Arrays

Jianguo Lu

January 5, 2025

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	3	6	2	8	9	3	2	8	5	1	7	2	8	3	7

Overview

Last lecture

- This course is about how to write good (efficient) programs/algorithms
- It is fundamental, useful, and fun (yet to be confirmed)
- What is algorithm. Selection sort example
- what is data structure (ADT). binary search tree. heap.
- Why is efficiency important?
- How do we know whether an algorithm is efficient: algorithm analysis.
- How to design efficient algorithms: algorithm design paradigms.
- Should I take this course?
- How can I score high in this course?

Array

A basic data structure in any programming language.

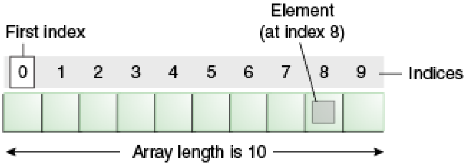
Same data type : An array is a sequenced collection of elements all of the **same** type.

Indexed : Each cell in an array has an index, which uniquely refers to the value stored in that cell.

- Loops are constructed using index
- Access by index is very efficient

Contiguous : A consecutive section of memory is allocated

Fixed length : Capacity of the array needs to be decided at the very beginning



Why indexing is efficient

Actual Address of the **1st** element of the array is known as **Base Address (B)**
Here it is 1100

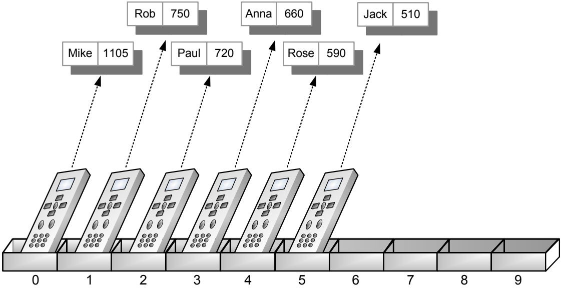
Memory space acquired by every element in the Array is called **Width (W)**
Here it is 4 bytes

Actual Address in the Memory	1100	1104	1108	1112	1116	1120
Elements	**15**	**7**	**11**	**44**	**93**	**20**
Address with respect to the Array (Subscript)	0	1	2	3	4	5

Lower Limit/Bound of Subscript **(LB)**

Example:Scoreboard

- Keep track of players and their best scores in an array, board
- The elements of board are objects of class GameEntry
- Array board is sorted by score



Create an Array

Method 1 use an assignment to a literal form when initially declaring the array, e.g.:

```
String[] names={"Rob","Mike","Rose","Jill"};
int[] scores = {750, 1105, 590, 740};
```

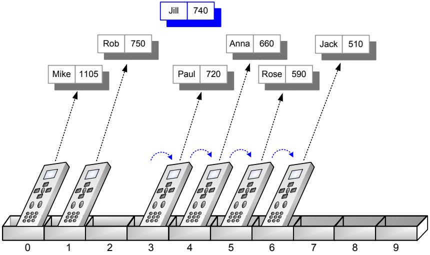
Method 2 use the **new** operator.

```
private GameEntry[] board=new GameEntry[capacity];
```

- Note that it is not a typical constructor. There is not an 'Array' class.

Add an entry

- To add e into array board at index i,
- Need to make room for it by shifting forward the $n - i$ entries $board[i], \ldots, board[n-1]$

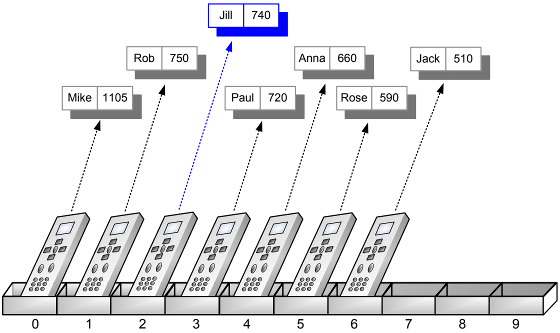


Add an element

```
public void add(GameEntry e) {
  int newScore = e.getScore();
  ...
  if { newScore>board[numEntries-1].getScore()) {
      int j = numEntries - 1;
      while (j > 0 && newScore>board[j-1].getScore()) {
        board[j] = board[j-1];
        j--;
      }
      board[j] = e;
    }
  }
```

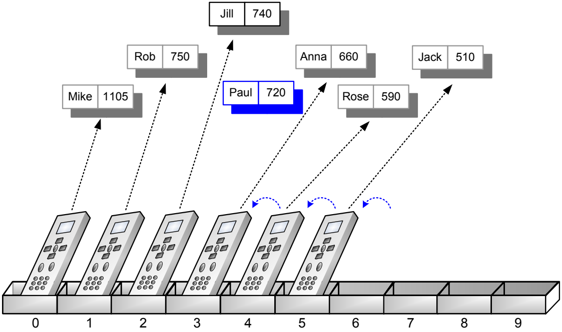
The cost of insertion operation is high

Remove an element



Remove operation

- To remove the entry e at index i, we need to fill the hole left by e by shifting backward the $n - i - 1$ elements



Remove an element

```
public GameEntry remove(int i)  {
  GameEntry temp = board[i];
  for (int j = i; j < numEntries-1; j++)
    board[j] = board[j+1];
  board[numEntries-1] = null;
  numEntries--;
  return temp;
}
```

The cost of remove operation is high

Disadvantages of Array

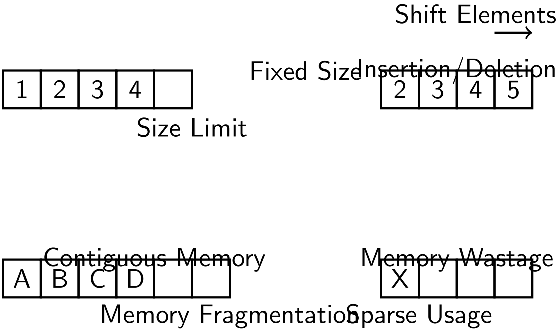
- Pre-allocate all needed memory up-front
  - waste memory space for cells not used
- Fixed-size–We may not know the size before hand
- One block allocation–empty memory space may be scattered/fragmental
- Not efficient for insert and remove operations–need to shift cells

Actual Address of the 1st element of the array is known as **Base Address (B)**
Here it is 1100

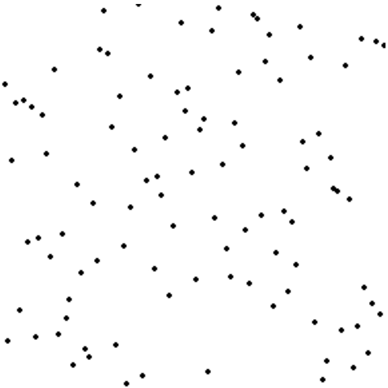
Memory space acquired by every element in the Array is called **Width (W)**
Here it is 4 bytes

Actual Address in the Memory	1100	1104	1108	1112	1116	1120
Elements	**15**	**7**	**11**	**44**	**93**	**20**
Address with respect to the Array (Subscript)	0	1	2	3	4	5

Lower Limit/Bound of Subscript **(LB)**


Shift Elements
1 2 3 4
Fixed Size
Insertion/Deletion
2 3 4 5
Size Limit
Contiguous Memory
A B C D
Memory Wastage
X
Memory Fragmentation
Sparse Usage

Selection Sorting Example



Selection Sorting Example

Algorithm 1: Selection Sort

Input: Array A of length n

Output: Sorted A

for *int i =0; i < n-1; i++* **do**

  min= minimal element in array[i+1:n];

  swap array[i] with min;

4	6	7	1	2	5	3
1	6	7	4	2	5	3

Selection Sorting Example

Algorithm 2: Selection Sort

Input: Array A of length n

Output: Sorted A

```
for int i =0; i < n-1; i++ do
    min= minimal element in array[i+1:n];
    swap array[i] with min;
```

4	6	7	1	2	5	3
1	6	7	4	2	5	3
1	2	7	4	6	5	3

Selection Sorting Example

Algorithm 3: Selection Sort

Input: Array A of length n
Output: Sorted A

```
for int i =0; i < n-1; i++ do
    min= minimal element in array[i+1:n];
    swap array[i] with min;
```

4	6	7	1	2	5	3
1	6	7	4	2	5	3
1	2	7	4	6	5	3
1	2	3	4	6	5	7

Selection Sorting Example

Algorithm 4: Selection Sort

Input: Array A of length n
Output: Sorted A

```
for int i =0; i < n-1; i++ do
    min= minimal element in array[i+1:n];
    swap array[i] with min;
```

4	6	7	1	2	5	3
1	6	7	4	2	5	3
1	2	7	4	6	5	3
1	2	3	4	6	5	7
1	2	3	4	6	5	7

Selection Sorting Example

Algorithm 5: Selection Sort

Input: Array A of length n
Output: Sorted A

for *int i =0; i < n-1; i++* **do**
    min= minimal element in array[i+1:n];
    swap array[i] with min;

4	6	7	1	2	5	3
1	6	7	4	2	5	3
1	2	7	4	6	5	3
1	2	3	4	6	5	7
1	2	3	4	6	5	7
1	2	3	4	5	6	7

Selection Sorting Example

Algorithm 6: Selection Sort

Input: Array A of length n
Output: Sorted A

```
for int i =0; i < n-1; i++ do
    min= minimal element in array[i+1:n];
    swap array[i] with min;
```

4	6	7	1	2	5	3
1	6	7	4	2	5	3
1	2	7	4	6	5	3
1	2	3	4	6	5	7
1	2	3	4	6	5	7
1	2	3	4	5	6	7
1	2	3	4	5	6	7

Selection Sorting Example

Algorithm 7: Selection Sort

Input: Array A of length n

Output: Sorted A

for *int i =0; i < n-1; i++* **do**

  min= minimal element in array[i+1:n];

  swap array[i] with min;

4	6	7	1	2	5	3
1	6	7	4	2	5	3
1	2	7	4	6	5	3
1	2	3	4	6	5	7
1	2	3	4	6	5	7
1	2	3	4	5	6	7
1	2	3	4	5	6	7

Number of Comparisons of Selection Sort

4	6	7	1	2	5	3	n-1
1	6	7	4	2	5	3	n-2
1	2	7	4	6	5	3	n-3
1	2	3	4	6	5	7	..
1	2	3	4	6	5	7	2
1	2	3	4	5	6	7	1
1	2	3	4	5	6	7	0

Proposition

- The number of comparisons of selection sort is $\approx n^2/2$
- Justification: Number of comparisons:

$$1 + 2 + \cdots + (n-2) + (n-1) = n(n-1)/2 \tag{1}$$

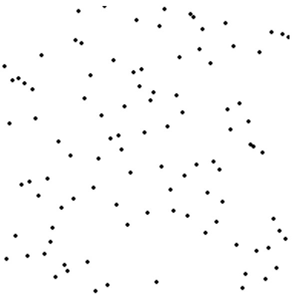
Selection Sort in Java

4	6	7	1	2	5	3	n-1
1	6	7	4	2	5	3	n-2
1	2	7	4	6	5	3	n-3
1	2	3	4	6	5	7	..
1	2	3	4	6	5	7	2
1	2	3	4	5	6	7	1
1	2	3	4	5	6	7	0

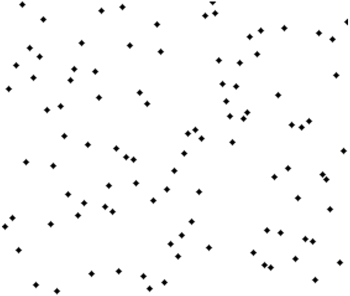
```
public static void selectionSort(String[] data) {
  int n = data.length;
  for (int i=0;i<n-1;i++)  {
    int minIndex=i;
    for (int j=i+1; j<n; j++){
      if (data[minIndex].compareTo(data[j])<0)
         minIndex=j;
     }
     if (i!=minIndex) swap(data, minIndex, i);
   }
}
```

How to improve it?

Illustration of Insertion Sort



Selection Sort



Insertion Sort

Insertion Sort Algorithm

Algorithm 8: Insertion Sort

Input: Array A of length n
Output: Sorted A

```
for i =1; i < n; i++ do
    j=i;
    while j>0 and A[j-1]>A[j] do
        swap(A[j], A[j-1]);
        j=j-1;
```

4	6	7	1	2	5	3
4	6	7	1	2	5	3

Insertion Sort Algorithm

Algorithm 9: Insertion Sort

Input: Array A of length n
Output: Sorted A

```
for i =1; i < n; i++ do
    j=i;
    while j>0 and A[j-1]>A[j] do
        swap(A[j], A[j-1]);
        j=j-1;
```

4	6	7	1	2	5	3
4	6	7	1	2	5	3
4	6	7	1	2	5	3

Insertion Sort Algorithm

Algorithm 10: Insertion Sort

Input: Array A of length n
Output: Sorted A

```
for i =1; i < n; i++ do
    j=i;
    while j>0 and A[j-1]>A[j] do
        swap(A[j], A[j-1]);
        j=j-1;
```

4	6	7	1	2	5	3
4	6	7	1	2	5	3
4	6	7	1	2	5	3
4	6	7	1	2	5	3

Insertion Sort Algorithm

Algorithm 11: Insertion Sort

Input: Array A of length n
Output: Sorted A

```
for i =1; i < n; i++ do
    j=i;
    while j>0 and A[j-1]>A[j] do
        swap(A[j], A[j-1]);
        j=j-1;
```

4	6	7	1	2	5	3
4	6	7	1	2	5	3
4	6	7	1	2	5	3
4	6	7	1	2	5	3
1	4	6	7	2	5	3

Insertion Sort Algorithm

Algorithm 12: Insertion Sort

Input: Array A of length n
Output: Sorted A

```
for i =1; i < n; i++ do
    j=i;
    while j>0 and A[j-1]>A[j] do
        swap(A[j], A[j-1]);
        j=j-1;
```

4	6	7	1	2	5	3
4	6	7	1	2	5	3
4	6	7	1	2	5	3
4	6	7	1	2	5	3
1	4	6	7	2	5	3
1	2	4	6	7	5	3

Insertion Sort Algorithm

Algorithm 13: Insertion Sort

Input: Array A of length n
Output: Sorted A

```
for i =1; i < n; i++ do
    j=i;
    while j>0 and A[j-1]>A[j] do
        swap(A[j], A[j-1]);
        j=j-1;
```

4	6	7	1	2	5	3
4	6	7	1	2	5	3
4	6	7	1	2	5	3
4	6	7	1	2	5	3
1	4	6	7	2	5	3
1	2	4	6	7	5	3
1	2	4	5	6	7	3

Insertion Sort Algorithm

Algorithm 14: Insertion Sort

Input: Array A of length n
Output: Sorted A

```
for i =1; i < n; i++ do
    j=i;
    while j>0 and A[j-1]>A[j] do
        swap(A[j], A[j-1]);
        j=j-1;
```

4	6	7	1	2	5	3
4	6	7	1	2	5	3
4	6	7	1	2	5	3
4	6	7	1	2	5	3
1	4	6	7	2	5	3
1	2	4	6	7	5	3
1	2	4	5	6	7	3
1	2	3	4	5	6	7

Insertion Sort Algorithm

Algorithm 15: Insertion Sort

Input: Array A of length n
Output: Sorted A

```
for i =1; i < n; i++ do
    j=i;
    while j>0 and A[j-1]>A[j] do
        swap(A[j], A[j-1]);
        j=j-1;
```

4	6	7	1	2	5	3
4	6	7	1	2	5	3
4	6	7	1	2	5	3
4	6	7	1	2	5	3
1	4	6	7	2	5	3
1	2	4	6	7	5	3
1	2	4	5	6	7	3
1	2	3	4	5	6	7

Insertion Sort Algorithm

Algorithm 16: Insertion Sort

Input: Array A of length n
Output: Sorted A

```
for i =1; i < n; i++ do
    j=i;
    while j>0 and A[j-1]>A[j] do
        swap(A[j], A[j-1]);
        j=j-1;
```

4	6	7	1	2	5	3
4	6	7	1	2	5	3
4	6	7	1	2	5	3
4	6	7	1	2	5	3
1	4	6	7	2	5	3
1	2	4	6	7	5	3
1	2	4	5	6	7	3
1	2	3	4	5	6	7

Why is it faster than selection sort

4	6	7	1	2	5	3	
4	6	7	1	2	5	3	1
4	6	7	1	2	5	3	$\sim 2/2$
4	6	7	1	2	5	3	$\sim 3/2$
1	4	6	7	2	5	3	$\sim 4/2$
1	2	4	6	7	5	3	..
1	2	4	5	6	7	3	$\sim$ (n-2)/2
1	2	3	4	5	6	7	$\sim$ (n-1)/2

The word-count problem

- Count the frequency of words in a text file, and return the most frequent word with its count.

```
there are two ways of constructing a software design one way
is to make it so simple that there are obviously no deficiencies
and the other way is to make it so complicated that there are no
obvious deficiencies
```

```
The most frequent word is "there" with 3 occurrences.
```

Implementation using arrays:

wordArray	there	are	two	ways	..
countArray	3	1	1	1	...

The algorithm

Input: Array of string tokens
Output: The most frequent word and its frequency

```
begin
    Initialize wordArray and countArray;
    for each token in the input text do
        if token in wordArray with index i then
            increment countArray[i];
        else
            find j the last position of wordArray;
            wordArray[j]=token;
            countArray[j]=1;
    Find the most frequent word;
```

wordArray	there	are	two	ways	..
countArray	3	1	1	1	...

The starter code

wordArray	there	are	two	ways	..
countArray	3	1	1	1	...

```
public Entry<String,Integer>count_ARRAY(String[] tokens){
    int CAPACITY = 1000000;
    String[] words = new String[CAPACITY];
    int[] counts = new int[CAPACITY];
    for (int j = 0; j < tokens.length; j++) {
        String token = tokens[j];
        for (int i = 0; i < CAPACITY; i++) {
            if (words[i] == null) {o
                words[i] = token;
                counts[i] = 1;
                break;
            } else if (words[i].equals(token))
                counts[i] = counts[i] + 1;
```

How slow is this algorithm? How can we improve it?

Smilar applications

- from all the web pages, find the URL(web page) that is mentioned(linked to) most
- from web log, find the most frequent visitor
- from all the queries received by Google, which query is most popular

```
123.123.123.123 - - [26/Apr/2000:00:23:48 -0400] "GET␣/pics/wpaper.gif␣HTTP/1.0" 200 6248 "http://www.jafs
123.123.123.123 - - [26/Apr/2000:00:23:47 -0400] "GET␣/asctortf/␣HTTP/1.0" 200 8130 "http://search.netscap
123.123.123.123 - - [26/Apr/2000:00:23:48 -0400] "GET␣/pics/5star2000.gif␣HTTP/1.0" 200 4005 "http://www.j
123.123.123.123 - - [26/Apr/2000:00:23:50 -0400] "GET␣/pics/5star.gif␣HTTP/1.0" 200 1031 "http://www.jafsc
```

Take aways

- Array is the basic data structure
- Demonstrated several algorithms and applications: sorting, word count
- Advantages: fast access by index, ...
- Drawbacks: size fixed, need to shift for insertion or remove operation,...
- Readings: GoodRich et al. p104-p111.

Linked list is an alternative to overcome the fixed size problem.