UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

INFR08026

Thursday 9th May 2024

13:00 to 15:00

INSTRUCTIONS TO CANDIDATES

1. **Answer all five questions in Part A, and two out of three questions in Part B. Each question in Part A is worth 10% of the total exam mark; each question in Part B is worth 25%.**
2. **If all three questions in Part B are attempted, only questions 1 and 2 will be marked.**
3. **This is a NOTES NOT PERMITTED, CALCULATORS PERMITTED examination. Notes and other written or printed material MAY NOT BE CONSULTED during the examination. CALCULATORS MAY BE USED IN THIS EXAMINATION.**

THIS EXAMINATION WILL BE MARKED ANONYMOUSLY

PART A

1. (a) Give formal definitions of the relations $f = O(g)$, $f = \Omega(g)$, $f = \Theta(g)$, where f, g are functions from $\mathbb{N}$ to the positive reals. *[3 marks]*

   (b) Is it true that $5\sqrt{n} = O(n)$? Rigorously justify your answer with reference to the formal definition of O. *[3 marks]*

   (c) Is it true that $5\sqrt{n} = \Theta(n)$? Rigorously justify your answer with reference to the formal definition of Θ. *[4 marks]*

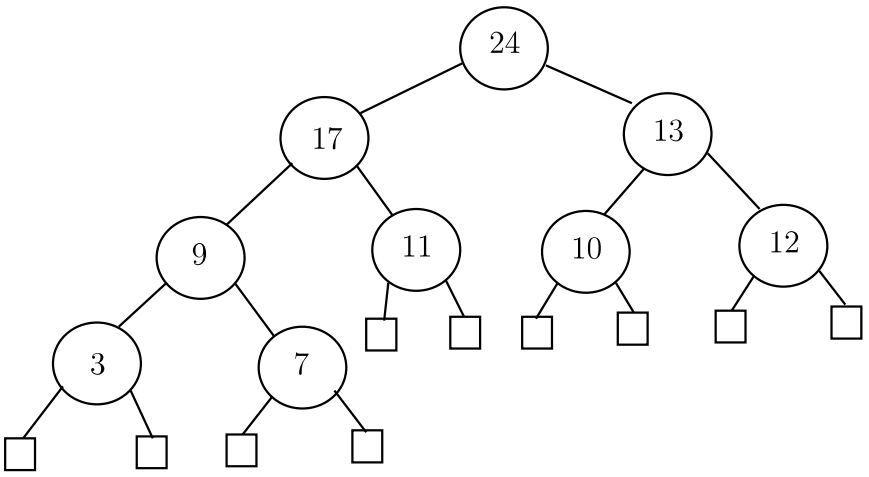
2. Recall from lectures that for natural numbers a, n, m where $m > 0$, the value of $a^n \mod m$ may be efficiently computed by a recursive procedure $\mathsf{Expmod}(a, n, m)$ with $n = 0$ as a base case.

   (a) Show how this procedure executes on the inputs $a = 2$, $n = 22$, $m = 23$, by showing what calls to Expmod take place, how these are nested, and what result is returned for each call. (You need not give the pseudocode for the procedure.) *[6 marks]*

   (b) Consider now the task of computing $\mathsf{Expmod}(2, n, 23)$ for various inputs n using this algorithm. Let $T(n)$ denote the runtime of this computation on input n. Write down an *asymptotic recurrence relation* satisfied by $T(n)$, paying attention to floors and/or ceilings. You may assume arithmetic operations such as div and mod execute within constant time. *[2 marks]*

   (c) Give Θ-estimates for $T(n)$ (i) as a function of n; (ii) as a function of d, the number of digits in the decimal representation of n. You need not justify your answers. *[2 marks]*

3. Consider the (Max) Heap below, showing the keys of the items stored within it.



Question continues on next page

Question continues from previous page

(a) Write down the array corresponding to this tree representation. [*1 mark*]

(b) Compute the resulting heap (either in tree or array format) after the following operations in sequence, showing the result after each operation:

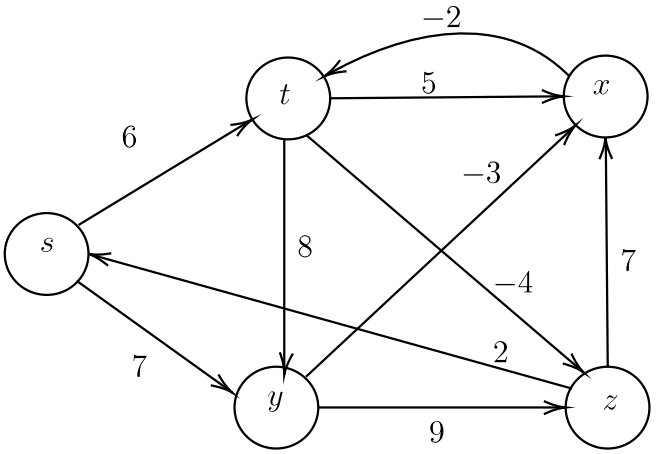
Max-Heap-Extract-Max, Max-Heap-Insert(6), Max-Heap-Extract-Max, Max-Heap-Insert(16)

[*6 marks*]

(c) On a general heap, what is the worst-case asymptotic running time of Max-Heap-Insert? Provide a brief argument to support your claim. [*3 marks*]

4. (a) Let G be a directed graph with positive and negative edge weights, and no cycles of negative total cost. Consider the Bellman-Ford algorithm for target node v, to compute the costs of all the minimum-cost paths from any node u to v. The algorithm starts with initialising $M[0, v] = 0$. Is it ever possible in the course of the execution of the algorithm for $M[i, v]$ to be anything other than 0? Justify your answer. [*2 marks*]

(b) Consider the directed graph G below. Run the Bellman-Ford algorithm for target node z, to compute the costs of all the minimum-cost paths from any node u to z. Indicate the values of $M[i, u]$ for any $i = 0, \ldots, 4$ and any node u of G. You do not need to compute the paths themselves. [*5 marks*]



(c) Assume that we would like to modify the algorithm to find the costs of all the minimum-cost paths from node z to any node u in the graph.
Write the appropriate recurrence relation to compute $M[i, u]$ in this modification. [*3 marks*]

5. (a) Consider the following input to the interval scheduling problem: $[0, 4]$, $[2, 3]$, $[1, 5]$, $[4, 6]$, $[7, 9]$, $[8, 10]$. What is the outcome of the EFT (earliest finishing time) algorithm on this input? [*2 marks*]

Question continues on next page

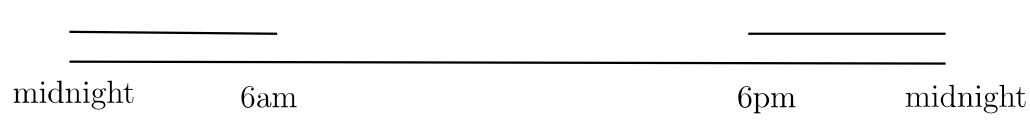
Question continues from previous page

(b) Consider the interval scheduling problem, and the following greedy algorithms presented in the lectures for choosing which interval to schedule next: EST (earliest starting time), and SI (smallest interval). Provide concrete examples of inputs to the problem that prove that the approximation ratios of these algorithms are at least $m - 1$, and 2 respectively. *[3 marks]*

Now consider a variant of the interval scheduling problem: We have a processor which can operate 24 hours a day (from midnight to midnight), every day, and there is a set of requests to run *daily* jobs on the processor. Each job is represented by an interval with a starting time s_i and a finishing time t_i, with $s_i < t_i$. If the job is accepted to run on the processor, it must run continuously, every day, for the period between its start time and finishing time. Note that certain jobs can start before midnight and end after midnight. Given a set of n such jobs, the goal is to accept as many as possible, subject to the constraint that the processor can run at most one job at any given point in time (i.e., two jobs with overlapping intervals cannot be both accepted).

(c) Consider the following input to this problem (6pm, 6am), (9pm, 4am), (3am, 2pm), (1pm, 7pm). Write down all the optimal solutions to this problem on this input.

Hint: It might be helpful to think of the 24-period as a circle, with intervals that start before midnight and end after midnight as "wrapping around", as in the figure below. *[2 marks]*



(d) Describe a polynomial-time greedy algorithm that solves the variant of the previous subquestion optimally, by using an optimal algorithm for the (standard) interval scheduling problem as a subroutine. For simplicity, you may assume that no two jobs have the same starting or finishing times. You do not have to provide pseudocode for the algorithm.

Hint: Note that if the schedule contains an interval that is active at midnight, then it cannot contain any other interval that is active at midnight.

[3 marks]

PART B

1. In this question we consider the implementation of queues via circular buffers (i.e. wraparound arrays). Specifically, a queue of integers will be represented by:

   - an integer array A (of size |A|) to store the queue items,
   - an integer i giving the current index of the head of the queue,
   - an integer j giving the index of the tail position into which the next added item will be inserted.

   In this version, the array can be expanded when necessary to accommodate more items, and can also be contracted when the number of items becomes small in order to economize on memory.

   In the initial state we have $|A| = 8$ and $i = j = 0$. The push and pop operations for the queue are given by the following pseudocode. Here the expansions and contractions are managed with the help of a function resize which it will be your task to implement in part (b). Note that because the expansions and contractions are by a factor of 2, the array size |A| will always be a power of 2.

```
push(x):
    A[j] = x
    j = (j+1) mod |A|          # new tail position allowing wraparound
    if j = i                   # current array is full
       (A,i,j) = resize(A,i,j,|A|*2)
    return

pop():
    if j = i then FAIL         # queue is empty
    x = A[i]
    i = (i+1) mod |A|          # new head position allowing wraparound
    q = (|A|+j−i) mod |A|      # current queue size
    if |A| ≥ 16 and q ≤ |A|/4
       (A,i,j) = resize(A,i,j,|A|/2)
    return x
```

   (a) Give Θ-estimates for the *best-case* runtimes of push and pop respectively. Briefly justify your answers. *Question continues on next page* [*4 marks*]

Question continues from previous page

(b) To complete the implementation, give pseudocode for a function resize(A,i,j,m) which creates a fresh array B of size m, copies the current queue contents to a sensible portion of B, and returns a triple (B,i',j') where i',j' are the appropriate head and tail positions for the new queue representation. Note that m may be either larger or smaller than |A|, but you may assume the current number of queue items is less thatn m. You may find it helpful to treat the cases $i \leq j$ and $j < i$ separately. [*8 marks*]

(c) In the light of your implementation of resize, give Θ-estimates for the *worst-case* runtimes of push and pop respectively, as functions of $n = |A|$. Briefly justify your answers. [*4 marks*]

(d) Suppose we have just performed a push which has expanded the current array size from n to $2n$. What is the minimum number of operations (pushes or pops or a mixture) now required to trigger a further call to resize? Justify your answer. [*5 marks*]

(e) Suppose that we perform a long sequence of push and pop operations (which may be in any order). Give an asymptotic estimate for the *amortized run-time cost per operation.* Give an informal but clear explanation of your reasoning. You may assume here, without explicit justification, that the situation for pop is asymptotically analogous to the situation for push as established in part (d). [*4 marks*]

2. Consider the (0/1)-Knapsack problem: We are given a set of n items $1, 2, \ldots, n$, each with a positive integer weight w_j and a positive integer value v_j, as well as an integer W which we call the capacity of the knapsack; each w_j and v_j as well as W are given by the binary representation. The goal is to choose a subset S of the items such that $\sum_{j \in S} w_j \leq W$ and $\sum_{j \in S} v_j$ is as large as possible. We may assume that $W \geq w_i$ for every $i \in \{1, 2, \ldots, n\}$, i.e., each item individually fits in the solution.

   (a) In the lectures we saw a dynamic programming-based algorithm that computes the (value of) the optimal solution to the (0/1)-Knapsack problem. Consider an input to the problem given by: $w_1 = 1$, $w_2 = 2$, $w_3 = 4$, $v_1 = 2$, $v_2 = 3$, $v_3 = 4$ and $W = 7$. Draw the (4×8)-dimensional table M that would be the outcome of the aforementioned algorithm, in which the rows correspond to indices of items and the columns correspond to weights. [*8 marks*]

   (b) Explain explicitly how you have calculated $M[2, 3]$ and $M[3, 5]$, making reference to the recurrence relation used by the algorithm for these particular values of i and w. [*2 marks*]

   (c) Is the dynamic programming-based algorithm mentioned above a polynomial-time algorithm? If it is, write down its asymptotic running time. If it is not, is there a different algorithm that solves the (0/1)-Knapsack problem optimally in polynomial time? Justify your answer. [*4 marks*]

   Now consider two greedy algorithms for the (0/1)-Knapsack problem, which differ only in their final step:

   Both algorithms: Sort the items in terms non-increasing ratio v_j/w_j. Greedily include items $a_1, \ldots, a_{i-1}$ in the solution according to that order. If all items can be accommodated within the given weight W, return this as the solution. Otherwise, stop when an item a_i is encountered for which the total weight of items included so far plus w_{a_i} is larger than W.

   Greedy A: Return the set $\{a_1, \ldots, a_{i-1}\}$ as the solution.

   Greedy B: If $\sum_{j=1,\ldots,i-1} v_{a_j} \geq v_{a_i}$, return $\{a_1, \ldots, a_{i-1}\}$ as the solution, otherwise return just $\{a_i\}$.

   (d) Show that Greedy B does not always output an optimal solution. [*2 marks*]

   (e) Show that the approximation ratio of Greedy B is at most 2. [*6 marks*]

   [*Hint:* You may want to make reference to the *fractional* version of the problem, where any fraction of any item can be included in the solution, and use the fact that the algorithm that returns the solution of Greedy A plus the largest percentage of item a_i that fits in the solution, solves the fractional version optimally.]

Question continues on next page

Question continues from previous page

(f) What is the asymptotic running time of the Greedy B algorithm above? Does the algorithm run in polynomial time or in pseudopolynomial time? Justify your answer. Is it possible to design a polynomial time algorithm with approximation ratio better than that of the Greedy B algorithm for the $(0, 1)$-Knapsack problem? Justify your answer. [*3 marks*]

3. Below is an LL(1) parse table for a context-free grammar that generates expressions such as (x(yx)), representing binary trees with leaves labelled x or y. The start symbol is T; the terminals and other non-terminals can be read off from the table.

	x	y	(	)	$
T	T →x	T →y	T →BPC		
P	P →TT	P →TT	P →TT		
B			B →(		
C				C →)	

(a) Show how the LL(1) parsing algorithm executes on the input string

(xy)

displaying at each stage the operation performed, remaining input and stack state. Use the table format used in lectures. [*10 marks*]

(b) Suppose now that the extra rule

T → BTC

were added to our grammar. Would the resulting grammar still be LL(1)? Justify your answer. [*3 marks*]

(c) The grammar embedded in the above parse table is nearly in *Chomsky Normal Form* (CNF). Which rule violates the requirements of CNF? Explain how the grammar may be modified to produce an equivalent grammar that is in CNF. You need not follow the general algorithm for conversion to CNF, but should apply just the part of this algorithm that is needed to fix the problem. [*3 marks*]

(d) Using your CNF version of the grammar, construct a CYK parse chart, again for the string (xy). Include backtrace pointers to show how the subphrases are constructed. [*6 marks*]

(e) It is possible, in principle, to adapt the CYK parsing algorithm so that it would work for our original grammar. What is the disadvantage in doing this? Give a precise quantitative statement to support your claim. [*3 marks*]