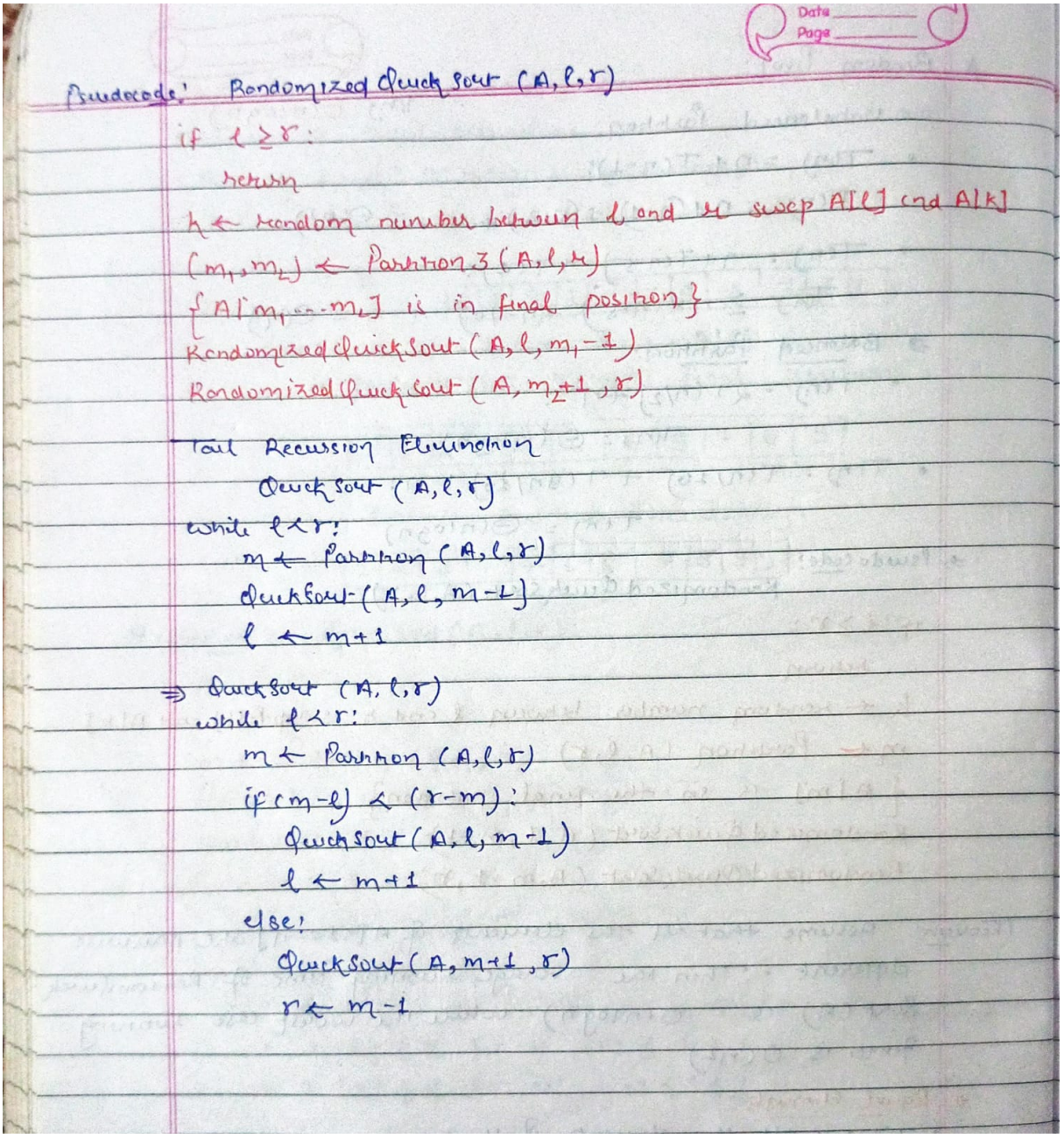
Sorting

Topics covered:

RandomizedQuick Sort

Change Problem

Dynamic Programming



Change Problem

Input: An integer money and positive integers $coin_1 \dots coin_d$

Output: The minimum number of coins with denominations $coin_1 \dots coin_d$ that changes money.

Greedy way:

```
GreedyChange(money)
change ← empty collection of coins
while money > 0:
    coin ← largest denomination
           that does not exceed money
    add coin to change
    money ← money - coin
return change
```

40 cents := 25 + 10 + 5 → 20 + 20

Greedy: is not optimal

in US in Tanzania

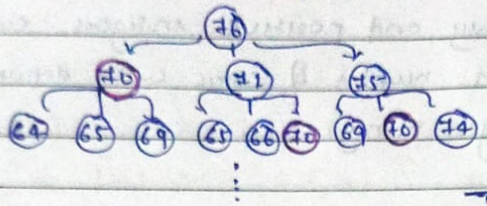
Recursive change

⇒ Give the denomination 6, 5 and 1 what is the minimum numbers of coins needed to change 9 cents ?

$$MinNumCoins(9) = \min \begin{cases} MinNumCoins(9-6)+1 \\ MinNumCoins(9-5)+1 \\ MinNumCoins(9-1)+1 \end{cases}$$

```
RecursiveChange(money, coins)
If money = 0:
    return 0
MinNumCoins ← ∞
for i from 1 to |coins|:
    if money ≥ coin_i:
        NumCoins ← RecursiveChange(money − coin_i, coins)
        if NumCoins + 1 < MinNumCoins:
            MinNumCoins ← MinCoins + 1
return MinNumCoins
```

How fast is RecursiveChange?



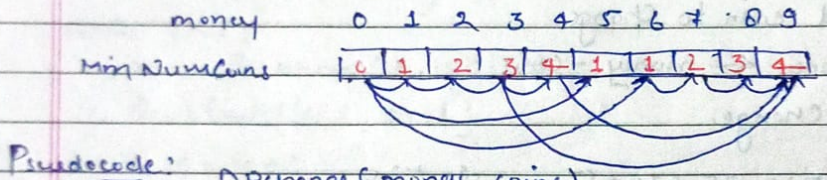
The optimal coin combination for 70 cents is computed at least three times!

The optimal coin combination for 30 cents is computed trillions of times.

It will take thousand years

Dynamic Programming

What is the minimum number of coins needed to change 0 cents for denominations 6, 5, and 1 ?



Pseudocode: DPchange(money, coins)

```
MinNumCoins(0) ← 0
for m from 1 to money:
    MinNumCoins(m) ← ∞
    for i from 1 to |coins|:
        if m ≥ coin_i:
            NumCoins ← MinNumCoins(m - coin_i) + 1
            if NumCoins < MinNumCoins(m):
                MinNumCoins(m) ← NumCoins
return MinNumCoins(money)
```

Date
Page

Cystic Fibrosis (disease)

→ Matching the gene which caused with their property

Computing Edit Distance

given: string A[1...n] and B[1...n], what is an optimal alignment (that results in minimum edit distance) of an i-prefix A[1...i] of the first string and a j-prefix B[1...j] of the second string.

* The last column of an optimal alignment is either
  an insertion,
  a deletion,
  a mismatch
  or a match.

what is left (after the removal of the last column) is an optimal alignment of the corresponding prefixes.

```
EditDistance(A[1...n], B[1...m])
D(i,0) ← and D(0,j) ← j for all i,j
for j from 1 to m:
    for i from 1 to n:
        insertion ← D(i,j-1) + 1
        deletion ← D(i-1,j) + 1
        match ← D(i-1,j-1)
        mismatch ← D(i-1,j-1) + 1
        if A[i] = B[j]
            D(i,j) ← min(insertion, deletion, match)
        else:
            D(i,j) ← min(insertion, deletion, mismatch)
return D(n,m)
```