

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)



definition of data structure

From Cloth Structure to Data Structure

Cloth Structure: Ordered



draft POC from iStock

Cloth Structure: Messy



draft POC 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)

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

GET-MIN-INDEX(A, ℓ, r)

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

if having data structure with faster **GET-MIN-INDEX**,
 \implies **SELECTION-SORT** *also faster* (to be taught)

algorithm :: data structure
 \sim recipe :: material/kitchenware structure

Data Structure Needs Accessing Algorithms

GET

- GET-BY-INDEX: for arrays
- GET-NEXT: for sequential access
- GET(key): for search
- ...

—generally assume to **read**
without deleting

INSERT

- INSERT-BY-INDEX: for arrays
- INSERT-AFTER: for sequential access
- INSERT(key, data)
- ...

—generally assume to **add without**
overriding

rule of thumb:
often-**GET** \iff **place** “nearby”

Data Structure Needs Maintenance Algorithms

CONSTRUCT

- usually possible with multiple **INSERT**
- sometimes possible to be **faster** than so

REMOVE

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

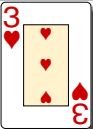
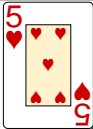
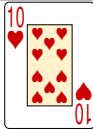



UPDATE

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

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

ordered array as data structure

Definition of Ordered Array

1	2	3	4	5	6
					

an array of **consecutive** elements with **ordered** values

INSERT of Ordered Array

Swap Version

INSERT(*A*, *data*)

```

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

	6	1	2	3	4	5	6
original	6♠	3♥	5♥	10♥	Q♥	6♠	
<i>i</i> = 4	6♠	3♥	5♥	10♥	6♠	Q♥	
<i>i</i> = 3	6♠	3♥	5♥	6♠	10♥	Q♥	
return	6♠	3♥	5♥	6♥	10♥	Q♥	

Direct Cut-in Version

INSERT(*A*, *data*)

```

1  key = data
2  i = A.length
3  while i > 0 and A[i] > key
4      A[i + 1] = A[i]
5      i = i - 1
6  A[i + 1] = key
7  return
```

	6	1	2	3	4	5	6
original	6♠	3♥	5♥	10♥	Q♥		
<i>i</i> = 4	6♠	3♥	5♥	10♥	Q♥	Q♥	
<i>i</i> = 3	6♠	3♥	5♥	10♥	10♥	Q♥	
return	6♠	3♥	5♥	6♥	10♥	Q♥	

INSERT: cut in from back

CONSTRUCT of Ordered Array

SELECTION-SORT

SELECTION-SORT(A)

```
1  for  $i = 1$  to  $A.length$ 
2       $m = \text{GET-MIN-INDEX}(A, i, A.length)$ 
3      SWAP( $A[i], A[m]$ )
4  return  $A$ 
```

GET-MIN-INDEX(A, ℓ, r)

```
1   $m = \ell$  // store current min. index
2  for  $i = \ell + 1$  to  $r$ 
3      // update if  $i$ -th element smaller
4      if  $A[m] > A[i]$ 
5           $m = i$ 
6  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   $key = A[m]$ 
2   $i = m - 1$ 
3  while  $i > 0$  and  $A[i] > key$ 
4       $A[i + 1] = A[i]$ 
5       $i = i - 1$ 
6   $A[i + 1] = key$ 
```

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       $A[i - 1] = A[i]$  // fill in
4       $i = i + 1$ 
5   $A.length = A.length - 1$ 
```

UPDATE

UPDATE($A, m, data$)

```
1   $key = data$ 
2   $i = m$ 
3  if  $A[i] > key$  // insert to front
4       $i = i - 1$ 
5      while  $i > 0$  and  $A[i] > key$ 
6           $A[i + 1] = A[i]$ 
7           $i = i - 1$ 
8       $A[i + 1] = key$ 
9  else // insert to back
10     (try to complete on your own)
```

ordered array: more maintenance efforts \Rightarrow faster GET (?)

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

	1	2	3	4	5	6	7
							
original							
$i = 1$							
$i = 2$							
$i = 3$							

SEQ-SEARCH(A, key, ℓ, r)

```

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

```

GET-MIN-INDEX(A, ℓ, r)





































```

1   $m = \ell$  // store current min. index
2  for  $i = \ell + 1$  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

	6	1	2	3	4	5	6	7
original								
$i = 1$								
$i = 2$								
$i = 3$								
$i = 4$								

SEQ-SEARCH-SHORTCUT(A, key, ℓ, 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

```

SEQ-SEARCH(A, key, ℓ, r)

```

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

```

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, ℓ, 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  $\ell \leq r$ 
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
```

“must-know” for programmers

java.util.Arrays

```
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.
}
```

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



slower INSERT
and/or maintenance

more space



faster computation

harder to implement/debug



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	★
coding	★	●
proof	none	●
test	simple	★
debug	★	●

data structures and algorithms:
moving from **coding** to **programming**

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