Module Title: Informatics 2 – Introduction to Algorithms and Data Structures
Exam Diet: April 2024
Brief notes on answers:

PART A:

1. (a) Bookwork.

$$\begin{aligned} f = O(g) &\Leftrightarrow \exists C > 0. \exists N. \forall n \geq N.\, f(n) \leq Cg(n) \\ f = \Omega(g) &\Leftrightarrow \exists c > 0. \exists N. \forall n \geq N.\, cg(n) \leq f(n) \\ f = \Theta(g) &\Leftrightarrow f = O(g)\ \&\ f = \Omega(g) \end{aligned}$$

   Equivalent variations accepted. 1 mark each, being generous with half-marks for partially correct definitions.

   (b) Yes. We may take e.g. $C = 1$, $N = 25$. Then for any $n \geq N$, we have

$$5\sqrt{n} = \sqrt{N}\sqrt{n} \leq \sqrt{n}\sqrt{n} = Cn$$

   [1 mark for 'yes', 1 mark for choice of C, N, 1 mark for verifying the inequality.]

   (c) No. It's not true that $5\sqrt{n} = \Omega(n)$. Given any proposed c and N, we may take $n \geq N$ so that $n > 25/c^2$. Then

$$cn = c\sqrt{n}\sqrt{n} > c\sqrt{n}(5/c) = 5\sqrt{n}$$

   [This will be a little harder. 1 mark for 'no', 1 mark for an attempt to show $\forall c \forall N. \cdots$, 1 mark for the details.]

2. (a) The nested call structure is

```
Expmod(2,22,23)
    Expmod(2,11,23)
        Expmod(2,5,23)
            Expmod(2,2,23)
                Expmod(2,1,23)
                    Expmod(2,0,23)
                    1
                2
            4
        32 mod 23 = 9
    162 mod 23 = 1
1
```

   [3 marks for the right idea/structure, 3 marks for the details.]

   (b) $T(n) =$ if $n = 0$ then $\Theta(1)$ else $T(\lfloor n/2 \rfloor) + \Theta(1)$.
   [1 mark for $\lfloor n/2 \rfloor$, 1 mark for other details.]

   (c) $\Theta(\lg n)$, $\Theta(d)$. [1 mark each.]

3. (a) The array representation of the heap is $[24, 17, 13, 9, 11, 10, 12, 3, 7]$

(b) After the first Max-Heap-Extract-Max operation, the array representation of the heap is $[17, 11, 13, 9, 7, 10, 12, 3]$.

After the Max-Heap-Insert(6) operation, the array representation of the heap is $[17, 11, 13, 9, 7, 10, 12, 3, 6]$.

After the second Max-Heap-Extract-Max operation, the array representation of the heap is $[13, 11, 12, 9, 7, 10, 6, 3]$.

After the Max-Heap-Insert(16) operation, the array representation of the heap is $[16, 13, 12, 11, 7, 10, 6, 3, 9]$.

(c) The Max-Heap-Insert operation can either be implemented recursively (via Max-Heapify-Up), or without recursion. In either case, the running time is $O(\lg n)$ where n is the size of the array representing the heap. We will argue for the recursive implementation, the other one is similar. The running time of Max-Heap-Insert consists of a few lines of pseudocode that each take $O(1)$ time, for increasing the heap size, adding the new element at the end, and setting the counter to the heap size to call Max-Heapify-Up. Max-Heapify-Up compares the value of the current element with that of its parent in $O(1)$ time and makes the swap if it is larger. If it makes the swap, then it calls itself recursively on its parent. The number of times that the procedure might be called is at most the height of the tree representing the heap. The height of this tree is $O(\lg n)$, so the running time is $O(\lg n)$.

Marking: 1 mark for the correct array representation in the first question. For the second question, 1.5 marks will be given for each correct step, with the total marks of the subquestion rounded up to the nearest integer (e.g., 3/4 correct answers award 5 marks). For the last question, 3 marks will be awarded for a complete answer; answers do not need to argue that the height of the tree is $O(\lg n)$. Partial answers in the right direction will be awarded partial marks.

4. (a) It is not possible for the value of $M[i, v]$ to be anything other than 0. If that was the case, that would mean that there is a path from v to v (i.e., a cycle) that has total negative cost. The Bellman-Ford algorithm does not run on graphs that have cycles of negative cost.

Marking: 1 mark for a correct example, 1 mark for the justification. Any example that works is acceptable.

(b) The $2D$-array M is given in the following table

	z	s	t	y	x
0	0	∞	∞	∞	∞
1	0	∞	-4	9	∞
2	0	2	-4	9	-6
3	0	2	-4	-9	-6
4	0	-2	-4	-9	-6

Marking: 5 marks for the correct table. 1 mark will be deducted for each incorrect row of the table.

(c) The modification to the recurrence relation is the following:

$$M[i, v] = \min\{M[i-1, v], \min_{u \in N^-(u)} c_{uv} + M[i-1, u]\},$$

where $N^-(u)$ is the set of nodes for which there is an edge (u, v) in the graph.

Marking: 3 marks for a correct recurrence relation, explaining what N^- is (or however they choose to name it). If this set is written without explanation, 1 mark will be deducted.

5. (a) The outcome of the EFT algorithm on this input is $[2, 3]$, $[4, 6]$, $[7, 9]$.

Marking: 2 marks for a correct answer. 1 mark will be deducted for each missing interval, or for any interval that is not part of the solution of the algorithm.

(b) These examples were presented in the lecture slides. For EST, consider an instance in which there are $m - 1$ non-overlapping intervals $1, 2, \ldots, m - 1$, with $s_{i+1} > f_i$ for all $i = 1, \ldots, m - 1$ and another interval m with $s_m = s_1 - \varepsilon$ and $f_m = f_{m-1}$. The EST algorithm will schedule only interval m whereas the optimal schedule will schedule the other $m - 1$ intervals instead. Hence the approximation ratio is at least $m - 1$.

For SI, consider an instance in which there are 3 intervals $1, 2, 3$, such that 1 and 2 overlap, 2 and 3 overlap, but 1 and 3 don't overlap and interval 2 is smaller than the other two. The SI algorithm will select only interval 2, whereas the optimal schedule will select the other two intervals, resulting in an approximation ratio of at least 2.

Marking: 2 marks for EST, 1 mark for SI. Any correct example awards full marks.

(c) The only schedule that is optimal is the one that includes the jobs (9pm, 4 am) and (1pm, 7pm).

Marking: 2 marks for the correct answer. This basically test the understanding of the student to comprehend a problem described to them. If the correct schedule is given, together with one more incorrect schedule, 1 mark will be awarded. Any three schedules award 0 marks.

(d) Following the hint, we observe that at most one interval s_j that is active at midnight can be included in the optimal schedule. Given s_j, then we can remove all intervals that overlap with s_j from consideration and use the optimal algorithm for the standard interval scheduling problem for the remaining intervals. Since we don't know s_j, we can try that for each possible s_j and keep the solution with the largest cardinality. If none of the jobs that are active at midnight are included in the schedule, then we can again run the standard interval scheduling optimal algorithm for the set of all jobs.

Marking: 3 marks for the correct answer. The solution does not have to entirely formal or detailed. A solution where the idea is correct but some details might be missing will be awarded full marks.

PART B:

1. (a) Both are $\Theta(1)$. In the best case, no resizing is involved, so we just perform a fixed number of constant-time operations. [1 mark for each $\Theta(1)$, 2 marks for justifications.]

   (b) Any correct implementation accepted, e.g.

```
resize(A,i,j,m):
    B = new array[m]
    if i ≤ j
        for k = i to j-1
            B[k−i] = A[k]
        return (B,0,j−i)
    else
        for k = i to |A|−1
            B[k−i] = A[i]
        for k = 0 to j−1
            B[|A|−i+k] = A[k]
        return (B,0,|A|−i+j)
```

   Starting at position 0 in B seems the obvious thing to do, but isn't required. [4 marks for the right idea, 4 marks for correct details.]

   (c) Both are $\Theta(n)$. For push, when a resize is performed, the dominant contributions are an array allocation of size $2n$ and a copying of n queue items in total. Likewise, for pop with a resize, we have an array allocation of size $n/2$ and a copying of $n/4$ queue items (note that a resize will be triggered as soon as the queue size hits $n/4$). [1 mark for each $\Theta(n)$, 2 marks in total for justifications.]

   (d) We currently have n queue items in an array of size $2n$. So if the next resize is an expansion, this will require n further queue items, i.e. at least n pushes. If the next resize is a contraction, this will require that the number of queue items shrinks to $n/2$, i.e. we need $n/2$ pops. (Note that $2n \geq 10$ since the array size is never allowed to dip below 10.) So it takes a minimum of $n/2$ operations to trigger another resize. [2 marks for right answer, accepting anything close to $n/2$. 3 marks for justification.]

   For the record, the situation for pop is as follows (this is not required, but is alluded to in part (e)). We currently have $n/4$ queue items sitting in an array of size $n/2$. So by the same reasoning as above, it would take a further $n/4$ operations to trigger another expansion, or (if $n/2 \geq 10$) $n/8$ operations to trigger another contraction. So minimum number of operations is $n/8$ if $n \geq 20$, or $n/4$ otherwise.

   (e) In a long run of operations, each push/pop that performs a resize takes time $\Theta(n)$, but is then followed by at least $n/8 - 1 = \Omega(n)$ operations that require no resize and so each take $\Theta(1)$ time. So by spreading the cost of the resize over these $\Omega(n)$ operations, we see that the amortized time per operation is $\Theta(1)$. [2 marks for $\Theta(1)$, 2 marks for explanation.]

2. (a) In this example all of the items obviously can fit in the knapsack. The question is however about running the dynamic programming-based algorithm. The table that the algorithm produces is shown in Table 1.

Marking: 8 marks if the student draws the correct table, marks will be deducted for mistakes in the table.

(b) For the calculation of $M[2,3]$ we have that $w_2 = 2$ and $w = 3$ so the item fits in the solution and we can use the recurrence relation

$$M[2,3] = \max\{M[1,3], v_2 + M[1,1]\} = \max\{2,5\} = 5.$$

Similarly, for $M[3,5]$ we have that $w_3 = 4$ and $w = 5$ so the item fits in the solution and we can use the recurrence relation

$$M[3,5] = \max\{M[2,5], v_3 + M[2,1]\} = \max\{5,6\} = 6.$$

Marking: 1 mark each for the correct explanation of how $M[2,3]$ and $M[3,5]$ are derived.

(c) The dynamic programming-based algorithm is not a polynomial time algorithm, it is only pseudopolynomial time. Its running time is $O(nW)$, and it is only polynomial time if the weights $w_1, \ldots, w_n, W$ are given in the input in unary representation. The problem is in fact NP-complete, so a polynomial-time algorithm is possible only if P = NP.

Marking: If the student answers "no" to the first question correctly, they will awarded 1 mark; they don't necessarily need to write down the running time. They should mention that the problem is NP-complete or NP-hard to get any extra marks. If they answer the second question as "no" then 1 mark will be deducted, if they don't mention "unless P = NP. If they answer "yes" to the first question incorrectly, but right down the correct running time as $O(nW)$ then they will be awarded 1 mark.

(d) For non-optimality, consider an instance with $w_1 = 1, w_2 = 2, w_3 = 3$ and $v_1 = 2, v_2 = 3, v_3 = 4$ and $W = 5$. The items $1, 2, 3$ are already sorted in terms of non-decreasing v_i/w_i, so the algorithm will consider them in that order. The two candidate sets to be included in the solution are thus $\{1,2\}$ for a total value of 5 and $\{3\}$ for total value of 4, and $\{1,2\}$ will be selected. We observe however that it is possible to choose the set $\{2,3\}$ with weight exactly 5, which has a total value of 7.

Marking: Any example that shows non-optimality will be awarded the full 2 marks, otherwise these 2 marks will not be awarded.

(e) We now argue that the approximation ratio of the algorithm is at most 2. Let x be the solution outputted by Greedy B, and let $v(x)$ be its value. Let x^* be an optimal solution to the (0/1)-Knapsack instance (where x^* is a set of items) and let $v(x^*)$ be the total value of that solution. Furthermore, let z^* denote the

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	2	2	2	2	2	2	2
2	0	2	3	5	5	5	5	5
3	0	2	3	5	5	6	7	9

Table 1: The table M output by the dynamic programming-based algorithm.

solution to the *fractional* version of the (0/1)-knapsack instance, where items are allowed to be partially added to the knapsack and let $v(z^*)$ be its value. Obviously, it holds that $v(x^*) \leq v(z^*)$. Now, observe that since the bang-per-buck greedy algorithm (Greedy A plus the largest fraction that fits from the next item) is optimal for the fractional version (this was shown in the lectures), it holds that z^* consists of items $a_1, \ldots, a_{i-1}$ and a λ fraction of item a_i.
We have

$$v(x^*) \leq v(z^*) = [v(a_1) + \ldots + v(a_{i-1})] + \lambda \cdot v(a_i) \leq 2 \cdot v(x),$$

where the last inequality holds since $v(x) = \max\{v(a_1) + \ldots + v(a_{i-1}), v(a_i)\}$.
Marking: For the upper bound, 6 marks will be awarded for a complete solution, partial marks will be awarded for solutions that are in the right direction.

(f) The running time of the Greedy B algorithm is $O(n \log n)$ needing to sort the items via their ratios, plus the time required for computing the sums of values and weights and the comparisons to select the output. This is a polynomial-time algorithm. This is not the best possible approximation algorithm for the problem. As was discussed in the lectures, there is a Fully Polynomial Time Approximation Scheme (FPTAS) for the algorithm, i.e., an algorithm which runs in time polynomial in the input size and $1/\epsilon$ and produces an $1+\epsilon$ approximation to the optimal solution.
Marking: 1 mark will be awarded for the correct running time. 2 marks will be awarded for identifying that there is an FPTAS and therefore Greedy B is not the best approximation algorithm.

3. (a) The LL(1) parsing algorithm executes thus:

Operation	Input left	Stack
	(x y)	T
Lookup T,(	(x y)	BPC
Lookup B,(	(x y)	(PC
Match (	x y)	PC
Lookup P,x	x y)	TTC
Lookup T,x	x y)	xTC
Match x	y)	TC
Lookup T,y	y)	yC
Match y	)	C
Lookup C,)	)	)
Match)		
Success!		

[Roughly 1 mark per correct line.]

(b) The grammar would no longer be LL(1). If we were expecting a T-phrase and saw '(' in the input, we wouldn't know whether to expand T to BPC or BTC. [1 mark for 'no', 2 marks for explanation.]

(c) The rule T $\rightarrow$ BPC violates Chomsky normal form as the RHS is ternary. We can fix this e.g. by introducing a new non-terminal U and replacing this rule by T $\rightarrow$ BU and U $\rightarrow$ PC. [1 mark for picking the right rule, 2 marks for the fix.]

(d) The CYK chart is:

	(	x	y	)
(	B			T
x		T	P	U
y			T	
)				C

with the evident backtrace pointers leftward and downward from the non-diagonal entries. [1 mark for a table of the right form, 3 marks for the entries, 2 marks for the pointers.]

(e) In the presence of ternary rules, CYK-style parsing is less efficient because for each segment of input we now need to consider all possible 3-way splits rather than just 2-way. This bumps up the overall runtime from $\Theta(n^3)$ to $\Theta(n^4)$. [2 marks for the idea, 1 mark for some relevant asymptotic assertion.]