Efficiency does Matter

Nov 29 (book ch 4.2)

Penn Percentage Project Survey

https://qfreeaccountssjc1.az1.qualtrics.com/jfe/form/SV_eJrorLliFIM9eaW

Banks and Bank Accounts

- Bank
  - Data
    - accounts
    - ...
  - Activities
    - Create Account
    - ....
- Bank Account
  - Data
    - Account Number
    - ...
  - Activities
    - Deposit
    - ...

Bank Account

```java
public class BankAccount {
    private final String accountNumber;
    private String name;
    private double balance;

    public BankAccount(String actNo, String name, double startDep) {
        this.accountNumber = actNo;
        this.name = name;
        this.balance = startDep;
    }

    public void changeName(String newName) {
        this.name = newName;
    }


    public double getBalance() {
        return balance;
    }

    public String getName() {
        return name;
    }

    public String toString() {
        return accountNumber + " " + name + " balance:" + balance;
    }

}
```

```java
// alternately, it would make sense to return new balance
    public boolean deposit(double dep) {
        if (dep < 0) {
            System.out.println("Cannot make a negative deposit");
            return false;
        }
        if (dep > 10000) {
            System.out.println("No. Would have to report this to the Treasury
            return false;
        }
        balance += dep;
        return true;
    }

    public boolean withdrawal(double withdrawal) {
        if (withdrawal < 0) {
            System.out.println("Cannot make a negative withdrawal");
            return false;
        }
        if ((balance - withdrawal) < 0) {
            System.out.println("Not enough money");
            return false;
        }
        balance -= withdrawal;
        return true;
    }
```

Bank

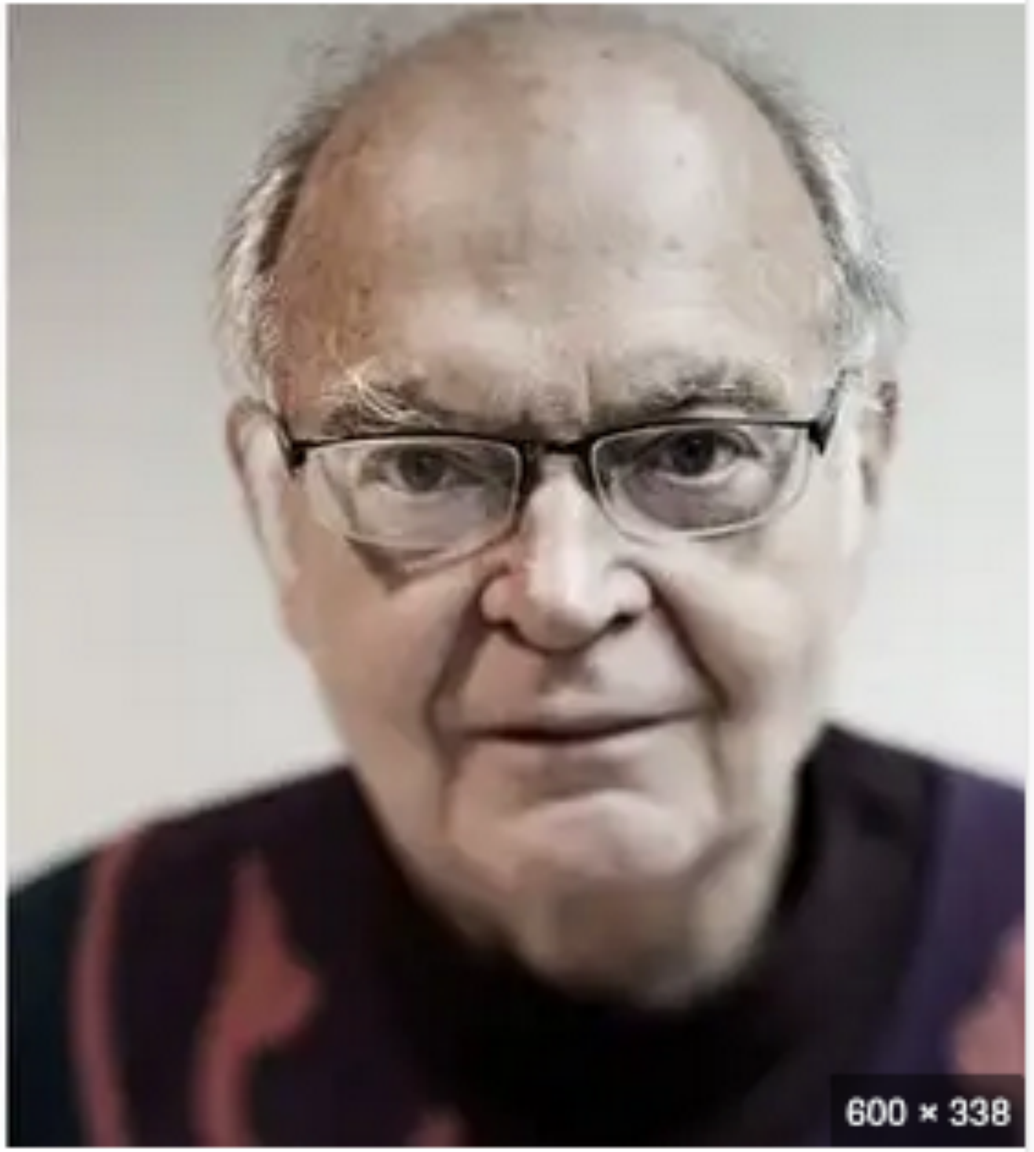
User

- make 3 accounts, each with a starting balance of $100
- 10 times
  - randomly pick an account
  - randomly pick an account in $10-20
  - randomly pick deposit/withdrawal

- Print the accounts

> We should forget about small efficiencies, say about 97% of the time: **premature optimization is the root of all evil**

— Donald Knuth



Listing Accounts

```
public void listAccounts() {
    for (int i = 0; i < activeAccounts; i++) {
        System.out.println(accounts[i]);
    }
}
```

- Order in which accounts were created is not very useful
- Better orders???

- How to get that order?

Find best, print, delete .... repeat

```
let arr = array of 10 integers greater than 0
for i in 0..10
   let bl = 0
   for j in 0..10
      if arr[bl] > arr[j]
         bl=j
   print arr[bl]
   arr[bl]=0
```

The "i" loop repeats 10 times, finding the next largest each time

The "j" loop finds the largest

delete?
Is this a general solution?

- How long does this take?
  - What do you mean by "long"?
- This algorithm is destructive!

Comparing Strings

- == vs equals
- > vs compareTo

```
"a".compareTo("b")  // -1
"a".compareTo("c") // -2
"e".compartTo("a") // 5
"aaa".compareTo("aab") // -1
"A".compareTo("a") // -32  (65-97)
```

- Note that compareTo is NOT boolean

```
"a".compareTo("b") < 0 // true if a < b
```

The `compareTo()` method compares two strings lexicographically.

The comparison is based on the **Unicode value** of each character in the strings.

The method returns 0 if the string is equal to the other string. A value less than 0 is returned if the string is "less" than the other string and a value greater than 0 if the string is "greater" than the other string.

Accounts ordered by name

using find best, print, delete

- make a copy of accounts!
- the things in temp are all aliases of things in accounts
- But temp is NOT an alias of accounts

```
public void listAccountsByName() {
    BankAccount[] temp = new BankAccount[activeAccounts];
    for (int i = 0; i < activeAccounts; i++) {
        temp[i] = accounts[i];
    }
    for (int i = 0; i < temp.length; i++) {
        int bestloc = -1;
        for (int j = 0; j < temp.length; j++) {
            if (temp[j] != null) {
                if (bestloc < 0) {
                    bestloc = j;
                } else {
                    if (temp[bestloc].getName().compareTo(temp[j].getName()) < 0) {
                        bestloc = j;
                    }
                }
            }
        }
        System.out.println(temp[bestloc]);
        temp[bestloc] = null;
    }
}
```

FindMove Sort

return the sorted result, destroy the original

```
public static int[] fbSort(int[] temp) {
    int[] result = new int[temp.length];
    for (int i = 0; i < temp.length; i++) {
        int bestloc = -1;
        for (int j = 0; j < temp.length; j++) {
            if (temp[j] >= 0) {
                if (bestloc < 0) {
                    bestloc = j;
                } else {
                    if (temp[bestloc] > temp[j]) {
                        bestloc = j;
                    }
                }
            }
        }
        result[i] = temp[bestloc];
        temp[bestloc] = -1;
    }
    return result;
}
```

This implementation assumes that numbers to be sorted are all non-negative integers. It also assumes sorting in descending order

Sorting

- You have to make a copy
- result does not last
- kind of slow

Do the sorting "in place"

Ideally do it faster

Bubble Sort

- idea, go across array
  - compare neighbors.
    - if neighbor is worse (better), SWAP

- After doing this once, what does the array look like

- So as with find, print, delete, need to repeat
  - faster than find, delete, print???

BubbleSort

- How many swaps?
  - Worst case?
- Can we improve??

```
public static void bubbleSort(int[] arr) {
    for (int i = 0; i < arr.length; i++) {
        for (int j = 1; j < (arr.length-i); j++) {
            if (arr[j-1] < arr[j]) {
                int temp = arr[j-1];
                arr[j-1] = arr[j];
                arr[j] = temp;
            }
        }
    }
}
```

Activity

Count operations for BubbleSort

1. Comparisons 2. Additions 3. Swaps

List 1		List 2		List 3	
0	10	0	90	0	10
1	22	1	82	1	12
2	54	2	74	2	14
3	86	3	66	3	16
4	56	4	56	4	26
5	15	5	55	5	35
6	55	6	45	6	45
7	7	7	37	7	47

```java
public static void bubbleSort(int[] arr) {
    for (int i = 0; i < arr.length; i++) {
        for (int j = 1; j < (arr.length-i); j++) {
            if (arr[j-1] < arr[j]) {
                int temp = arr[j-1];
                arr[j-1] = arr[j];
                arr[j] = temp;
            }
        }
    }
}
```

Improving Bubble

- Observation, at end of each inner loop, one additional item in place
- Moved a lot of stuff moves, but last item is done
- So, only move the last item?
  - Looks a lot like find, print, delete, repeat
    - Algorithm: find, swap, repeat

Selection Sort

find, swap, repeat

```
public static void selectionSort(int[] arr) {
    for (int i = 0; i < arr.length; i++) {
        int best = 0;
        for (int j = 1; j < (arr.length - i); j++) {
            if (arr[best] < arr[j]) {
                best = j;
            }
        }
        int temp = arr[best];
        arr[best] = arr[arr.length - i - 1];
        arr[arr.length - i - 1] = temp;
    }
}
```

Activity

Count operations for SelectionSort

1. Comparisons 2. Additions 3. Swaps

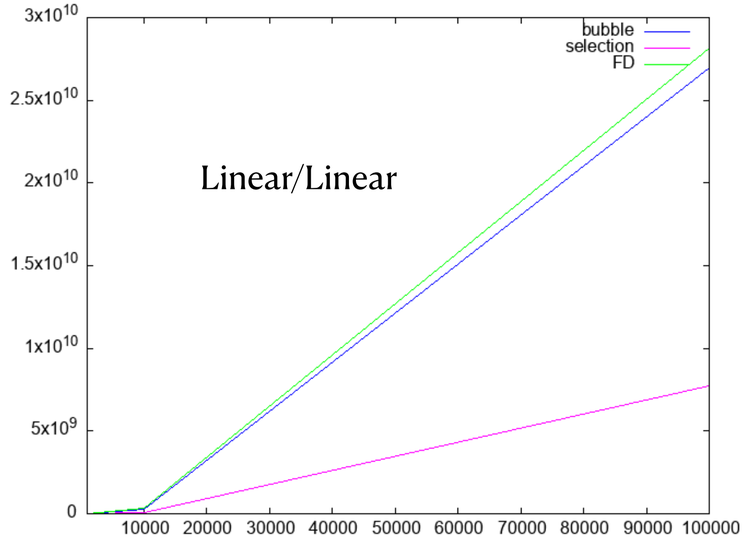
List 1	
0	10
1	22
2	54
3	86
4	56
5	15
6	55
7	7
8	84
9	67

List 2	
0	90
1	82
2	74
3	66
4	56
5	55
6	45
7	37
8	34
9	27

List 3	
0	10
1	12
2	14
3	16
4	26
5	35
6	45
7	47
8	54
9	67

Bubble vs Selection

selection is much faster



When one variable changes as a constant power of another, a log-log graph shows the relationship as a straight line.