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)

Roadmap

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

From Cloth Structure to Data Structure

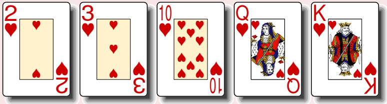
Cloth Structure: Ordered

(copyright purchased from iStock)

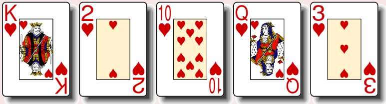
Cloth Structure: Messy

(copyright purchased from iStock)

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)
for i = 1 to A.length
    m = GET-MIN-INDEX(A, i, A.length)
    SWAP(A[i], A[m])
return A // which has been sorted in place
```

```
GET-MIN-INDEX(A, ℓ, r)
m = ℓ // store current min. index
for i = ℓ + 1 to r
    // update if i-th element smaller
    if A[m] > A[i]
        m = i
return m
```

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

$\Longrightarrow$ 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 $\Longleftrightarrow$ INSERT "nearby"

Data Structure Needs Maintenance Algorithms

CONSTRUCT

- baseline: with multiple INSERT
- often faster if designed carefully & strategically

REMOVE

- often viewed as deleting after GET
- $\sim$ UNINSERT: often harder than INSERT

UPDATE

- usually possible with REMOVE + INSERT
- can be viewed as INSERT with overriding

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

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

Fun Time

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

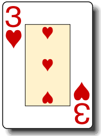
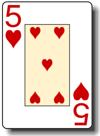
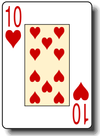
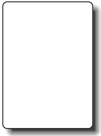
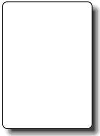
1. CONSTRUCT
2. GET
3. REMOVE
4. UPDATE

Reference Answer: 3

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

ordered array as data structure

Definition of Ordered Array

1	2	3	4	5	6
					
$A[1]$	$A[2]$	$A[3]$	$A[A.length]$		

an array of consecutive elements with ordered values

INSERT of Ordered Array

Swap Version

INSERT(*A*, *data*)

```
n = A.length
A.[n + 1] = data // put in the back
for i = n downto 1
    if A[i + 1] < A[i]
        SWAP(A[i], A[i + 1]) // cut in
    else
        return
```

6♥	1	2	3	4	5	6
original	3♥	5♥	10♥	Q♥	6♥	
$i = 4$	3♥	5♥	10♥	6♥	Q♥	
$i = 3$	3♥	5♥	6♥	10♥	Q♥	
return	3♥	5♥	6♥	10♥	Q♥	

Direct Cut-in Version

INSERT(*A*, *data*)

```
i = A.length
while i > 0 and A[i] > data
    A[i + 1] = A[i]
    i = i - 1
A[i + 1] = data
```

6♥	1	2	3	4	5	6
original	3♥	5♥	10♥	Q♥		
$i = 4$	3♥	5♥	10♥	Q♡	Q♥	
$i = 3$	3♥	5♥	10♡	10♥	Q♥	
return	3♥	5♥	6♥	10♥	Q♥	

INSERT of ordered array: cut in from back

CONSTRUCT of Ordered Array

SELECTION-SORT, remember? :-)

```
SELECTION-SORT(A)
for i = 1 to A.length
    m = GET-MIN-INDEX(A, i, A.length))
    SWAP(A[i], A[m])
return A
```

```
GET-MIN-INDEX(A, ℓ, r)
m = ℓ // store current min. index
for i = ℓ + 1 to r
    // update if i-th element smaller
    if A[m] > A[i]
        m = i
return m
```

or INSERTION-SORT

```
INSERTION-SORT(A)
for i = 1 to A.length
    INSERT(A, i)

return A
```

```
INSERT(A, m)
data = A[m]
i = m − 1
while i > 0 and A[i] > data
    A[i + 1] = A[i]
    i = i − 1
A[i + 1] = data
```

INSERTION-SORT: CONSTRUCT with multiple INSERT

REMOVE and UPDATE of Ordered Array

REMOVE

REMOVE(A, m)

```
i = m + 1
while i < A.length
    A[i − 1] = A[i] // fill in
    i = i + 1
A.length = A.length − 1
```

UPDATE

UPDATE(A, m, $data$)

```
i = m
if A[i] > data // cut in to front
    i = i − 1
    while i > 0 and A[i] > data
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = data
else // cut in to back
    ... complete on your own ...
```

ordered array: more maintenance efforts than unordered
$\Longrightarrow$ 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)
i = A.length
while i > 0 and A[i] > data
    A[i + 1] = A[i]
    i = i − 1
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)
i = A.length
while i > 0 and A[i] > data
        A[i + 1] = A[i]
        i = i − 1
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

figure by LaiAndrewKimmy,

licensed under CC BY-SA 3.0 via Wikimedia Commons

GET book with ID as key in ordered array

Sequential Search Algorithm for Any Array

5♥	1	2	3	4	5	6	7
original	3♥	4♥	5♥	7♥	9♥	10♥	Q♥
$i = 1$	3♥	4♥	5♥	7♥	9♥	10♥	Q♥
$i = 2$	3♥	4♥	5♥	7♥	9♥	10♥	Q♥
$i = 3$	3♥	4♥	**5♥**	7♥	9♥	10♥	Q♥

```
SEQ-SEARCH(A, key, ℓ, r)

for i = ℓ to r
    // return when found
    if A[i] equals key
        return i
return NIL
```

```
GET-MIN-INDEX(A, ℓ, r)
m = ℓ // store current min. index
for i = ℓ + 1 to r
    // update if i-th element smaller
    if A[m] > A[i]
        m = i
return m
```

SEQ-SEARCH: structurally similar to GET-MIN-INDEX

Ordered Array: Sequential Search with Shortcut

6♥	1	2	3	4	5	6	7
original	3♥	4♥	5♥	7♥	9♥	10♥	Q♥
$i = 1$	3♥	4♥	5♥	7♥	9♥	10♥	Q♥
$i = 2$	3♥	4♥	5♥	7♥	9♥	10♥	Q♥
$i = 3$	3♥	4♥	5♥	7♥	9♥	10♥	Q♥
$i = 4$	3♥	4♥	5♥	7♥	9♥	10♥	Q♥

SEQ-SEARCH-SHORTCUT(A, key, ℓ, r)

```
for i = ℓ to r
    // return when found
    if A[i] equals key
        return i
    elseif A[i] > key
        return NIL
return NIL
```

SEQ-SEARCH(A, key, ℓ, r)

```
for i = ℓ to r
    // return when found
    if A[i] equals key
        return i


return NIL
```

ordered: possibly easier to declare NIL

Ordered Array: Binary Search Algorithm

6♥	1	2	3	4	5	6	7
original	3♥	4♥	5♥	7♥	9♥	10♥	Q♥
$[1, 7]$	3♥	4♥	5♥	7♥	9♥	10♥	Q♥
$[1, 3]$	3♥	4♥	5♥	7♥	9♥	10♥	Q♥
$[3, 3]$	3♥	4♥	5♥	7♥	9♥	10♥	Q♥

BIN-SEARCH(A, key, ℓ, r)

```
while ℓ ≤ r
    m = floor((ℓ + r)/2)
    if A[m] equals key
        return m
    elseif A[m] > key
        r = m − 1 // cut out end
    elseif A[m] < key
        ℓ = m + 1 // cut out begin
return NIL
```

SEQ-SEARCH-SHORTCUT(A, key, ℓ, r)

```
for i = ℓ to r
    // return when found
    if A[i] equals key
        return i
    elseif A[i] > key
        return NIL
return NIL
```

BIN-SEARCH: multiple shortcuts
by quickly checking the middle

Binary Search in Open Source

BIN-SEARCH(A, key, ℓ, r)

```
while ℓ ≤ r
    m = floor((ℓ + r)/2)
    if A[m] equals key
        return m
    elseif A[m] > key
        r = m − 1 // cut out end
    elseif A[m] < key
        ℓ = m + 1 // cut out begin
return NIL
```

"must-know" for programmers

java.util.Arrays

```java
private 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;
              // key found
    }
    return -(low + 1);
      // key not found.
}
```

Fun Time

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

Fun Time

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	$\Longleftrightarrow$	slower INSERT and/or maintenance
more space	$\Longleftrightarrow$	faster computation
harder to implement/debug	$\Longleftrightarrow$	faster computation

good program needs understanding trade-off

Programming $\neq$ Coding

programming :: building house $\sim$ coding :: construction work

	Introduction to C	Data Structures and Algorithms
requirement	simple	simple
analysis	simple	simple
design	simple	$\star$
coding	$\star$	●
proof	none	●
test	simple	$\star$
debug	$\star$	●

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

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 BIN-SEARCH.

Summary

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**