Data Structures and Algorithms
(資料結構與演算法)
Lecture 2: Data Structure

Hsuan-Tien Lin (林軒田)
htlin@csie.ntu.edu.tw

Department of Computer Science
& Information Engineering
National Taiwan University
(國立台灣大學資訊工程系)
1 the one where it all began

Lecture 1: Algorithm

clearly-illustrated instructions to provably solve a computational task

Lecture 2: Data Structure

- definition of data structure
- ordered array as data structure
- GET (search) in ordered array
- why data structures and algorithms

2 the data structures awaken

3 fantastic trees and where to find them

4 the search revolutions

5 sorting: the final frontier
definition of data structure
Data Structure

Definition of data structure

From Cloth Structure to Data Structure

Cloth Structure: Ordered

Cloth Structure: Messy

Data Structure: Sorted

Data Structure: Unsorted

Data structure: scheme of organizing data within computer
Good Algorithm Needs Proper Data Structure

**SELECTION-SORT** with **GET-MIN-INDEX**, remember? :-)

### SELECTION-SORT($A$)

1. for $i = 1$ to $A$.length
2. \[ m = \text{GET-MIN-INDEX}(A, i, A.\text{length}) \]
3. \[ \text{SWAP}(A[i], A[m]) \]
4. return $A$ // which has been sorted in place

### GET-MIN-INDEX($A, \ell, r$)

1. \[ m = \ell \] \( // \) store current min. index
2. for $i = \ell + 1$ to $r$
3. \[ \text{// update if } i\text{-th element smaller} \]
4. if $A[m] > A[i]$
5. \[ m = i \]
6. return $m$

if having data structure with faster **GET-MIN-INDEX**,

\[ \rightarrow \text{SELECTION-SORT} \] also faster (to be taught)

algorithm :: data structure

\[ \sim \] recipe :: ingredient structure
## Data Structure Needs Accessing Algorithms

### GET
- **GET-BY-INDEX**: for arrays
- **GET-NEXT**: for sequential access
- **GET(item)**: for search
- ...

—generally assume to **read** without deleting

### INSERT
- **INSERT-BY-INDEX**: for arrays
- **INSERT-AFTER**: for sequential access
- **INSERT(item)**
- ...

—generally assume to **add without overriding**

### ‘philosophical’ rule of thumb:
often-**GET** ⇔ **INSERT** “nearby”
### Data Structure Needs Maintenance Algorithms

<table>
<thead>
<tr>
<th>Construct</th>
<th>Remove</th>
<th>Update</th>
</tr>
</thead>
</table>
| • baseline: with multiple **INSERT**  
• often **faster** if designed carefully & strategically | • often viewed as deleting **after GET**  
• **UNINSERT**: often harder than **INSERT** | • usually possible with **REMOVE + INSERT**  
• can be viewed as **INSERT** with overriding |

**hidden cost of data structure:**  
maintenance effort (especially **REMOVE & UPDATE**)

---

*Data Structure*  
*definition of data structure*
Fun Time

Which of the following can be viewed as the reverse algorithm of `INSERT` within a data structure?

1. **CONSTRUCT**
2. **GET**
3. **REMOVE**
4. **UPDATE**

Reference Answer: 3

- **REMOVE** -ing an item from the data structure essentially takes out what has been **INSERT**-ed.
Fun Time

Which of the following can be viewed as the reverse algorithm of \textsc{Insert} within a data structure?

1. \textsc{Construct}
2. \textsc{Get}
3. \textsc{Remove}
4. \textsc{Update}

Reference Answer: 3

\textsc{Remove}-ing an item from the data structure essentially takes out what has been \textsc{Insert}-ed.
ordered array as data structure
### Definition of Ordered Array

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>3♥️</td>
<td>5♥️</td>
<td>10♥️</td>
<td>Q♥️</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

An array of **consecutive** elements with **ordered** values.
**INSERT of Ordered Array**

### Swap Version

**INSERT**(*)\(A, data\)

1. \(n = A.\text{length}\)
2. \(A[n + 1] = data\) // put in the back
3. for \(i = n\) downto 1
4. if \(A[i + 1] < A[i]\)
5. \(\text{SWAP}(A[i], A[i + 1])\) // cut in
6. else
7. return

### Direct Cut-in Version

**INSERT**(*)\(A, data\)

1. \(i = A.\text{length}\)
2. while \(i > 0\) and \(A[i] > data\)
3. \(A[i + 1] = A[i]\)
4. \(i = i - 1\)
5. \(A[i + 1] = data\)
6. return

**INSERT of ordered array:** cut in from **back**
CONSTRUCT of Ordered Array

**SELECTION-SORT, remember? :-)**

**SELECTION-SORT**(*A*)

1. **for** *i* = 1 **to** *A*.length
2. 
3. 
4. **return** *A*

**GET-MIN-INDEX**(*A*, ℓ, *r*)

1. 
2. **for** *i* = ℓ + 1 **to** *r*
3. 
4. 
5. **return** *m*

**or INSERTION-SORT**

**INSERTION-SORT**(*A*)

1. **for** *i* = 1 **to** *A*.length
2. **INSERT**(*A*, *i*)
3. 
4. **return** *A*

**INSERT**(*A*, *m*)

1. 
2. 
3. **while** *i* > 0 and *A*[i] > *data*
4. 
5. 
6. *A*[i + 1] = *data*

**INSERTION-SORT: CONSTRUCT** with multiple **INSERT**
REMOVE and UPDATE of Ordered Array

**REMOVE**

REMOVE \((A, m)\)

1. \(i = m + 1\)
2. while \(i < A\. length\)
   3. \(A[i - 1] = A[i] \quad \text{// fill in}\)
   4. \(i = i + 1\)
3. \(A\. length = A\. length - 1\)

**UPDATE**

UPDATE \((A, m, data)\)

1. \(i = m\)
2. if \(A[i] > data \quad \text{// cut in to front}\)
   3. \(i = i - 1\)
   4. while \(i > 0 \text{ and } A[i] > data\)
      5. \(A[i + 1] = A[i]\)
      6. \(i = i - 1\)
   7. \(A[i + 1] = data\)
   8. else \quad \text{// cut in to back}\)
   9. \(... \text{ complete on your own ...}\)

ordered array: more **maintenance efforts** than unordered

\[\implies\] faster GET (?)
Fun Time

Consider the direct cut-in version of `INSERT`. Assume that some `data` is inserted to an array `A` with `A.length = 6211` (prior to insertion) and ends up in position `A[1126]`. How many comparisons of the form `A[i] > data` has been conducted?

```
INSERT(A, data)
1  i = A.length
2  while i > 0 and A[i] > data
3    A[i + 1] = A[i]
4    i = i - 1
5  A[i + 1] = data
```

1 1126
2 5087
3 6211
4 7337
Fun Time

Consider the direct cut-in version of INSERT. Assume that some data is inserted to an array $A$ with $A.length = 6211$ (prior to insertion) and ends up in position $A[1126]$. How many comparisons of the form $A[i] > data$ has been conducted?

```
INSERT(A, data)
1  i = A.length
2  while i > 0 and A[i] > data
3    A[i+1] = A[i]
4    i = i - 1
5  A[i+1] = data
```

1 1126
2 5087
3 6211
4 7337

Reference Answer: 2

When data ends up in position $A[1126]$, $6212 - 1126$ elements are larger than data (pushed back within while). Another comparison with $A[1125]$ terminates while. So the total is $6212 - 1126 + 1 = 5087$. 

GET (search) in ordered array
Application: Book Search within (Digital) Library

GET book with ID as key in ordered array

figure by LaiAndrewKimmy,
licensed under CC BY-SA 3.0 via Wikimedia Commons
Sequential Search Algorithm for Any Array

**SEQ-SEARCH** \((A, \text{key}, \ell, r)\)

1. \(m = \ell \) // store current min. index
2. \(\text{for } i = \ell + 1 \text{ to } r\)
3.   // update if \(i\)-th element smaller
4.   if \(A[i]\) equals \text{key}
5.       return \(i\)
6. return NIL

**GET-MIN-INDEX** \((A, \ell, r)\)

1. \(m = \ell \) // store current min. index
2. \(\text{for } i = \ell + 1 \text{ to } r\)
3.   // update if \(i\)-th element smaller
4.   if \(A[m]\) > \(A[i]\)
5.       \(m = i\)
6. return \(m\)

**SEQ-SEARCH**: structurally similar to **GET-MIN-INDEX**
Ordered Array: Sequential Search with Shortcut

\[
\begin{array}{|c|c|c|c|c|c|c|c|}
\hline
& 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
\text{original} & \text{3} & \text{4} & \text{5} & \text{7} & \text{9} & \text{10} & \text{Q} \\
\hline
i = 1 & \text{3} & \text{4} & \text{5} & \text{7} & \text{9} & \text{10} & \text{Q} \\
\hline
i = 2 & \text{3} & \text{4} & \text{5} & \text{7} & \text{9} & \text{10} & \text{Q} \\
\hline
i = 3 & \text{3} & \text{4} & \text{5} & \text{7} & \text{9} & \text{10} & \text{Q} \\
\hline
i = 4 & \text{3} & \text{4} & \text{5} & \text{7} & \text{9} & \text{10} & \text{Q} \\
\hline
\end{array}
\]

**SEQ-SEARCH-SHORTCUT**(A, key, \( \ell \), r)

1. \textbf{for} \( i = \ell \) \textbf{to} r
2. \hspace{1em} // return when found
3. \hspace{1em} \textbf{if} A[i] equals key
4. \hspace{2em} return i
5. \hspace{1em} \textbf{elseif} A[i] > key
6. \hspace{2em} return NIL
7. return NIL

**SEQ-SEARCH**(A, key, \( \ell \), r)

1. \textbf{for} \( i = \ell \) \textbf{to} r
2. \hspace{1em} // return when found
3. \hspace{1em} \textbf{if} A[i] equals key
4. \hspace{2em} return i
5. \hspace{1em} \textbf{elseif} A[i] > key
6. \hspace{2em} return NIL
7. return NIL

ordered: possibly easier to declare NIL
## Ordered Array: Binary Search Algorithm

### Bin-Search Algorithm

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>original</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>9</td>
<td>10</td>
<td>Q</td>
</tr>
<tr>
<td>[1, 7]</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>9</td>
<td>10</td>
<td>Q</td>
</tr>
<tr>
<td>[1, 3]</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>9</td>
<td>10</td>
<td>Q</td>
</tr>
<tr>
<td>[3, 3]</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>9</td>
<td>10</td>
<td>Q</td>
</tr>
</tbody>
</table>

**Bin-Search**($A$, $key$, $\ell$, $r$)

1. while $\ell \leq r$
2. $m = \text{floor}((\ell + r)/2)$
3. if $A[m]$ equals $key$
   4. return $m$
5. elseif $A[m] > key$
   6. $r = m - 1$ // cut out end
7. elseif $A[m] < key$
   8. $\ell = m + 1$ // cut out begin
9. return NIL

**Seq-Search-Shortcut**($A$, $key$, $\ell$, $r$)

1. for $i = \ell$ to $r$
2. // return when found
3. if $A[i]$ equals $key$
4. return $i$
5. elseif $A[i] > key$
6. return NIL
7. return NIL

**Bin-Search**: multiple shortcuts by quickly checking the middle
**Binary Search in Open Source**

**BIN-SEARCH**($A$, $key$, $ℓ$, $r$)

1. while $ℓ ≤ r$
2. $m = \text{floor}((ℓ + r)/2)$
3. if $A[m]$ equals $key$
   4. return $m$
4. elseif $A[m] > key$
   5. $r = m - 1$ // cut out end
5. elseif $A[m] < key$
   6. $ℓ = m + 1$ // cut out begin
7. return NIL

---

**“must-know” for programmers**

```java
public static int binarySearch(int[] a, int key) {
    int low = 0;
    int high = a.length - 1;

    while (low <= high) {
        int mid = (low + high) >>> 1;
        int midVal = a[mid];

        if (midVal < key)
            low = mid + 1;
        else if (midVal > key)
            high = mid - 1;
        else
            return mid;
    }
    return -(low + 1);
}
```
Consider running the **BIN-SEARCH** algorithm on an ordered array of size 15 with some *key* that is not in the array. How many comparisons does **BIN-SEARCH** take before returning **NIL**?

1. 1
2. 2
3. 4
4. 15

Reference Answer:

3

The first comparison is a shortcut that leaves only 7 remaining elements; the second leaves 3; the third leaves 1; the fourth eliminates all possibilities.
Consider running the `BIN-SEARCH` algorithm on an ordered array of size 15 with some key that is not in the array. How many comparisons does `BIN-SEARCH` take before returning `NIL`?

1. 1
2. 2
3. 4
4. 15

Reference Answer: 3

The first comparison is a shortcut that leaves only 7 remaining elements; the second leaves 3; the third leaves 1; the fourth eliminates all possibilities.
why data structures and algorithms
Why Data Structures and Algorithms?

good program: proper use of resources

Space Resources
- memory
- disk(s)
- transmission bandwidth
—usually cared by data structure

Computation Resources
- CPU(s)
- GPU(s)
- computation power
—usually cared by algorithm

Other Resources
- manpower
- budget
—usually cared by management

data structures and algorithms: for writing good program
Proper Use: Trade-off of Different Factors

- faster GET \iff slower INSERT and/or maintenance
- more space \iff faster computation
- harder to implement/debug \iff faster computation

**good program** needs understanding trade-off
# Programming ≠ Coding

programming :: building house \(\sim\) coding :: construction work

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Introduction to C</th>
<th>Data Structures and Algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis</td>
<td>simple</td>
<td>simple</td>
</tr>
<tr>
<td>Design</td>
<td>simple</td>
<td>simple</td>
</tr>
<tr>
<td>Coding</td>
<td>⋆</td>
<td>⋆</td>
</tr>
<tr>
<td>Proof</td>
<td>none</td>
<td>⋅</td>
</tr>
<tr>
<td>Test</td>
<td>simple</td>
<td>⋅</td>
</tr>
<tr>
<td>Debug</td>
<td>⋆</td>
<td>⋅</td>
</tr>
</tbody>
</table>

Data structures and algorithms:
moving from **coding** to **programming**
Fun Time

Which of the following is a property of an ordered array when compared with an unordered one with the same number of elements?

1. faster GET
2. faster INSERT
3. more space
4. none of the other choices

Reference Answer:
1. An ordered array allows faster GET by Binary Search.
Which of the following is a property of an ordered array when compared with an unordered one with the same number of elements?

1. faster GET
2. faster INSERT
3. more space
4. none of the other choices

Reference Answer: 1

An ordered array allows faster GET by BIN-SEARCH.
Lecture 2: Data Structure

- definition of data structure
- organize data with access/maintenance algorithms
- ordered array as data structure
  - insert by cut-in, remove by fill-in
- GET (search) in ordered array
  - binary search using order for shortcuts
- why data structures and algorithms
  - study trade-off to move from coding to programming

• next: tools for analyzing/studying trade-off