td>$O(n)$</td>
<td>Linear complexity</td>
</tr>
<tr>
<td>$O(n \log n)$</td>
<td>$n \log n$ complexity</td>
</tr>
<tr>
<td>$O(n^p)$</td>
<td>Polynomial complexity</td>
</tr>
<tr>
<td>$O(b^n), b &gt; 1$</td>
<td>Exponential complexity</td>
</tr>
<tr>
<td>$O(n!)$</td>
<td>Factorial complexity</td>
</tr>
</tbody>
</table>