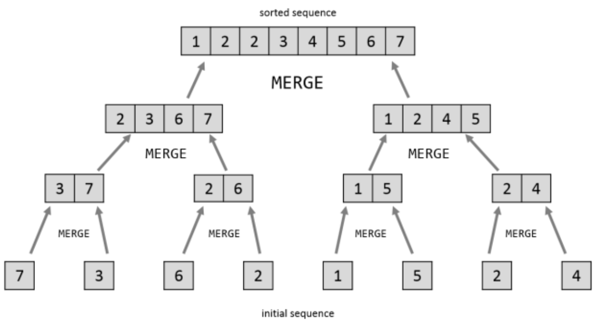
CO202 – Software Engineering – Algorithms

Divide and Conquer - Exercises

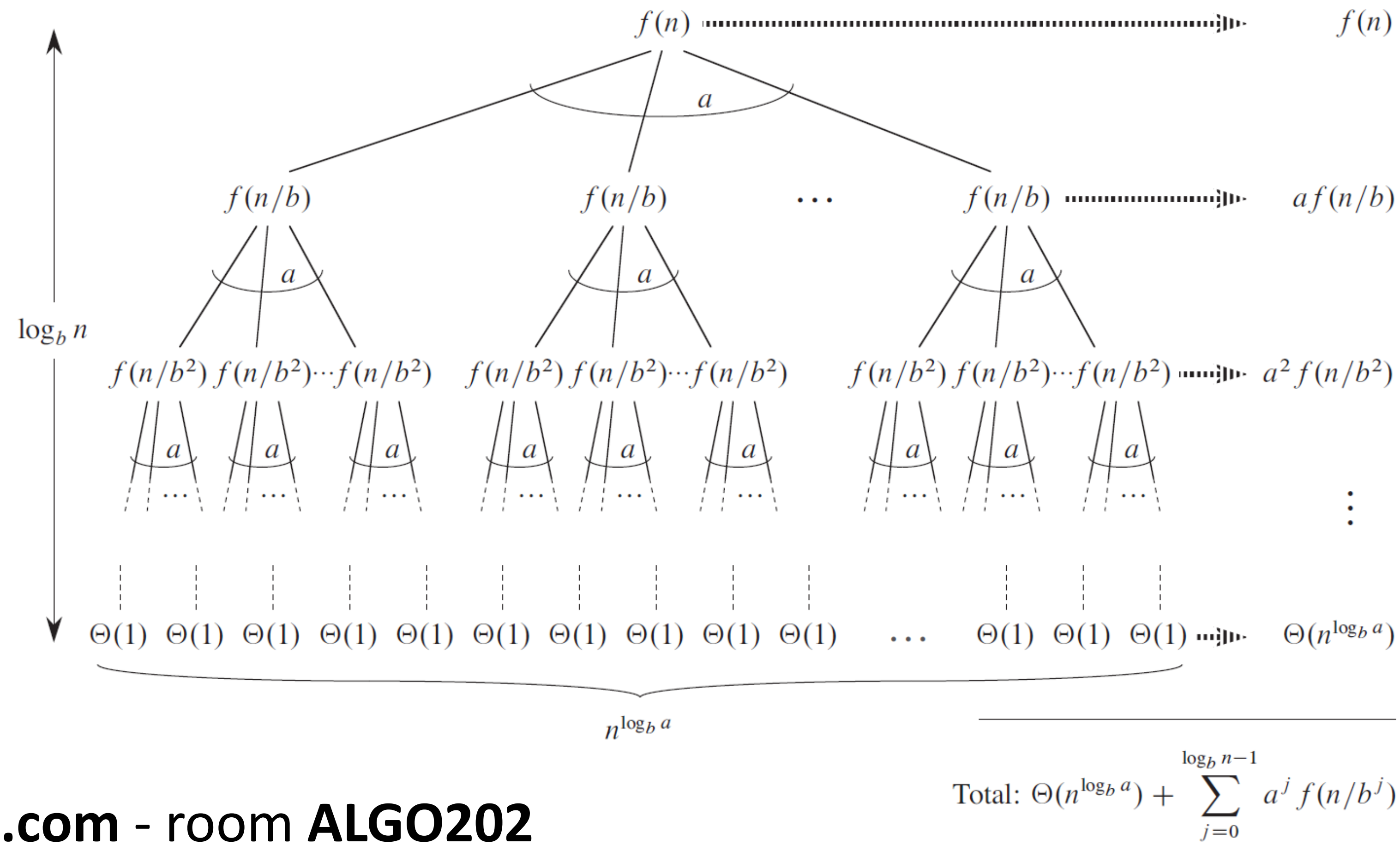
Exercise 1: Illustrate the Operations of Merge Sort

$$A = \langle 3, 41, 52, 26, 38, 57, 9, 49 \rangle$$



Exercise 2: Draw a Recursion Tree, Guess, and Verify

$$T(n) = 3T(n/4) + cn^2$$



b.socrative.com - room **ALGO202**

A: $O(\lg n)$ **B**: $O(n)$ **C**: $O(n \lg n)$ **D**: $O(n^2)$ **E**: $O(2^n)$

Exercise 3: Master Method

1. If $d < \log_b a$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $d = \log_b a$, then $T(n) = \Theta(n^d \lg n)$.
3. If $d > \log_b a$, then $T(n) = \Theta(n^d)$.

- $T(n) = 3T(n/4) + cn^2$
- $T(n) = 2T(^n/_4) + \sqrt{n}$
- $T(n) = 8T(n/2) + n^2$
- $T(n) = T(n/2) + 1$

Exercise 4: Substitution Method

$$T(n) = 2T(n/2) + 1$$

1) Obtain the running time using the Master Method
2) Confirm with the Substitution Method

Exercise 5: Divide and Conquer

1) Write in pseudo-code a recursive function $f(x, n) = x^n$ for powering a number using divide-and-conquer.

Hint:

- $x^n = x^{n/2} \cdot x^{n/2}$ for even n
- $x^n = x^{(n-1)/2} \cdot x^{(n-1)/2} \cdot x$ for odd n

2) Show that the running time complexity is $O(\lg n)$.

CO202 – Software Engineering – Algorithms

Dynamic Programming - Exercises

Exercise 1: Compute LCS

	j	0	1	2	3	4	5	6
i		y_j	**B**	**D**	**C**	**A**	**B**	**A**
0	x_i							
1	**A**							
2	**B**							
3	**C**							
4	**B**							
5	**D**							
6	**A**							
7	**B**							

Exercise 1: Compute LCS

	j	0	1	2	3	4	5	6
i		y_j	**B**	**D**	**C**	**A**	**B**	**A**
0	x_i	0	0	0	0	0	0	0
1	**A**	0						
2	**B**	0						
3	**C**	0						
4	**B**	0						
5	**D**	0						
6	**A**	0						
7	**B**	0						

```
for j = 1 to n
    for i = 1 to m
        if x_i == y_j
            c[i,j] = c[i-1,j-1] + 1
            b[i,j] = ↖
        elseif c[i-1,j] ≥ c[i,j-1]
            c[i,j] = c[i-1,j]
            b[i,j] = ↑
        else
            c[i,j] = c[i,j-1]
            b[i,j] = ←
```

Exercise 1: Compute LCS

	j	0	1	2	3	4	5	6
i		y_j	**B**	**D**	**C**	**A**	**B**	**A**
0	x_i	0	0	0	0	0	0	0
1	**A**	0	0					
2	**B**	0						
3	**C**	0						
4	**B**	0						
5	**D**	0						
6	**A**	0						
7	**B**	0						

```
for j = 1 to n
    for i = 1 to m
        if x_i == y_j
            c[i,j] = c[i-1,j-1] + 1
            b[i,j] = ↖
        elseif c[i-1,j] ≥ c[i,j-1]
            c[i,j] = c[i-1,j]
            b[i,j] = ↑
        else
            c[i,j] = c[i,j-1]
            b[i,j] = ←
```

Exercise 1: Compute LCS

	j	0	1	2	3	4	5	6
i		y_j	**B**	**D**	**C**	**A**	**B**	**A**
0	x_i	0	0	0	0	0	0	0
1	**A**	0	↑ 0					
2	**B**	0						
3	**C**	0						
4	**B**	0						
5	**D**	0						
6	**A**	0						
7	**B**	0						

```
for j = 1 to n
    for i = 1 to m
        if xi == yj
            c[i,j] = c[i-1,j-1] + 1
            b[i,j] = ↖
        elseif c[i-1,j] ≥ c[i,j-1]
            c[i,j] = c[i-1,j]
            b[i,j] = ↑
        else
            c[i,j] = c[i,j-1]
            b[i,j] = ←
```

Exercise 1: Compute LCS

	j	0	1	2	3	4	5	6
i		y_j	**B**	**D**	**C**	**A**	**B**	**A**
0	x_i	0	0	0	0	0	0	0
1	**A**	0	↑ 0					
2	**B**	0	1					
3	**C**	0						
4	**B**	0						
5	**D**	0						
6	**A**	0						
7	**B**	0						

```
for j = 1 to n
    for i = 1 to m
        if x_i == y_j
            c[i,j] = c[i-1,j-1] + 1
            b[i,j] = ↖
        elseif c[i-1,j] ≥ c[i,j-1]
            c[i,j] = c[i-1,j]
            b[i,j] = ↑
        else
            c[i,j] = c[i,j-1]
            b[i,j] = ←
```

Exercise 1: Compute LCS

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0					
2	B	0	↖ 1					
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

```
for j = 1 to n
    for i = 1 to m
        if x_i == y_j
            c[i,j] = c[i-1,j-1] + 1
            b[i,j] = ↖
        elseif c[i-1,j] ≥ c[i,j-1]
            c[i,j] = c[i-1,j]
            b[i,j] = ↑
        else
            c[i,j] = c[i,j-1]
            b[i,j] = ←
```

Exercise 1: Compute LCS

	j	0	1	2	3	4	5	6
i		y_j	**B**	**D**	**C**	**A**	**B**	**A**
0	x_i	0	0	0	0	0	0	0
1	**A**	0	↑ 0					
2	**B**	0	↖ 1					
3	**C**	0	1					
4	**B**	0						
5	**D**	0						
6	**A**	0						
7	**B**	0						

```
for j = 1 to n
    for i = 1 to m
        if x_i == y_j
            c[i,j] = c[i-1,j-1] + 1
            b[i,j] = ↖
        elseif c[i-1,j] ≥ c[i,j-1]
            c[i,j] = c[i-1,j]
            b[i,j] = ↑
        else
            c[i,j] = c[i,j-1]
            b[i,j] = ←
```

Exercise 1: Compute LCS

	j	0	1	2	3	4	5	6
i		y_j	**B**	**D**	**C**	**A**	**B**	**A**
0	x_i	0	0	0	0	0	0	0
1	**A**	0	↑ 0					
2	**B**	0	↖ 1					
3	**C**	0	↑ 1					
4	**B**	0						
5	**D**	0						
6	**A**	0						
7	**B**	0						

```
for j = 1 to n
    for i = 1 to m
        if x_i == y_j
            c[i,j] = c[i-1,j-1] + 1
            b[i,j] = ↖
        elseif c[i-1,j] ≥ c[i,j-1]
            c[i,j] = c[i-1,j]
            b[i,j] = ↑
        else
            c[i,j] = c[i,j-1]
            b[i,j] = ←
```

Exercise 2: Compute Levenshtein Distance

	j	0	1	2	3	4	5	6	7	8	9
i		y_j	**E**	**M**	**P**	**I**	**R**	**I**	**C**	**A**	**L**
0	x_i										
1	**I**										
2	**M**										
3	**P**										
4	**E**										
5	**R**										
6	**I**										
7	**A**										
8	**L**										

Exercise 2: Compute Levenshtein Distance

	j	0	1	2	3	4	5	6	7	8	9
i		y_j	E	M	P	I	R	I	C	A	L
0	x_i	0	1	2	3	4	5	6	7	8	9
1	I	1									
2	M	2									
3	P	3									
4	E	4									
5	R	5									
6	I	6									

```
for j = 1 to n
    for i = 1 to m
        c = x_i == y_j ? 0 : 1
        d[i,j] = min(d[i-1,j] + 1, d[i,j-1] + 1, d[i-1,j-1] + c)
```

Exercise 2: Compute Levenshtein Distance

	j	0	1	2	3	4	5	6	7	8	9
i		y_j	**E**	**M**	**P**	**I**	**R**	**I**	**C**	**A**	**L**
0	x_i	0	1	2	3	4	5	6	7	8	9
1	**I**	1	R 1								
2	**M**	2									
3	**P**	3									
4	**E**	4									
5	**R**	5									
6	**I**	6									

```
for j = 1 to n
    for i = 1 to m
        c = x_i == y_j ? 0 : 1
        d[i,j] = min(d[i-1,j] + 1, d[i,j-1] + 1, d[i-1,j-1] + c)
```

Exercise 2: Compute Levenshtein Distance

	j	0	1	2	3	4	5	6	7	8	9
i		y_j	E	M	P	I	R	I	C	A	L
0	x_i	0	1	2	3	4	5	6	7	8	9
1	I	1	R 1								
2	M	2	R D 2								
3	P	3									
4	E	4									
5	R	5									
6	I	6									

```
for j = 1 to n
    for i = 1 to m
        c = x_i == y_j ? 0 : 1
        d[i,j] = min(d[i-1,j] + 1, d[i,j-1] + 1, d[i-1,j-1] + c)
```

Exercise 2: Compute Levenshtein Distance

	j	0	1	2	3	4	5	6	7	8	9
i		y_j	**E**	**M**	**P**	**I**	**R**	**I**	**C**	**A**	**L**
0	x_i	0	1	2	3	4	5	6	7	8	9
1	**I**	1	R 1								
2	**M**	2	R D 2								
3	**P**	3	R D 3								
4	**E**	4									
5	**R**	5									
6	**I**	6									

```
for j = 1 to n
    for i = 1 to m
        c = xi == yj ? 0 : 1
        d[i,j] = min(d[i-1,j] + 1, d[i,j-1] + 1, d[i-1,j-1] + c)
```

Exercise 2: Compute Levenshtein Distance

	j	0	1	2	3	4	5	6	7	8	9
i		y_j	**E**	**M**	**P**	**I**	**R**	**I**	**C**	**A**	**L**
0	x_i	0	1	2	3	4	5	6	7	8	9
1	**I**	1	R 1								
2	**M**	2	R D 2								
3	**P**	3	R D 3								
4	**E**	4	K 3								
5	**R**	5									
6	**I**	6									

```
for j = 1 to n
    for i = 1 to m
        c = x_i == y_j ? 0 : 1
        d[i,j] = min(d[i-1,j] + 1, d[i,j-1] + 1, d[i-1,j-1] + c)
```

Exercise 2: Compute Levenshtein Distance

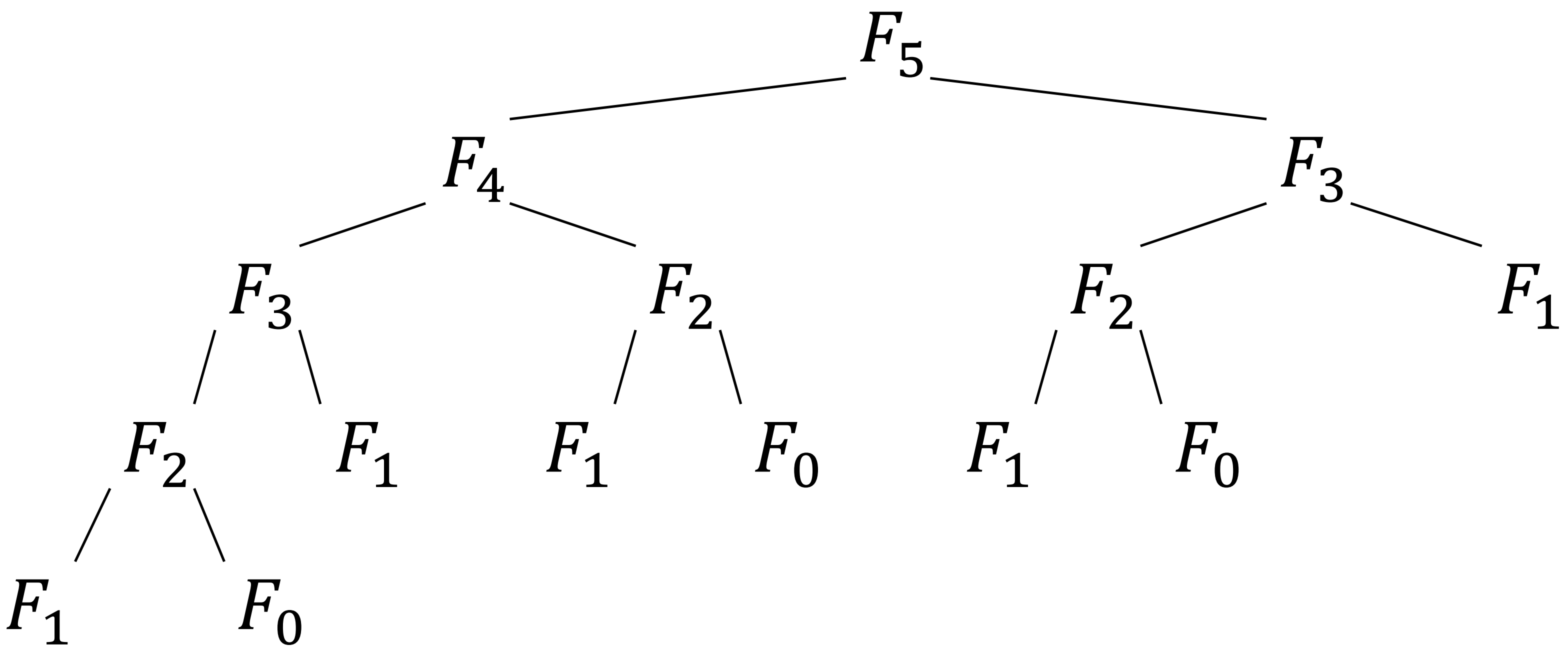
	j	0	1	2	3	4	5	6	7	8	9
i		y_j	E	M	P	I	R	I	C	A	L
0	x_i	0	1	2	3	4	5	6	7	8	9
1	I	1	R 1								
2	M	2	R D 2								
3	P	3	R D 3								
4	E	4	K 3								
5	R	5	D 4								
6	I	6									

```
for j = 1 to n
    for i = 1 to m
        c = x_i == y_j ? 0 : 1
        d[i,j] = min(d[i-1,j] + 1, d[i,j-1] + 1, d[i-1,j-1] + c)
```

Exercise 3: Fibonacci Sequence

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, …

$$F(n) = \begin{cases} 0, & \text{if } n = 0 \\ 1, & \text{if } n = 1 \\ F(n-1) + F(n-2), & \text{otherwise} \end{cases}$$



Exercise 3: Fibonacci Sequence

```
NAÏVE-FIBONACCI(n)
if n == 0
    return 0
if n == 1
    return 1
return NAÏVE-FIBONACCI(n-1) + NAÏVE-FIBONACCI(n-2)
```

Running time of NAÏVE-FIBONACCI:

$$T(n) = O(2^{0.694n})$$

Exercise 3: Fibonacci Sequence

```
BOTTOM-UP-FIBONACCI(n)
  if n == 0
      return 0
  let f[0..n] be a new array
  f[0] = 0
  f[1] = 1
  ?
  ?
  ?
```

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, …

$$F(n) = \begin{cases} 0, & \text{if } n = 0 \\ 1, & \text{if } n = 1 \\ F(n-1) + F(n-2), & \text{otherwise} \end{cases}$$

What is the running time of BOTTOM-UP-FIBONACCI?

Exercise 4: Fibonacci Challenge

D&C Fibonacci Revisited

Naïve:

$$F(n) = \begin{cases} 0, & \text{if } n = 0 \\ 1, & \text{if } n = 1 \\ F(n-1) + F(n-2), & \text{otherwise} \end{cases}$$

Let's rewrite Fibonacci

$F(n) = F(2k) = F(k)^2 + 2F(k)F(k-1)$ for even n

$F(n) = F(2k-1) = F(k)^2 + F(k-1)^2$ for odd n

$$F(n) = \begin{cases} 0, & \text{if } n = 0 \\ 1, & \text{if } n = 1 \\ F(\lceil n/2 \rceil)^2 + 2F(\lceil n/2 \rceil)F(\lceil n/2 \rceil - 1), & \text{if } n \geq 2 \text{ and } n \text{ is even} \\ F(\lceil n/2 \rceil)^2 + F(\lceil n/2 \rceil - 1)^2, & \text{if } n \geq 2 \text{ and } n \text{ is odd} \end{cases}$$

Exercise 4: Fibonacci Challenge

D&C Fibonacci Revisited

$$F(n) = \begin{cases} 0, & \text{if } n = 0 \\ 1, & \text{if } n = 1 \\ \boldsymbol{F(\lceil n/2 \rceil)}^2 + 2\boldsymbol{F(\lceil n/2 \rceil)}\boldsymbol{F(\lceil n/2 \rceil - 1)}, & \text{if } n \geq 2 \text{ and } n \text{ is even} \\ \boldsymbol{F(\lceil n/2 \rceil)}^2 + \boldsymbol{F(\lceil n/2 \rceil - 1)}^2, & \text{if } n \geq 2 \text{ and } n \text{ is odd} \end{cases}$$

```
DC-FIBONACCI(n)
 if n == 0 || n == 1
     return n
 else
 ?
```

What is the running time?

Exercise 5: Coin Change Problem

Coin change is the problem of finding the least number of coins for a given amount of money.

For example, the UK coin set contains the following coins:

- 1p, 2p, 5p, 10p, 20p, 50p, £1, £2, and £5 (very uncommon).
- For £2.82, the optimal change is £2, 50p, 20p, 10p, 2p.

1. Write a mathematical recurrence equation that determines the least number of coins.

2. Devise a pseudo-code, bottom-up dynamic programming algorithm `coin_change(n,coins)`.

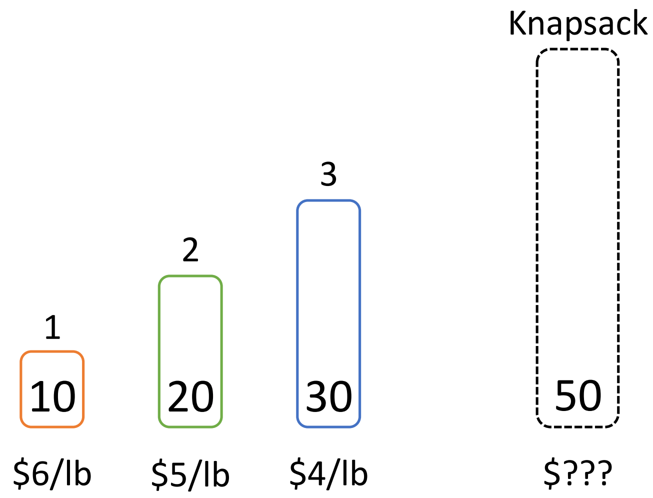
CO202 – Software Engineering – Algorithms

Greedy Algorithms - Exercises

Exercise 1: Implement Fractional Knapsack

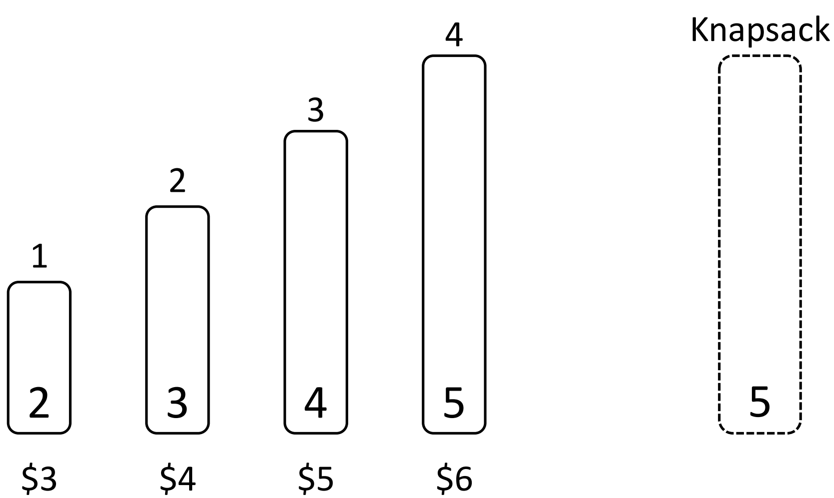
```
FRACTIONAL-KNAPSACK(v,w,K)
```

i	1	2	3
v_i	60	100	120
w_i	10	20	30
v_i/w_i	6	5	4



Exercise 2: Solving 0-1 Knapsack

```
for i = 1 to n
    c[i,0] = 0
    for j = 1 to K
        if w[i] ≤ j
            c[i,j] = max(v[i] + c[i-1,j-w[i]], c[i-1,j])
        else
            c[i,j] = c[i-1,j]
```



i \ j	0	1	2	3	4	5
0	0	0	0	0	0	0
1						
2						
3						
4						

Exercise 3: Coin Change Problem

Prove that a greedy strategy of picking the highest valued coin which is less or equal than the remaining amount is not guaranteed to produce optimal results.

Exercise 4: Planning a Party

Invite as many people as possible from a set of n people, such that

1. Every person invited should know at least five other people that are invited
2. Every person invited should not know at least five other people that are invited

Hint: Maximizing the number of invitees is the same as minimizing the number of people that are not invited.

CO202 – Software Engineering – Algorithms

Randomised Algorithms - Exercises

Exercise 1: Illustrate the Operations of Partition

$$A = \langle 3, 5, 2, 1, 8, 9 \rangle$$

```
PARTITION(A,p,r)
 x = A[r]
 i = p-1
 for j = p to r-1
     if A[j] ≤ x
         i = i+1
         SWAP(A[i],A[j])
 SWAP(A[i+1],A[r])
 return i+1
```

Exercise 2: The Original Partition Algorithm

$$A = \langle 3, 5, 2, 1, 8, 9 \rangle$$

```
HOARE-PARTITION(A,p,r)
x = A[p]
i = p
j = r+1
while TRUE
    repeat
        j = j-1
    until A[j] ≤ x or j == p
    repeat
        i = i+1
    until A[i] ≥ x or i == r
    if i < j
        SWAP(A[i],A[j])
    else
        SWAP(A[p],A[j])
        return j
```

Exercise 3: Randomised BST Insert

$$\langle 2,3,1,5,7,8,9\rangle \quad \langle 0,1,0,1,1,0,0\rangle$$

```
INSERT-RAND(t,z)
if t == NIL
    return z
r = RANDOM(1,t.size+1)
if r == 1
    return ROOT-INSERT(t,z)
if z.key < t.key
    t.left = INSERT-RAND(t.left,z)
else
    t.right = INSERT-RAND(t.right,z)
t.size = t.size + 1
return t
```

```
ROOT-INSERT(t,z)
if t == NIL
    return z
if z.key < t.key
    t.left = ROOT-INSERT(t.left,z)
    t.size = t.size + 1
    return RIGHT-ROTATE(t)
else
    t.right = ROOT-INSERT(t.right,z)
    t.size = t.size + 1
    return LEFT-ROTATE(t)
```

```
LEFT-ROTATE(t)
r = t.right
t.right = r.left
r.left = t
r.size = t.size
t.size -= r.right.size + 1
return r
```

```
RIGHT-ROTATE(t)
l = t.left
t.left = l.right
l.right = t
l.size = t.size
t.size -= l.left.size + 1
return l
```

Exercise 4: Approximating Pi

How can we approximate pi using random numbers?

Exercise 5: Finding the k-th Smallest Element

Given set $A = \{a_1, \dots, a_n\}$, find the k-th smallest Element

CO202 – Software Engineering – Algorithms

String Matching - Exercises

Exercise 1: Prefix Function - Example 3

Complete the prefix function table

$$\pi[q] = \max\{k : k < q \text{ and } P_k \sqsupset P_q\}$$

i	1	2	3	4	5	6	7	8	9	10	11
$P[i]$	a	b	r	a	c	a	d	a	b	r	a
$\pi[i]$											

Exercise 2: Boyer-Moore Preprocessing

BCR Table

c	a	b	c	d	r
$bcr[c]$					

GSR Table

i	1	2	3	4	5	6	7	8	9	10	11
$P[i]$	a	b	r	a	c	a	d	a	b	r	a
$gsr[i]$											

Exercise 3: Worst-Case of Boyer-Moore

Show that the worst-case running time of the `BM-MATCHER` algorithm is $O(nm)$.

Exercise 4: Prefix Function as String Matcher

1) Devise an algorithm `PF-MATCHER` that uses the prefix function directly to find all occurrences of P in T?

2) What is the running time complexity of this method?

CO202 – Software Engineering – Algorithms

Radix Search - Exercises

Exercise 1: Digital Search Trees

Draw the DST that results when you insert the following keys in that order into an initially empty tree

E A S Y Q U T I O N

E	00101
A	00001
S	10011
Y	11001
Q	10001
U	10101
T	10100
I	01001
O	01111
N	01110

Exercise 2: Binary Search Tries

Draw the binary trie that results when you insert the following keys in that order into an initially empty trie

E A S Y Q U T I O N

E	00101
A	00001
S	10011
Y	11001
Q	10001
U	10101
T	10100
I	01001
O	01111
N	01110

Exercise 3: Patricia Tries

Draw the Patricia trie that results when you insert the following keys in that order into an initially empty trie

E A S Y Q U T I O N

E	00101
A	00001
S	10011
Y	11001
Q	10001
U	10101
T	10100
I	01001
O	01111
N	01110

Exercise 3: Patricia Tries

Draw the Patricia trie that results when you insert the following keys in that order into an initially empty trie

E A S Y Q U ~~T I O N~~

E	00101
A	00001
S	10011
Y	11001
Q	10001
U	10101
T	10100
I	01001
O	01111
N	01110

CO202 – Software Engineering – Algorithms

Graph Algorithms - Exercises

Exercise 1: Optimal Substructure of Shortest Path

Lemma: Given a weighted, directed graph $G = (V, E)$ with weight function $w: E \to \mathbb{R}$, let $p = \langle v_0, v_1, \ldots, v_k \rangle$ be a shortest path from vertex v_0 to v_k and, for any i and j such that $0 \le i \le j \le k$, let $p_{ij} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$ be the subpath of p from vertex v_i to v_j.

Then, p_{ij} is a shortest path from v_i to v_j.

Exercise 2: Good guys, bad guys

There are two types of professional wrestlers: “babyfaces” (good guys) and “heels” (bad guys). Between any pair of professional wrestlers, there may or may not be a rivalry.

Suppose we have n professional wrestlers and we have a list of r pairs of wrestlers for which there are rivalries.

Task: Devise a strategy using graph algorithms that assigns each wrestler to one of the two types such that no rivalry exists between wrestlers of the same type.

Hint: This might not always be possible.

Exercise 3: Max-Flow/Min-Cut

c	s	1	2	3	4	t
s	0	9	3	0	0	0
1	0	0	2	0	7	0
2	0	0	0	6	0	0
3	0	0	0	0	3	6
4	0	0	0	0	0	5
t	0	0	0	0	0	0

f	s	1	2	3	4	t
s	0	5	3	0	0	0
1	-5	0	2	0	3	0
2	-3	-2	0	5	0	0
3	0	0	-5	0	2	3
4	0	-3	0	-2	0	5
t	0	0	0	-3	-5	0