UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

INFR08026 INFORMATICS 2: INTRODUCTION TO ALGORITHMS AND DATA STRUCTURES

Monday 15 th August 2022

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. **This is an Open Book examination.**

Convener: D.K.Arvind
External Examiner: J.Gibbons

THIS EXAMINATION WILL BE MARKED ANONYMOUSLY

PART A

1. (a) State which of the following claims are true and which are false, without needing to justify your answers. The functions are assumed to be defined on the domain $\mathbb{N}^+$ of positive integers. [*5 marks*]

   i. $\lg(e^n) = \omega(n)$
   ii. $n^3 + 16n^2 - 100 = \Omega(n^2)$.
   iii. $\lg(n^2 + n^5) = O(n)$
   iv. $2^n = \Omega(n!)$
   v. $2^n + n^{100} = \Theta(n^{100})$.

   (b) Consider an implementation of a queue via a "wraparound" array of size 10. Initially the queue contains 5 items, with the head item at position 3 and the tail item at position 7.

   We then try to perform 3 'dequeue' operations followed by 6 'enqueue' operations.

   Explain the position of the head pointer and the tail pointer after these 9 operations have finished, justifying your answer. [*5 marks*]

2. Given an array A whose length is a power of two, the following pseudocode performs a non-random *shuffle* of the elements of A by splitting A into two halves, recursively shuffling these, and interleaving the results:

```
interleave (B,C):   # interleaves two arrays of equal length
    D = new array(|B|*2)
    for i = 0 to |B|−1
       D[2i] = B[i]
       D[2i+1] = C[i]
    return D

shuffle (A):
    n = |A|
    if n ≤ 2 then return A
    else
        B = shuffle (A[0,...,n/2−1])
        C = shuffle (A[n/2,...,n−1])
        return interleave (B,C)
```

   (a) Give a tight asymptotic estimate of the runtime of interleave(B,C) as a function of n, the size of B. [*2 marks*]

   (b) Let $T(n)$ denote the runtime of shuffle(A) on an array A of length n (assumed to be a power of two). Write down an asymptotic recurrence relation satisfied by $T(n)$. [*4 marks*]

(c) Using the Master Theorem, obtain a tight asymptotic estimate for $T(n)$. State the relevant values of the constants a, b, k appearing in the Master Theorem. [*4 marks*]

3. We have seen a number of sorting algorithms in this course, and in this question we consider Insertsort, Mergesort, and Quicksort and some particular details of their behaviour and performance.

   (a) For which of these three algorithms is the worst-case running time most similar (asympotically) to the best-case running time? Write a few sentences explaining the reason for this, referring to the structure of the algorithm. [*3 marks*]

   (b) For which of these three algorithms do we see the biggest gap between asymptotic best-case running time and asymptotic worst-case running time? Again, write a few sentences to explain why we see this significant difference, referring to the structure of the algorithm. [*3 marks*]

   (c) Consider a situation where we are banned from using any direct sorting algorithm, but we have access to Red-Black trees. Describe how could we exploit this data structure to achieve a full sort of n integer keys in $O(n \lg(n))$ time. [*4 marks*]

4. (a) Compute the edit distance of the two strings "lord" and "board", giving details of the distance matrix d. You do not need to build the partner matrix a but please show the optimum alignment of the strings. [*5 marks*]

   (b) The edit distance algorithm works with respect to two tables of dimensions $(n+1) \times (m+1)$, where m, n are the lengths of the input strings. We demonstrated that it has worst-case running-time $\Theta(nm)$.

   The seam-carving dynamic programming algorithm also works on tables of size $(m+1) \times (n+1)$, and it also has worst-case running-time $\Theta(nm)$. Explain why we could infer the asymptotic running-time of seam carving *by comparison* to edit distance, making reference to the table dimensions and recurrence details of both problems. [*3 marks*]

   (c) The running time of both of the algorithms (mentioned in part (b)) is $\Theta(mn)$ but they take inputs of different types and sizes. For which algorithm would we consider the running-time to be asymptotically lower *with respect to the size of the input*? [*2 marks*]

5. Consider the following grammar (with start symbol S) for even-length palindromic strings over $\{a, b\}$:

$$S \rightarrow \epsilon \mid aSa \mid bSb$$

   Convert this grammar to one in Chomsky Normal Form. The resulting grammar should generate exactly the same strings as the above, with the exception of ϵ. [*10 marks*]

PART B

1. In this question we consider open-address hash tables with probing. Specifically, we consider a hash table T of some size $m \geq 3$, with cells indexed by elements of the set $[m] = \{0, \ldots, m-1\}$. Keys will be ordinary natural numbers, and our table uses the linear hash-probe function

$$\# \ : \ \mathbb{N} \times [m] \ \rightarrow \ [m]$$

defined by

$$\#(k, i) \ = \ (k + i) \bmod m.$$

To insert an entry for a new key, we probe cells in the way described in lectures until we find a free cell; the number of cells thus probed is called the number of *attempts* involved in the insertion. We shall be interested here in the *total* number of attempts involved in a whole sequence of insertions involving keys $k_1, \ldots, k_n$, starting from an empty table.

(a) For an arbitrary $n \leq m$, suggest a sequence of distinct keys $k_1, \ldots, k_n$ that exhibits the best-case behaviour: i.e. one that minimizes the total number of attempts. Give an exact formula for this total number of attempts as a function of n. [*3 marks*]

(b) Again for an arbitrary $n \leq m$, suggest a sequence of *distinct* keys $k_1, \ldots, k_n$ exhibiting the worst-case behaviour: i.e. one that maximizes the total number of attempts. (Recall that individual keys may be larger than m.) Give an exact formula for this number of attempts as a function of n, briefly justifying your answer and explaining why this is indeed the worst case. Also identify the asymptotic growth rate of your function (no justification required). [*8 marks*]

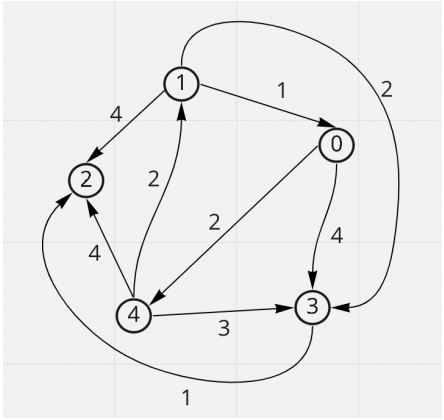
(c) Now suppose we replace the above linear hash-probe function # by some other hash-probe function $\#'$, also of type $\mathbb{N} \times [m] \ \rightarrow \ [m]$. (Recall that for any key k, it is required that $\#'(k, 0), \#'(k, 1), \ldots, \#'(k, m-1)$ is a *permutation* of $[m]$.) Could this result in a formula for the worst-case number of attempts that was different from the one you gave in part (b)? Justify your answer.

[Hint: How many permutations of m are there? How many keys are there?] [*7 marks*]

(d) Our linear hash-probe function # has the interesting (and non-obvious) property that the total number of attempts required to insert any set of keys $k_1, \ldots, k_n$ (where $n \leq m$) is independent of the *order* in which these keys are inserted. Show by an example that there are other hash-probe functions $\#'$ (of the same type) that do *not* enjoy this property. You need not give an explicit mathematical formula for a suitable $\#'$, but your solution should make it clear that such hash-probe functions do exist. [*7 marks*]

2. In this question we consider the problem of computing shortest paths in directed graphs. We use $G = (V, E)$ to denote our directed graph (with $(u, v) \in E$ interpreted with the direction $u \to v$) and we consider both "Single-source shortest paths (SSSP)" (via Dijkstra's algorithm) and "All pairs shortest paths (APSP)" (via the Floyd-Warshall dynamic programming algorithm).

   For parts (a) and (b) we will consider the following example graph:



   (a) Execute Djikstra(G, 0) from start vertex 0 for the example graph, showing the updates to the set S (of completed vertices), the array d (of "best so far" distances) and the array π (of parent nodes for "best so far" paths) at each step. The initial settings of these parameters, before we consider the edges outgoing from 0, will be: [*8 marks*]

$S = \{0\}$ $\quad d$

0	∞	∞	∞	∞

$\quad \pi$

–	NIL	NIL	NIL	NIL

   (b) Execute FloydWarshall(G), showing details of each of the matrices $D^{<1}$, $D^{<2}$, $D^{<3}$, $D^{<4}$, $D^{<5}$ as you progress through the stages of the algorithm, and writing a sentence of explanation for each. The initial matrix $D^{<0}$ is [*10 marks*]

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & \infty & \infty & 4 & 2 \\ 1 & 0 & 4 & 2 & \infty \\ \infty & \infty & 0 & \infty & \infty \\ \infty & \infty & 1 & 0 & \infty \\ \infty & 2 & 4 & 3 & 0 \end{bmatrix} \end{array}$$

   (c) Our Floyd-Warshall algorithm for APSP, which has worst-case running-time $\Theta(n^3)$ on a graph of n vertices, operates with an Adjacency matrix representation of the input graph. If we instead stored the input graph in Adjacency list format, would it help, hinder or make no difference to the running-time? Justify your answer with a couple of sentences. [*4 marks*]

(d) Our Dijkstra's algorithm for SSSP works with an Adjacency list representation of the input graph, and has worst-case running-time $\Theta((n+m)\lg(n))$ when an augmented Min-Heap is used to manage the choice of "best fringe vertex". Here m is the number of edges in E. If we instead stored the input graph as an Adjacency matrix format, would it help, hinder or make no difference to the running-time? Justify your answer with a couple of sentences. [*3 marks*]

3. In this question we consider the NP-complete problem Independent Set, and some simple heuristic approaches to the problem. The input to this problem is an undirected graph $G = (V, E)$.

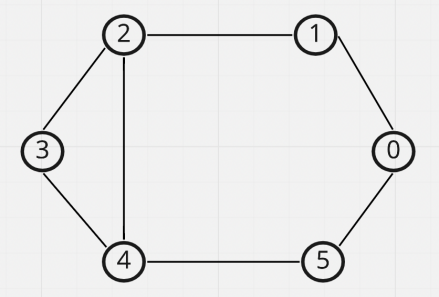
   One simple heuristic for Maximum Independent Set that we will consider is the following "Greedy" approach:

```
GreedyIS(V, E):
    V' ← V, E' ← E
    I ← ∅
    while V' ≠ ∅
        choose v ∈ V' of min degree in E'
            (break ties in pref of lower index)
        I ← I ∪ {v}
        V' ← V' \ ({v} ∪ Nbd(v))
        E' ← E' − {e : v an endpoint of e}
    return |I|
```

   Note: we always select a vertex of "minimum *residual* degree" (degree being computed in terms of the remaining vertices/edges).

   (a) Justify that GreedyIS is *polynomial-time*. [*5 marks*]

   (b) Iterate the steps of GreedyIS on the following graph to obtain an Independent set, and explain why it is *not* a maximum-sized Independent Set for this graph. [*5 marks*]



   (c) Suppose we consider the special case of Independent Set where the input graph is a *tree* $T = (V, E)$ with an identified root $r \in V$ (there are no other limitations on structure). This means that for every $v \in V(T)$, the subtree rooted at v is well-defined - we will refer to this subtree as T_v.

   For every $v \in V(T)$, we define the following:

   - $\kappa_{v,0}$ to be the size of the maximum-sized Independent set of subtree T_v which does not include v itself.
   - $\kappa_{v,1}$ to be the size of the maximum-sized Independent set of subtree T_v with v belonging to the independent set.

Develop a *pair of recurrences* which express the value of $\kappa_{v,0}$ (and similarly of $\kappa_{v,1}$) in terms of the κ values of the child nodes of v. Include the "base case" when v is a leaf, and then described how we can exploit the recurrences to design a polynomial-time algorithm to solve INDEPENDENT SET for a tree. [*10 marks*]

(d) In lectures we saw a $\leq_P$ reduction from 3-SAT to INDEPENDENT SET, and argued that this proves that INDEPENDENT SET is an NP-complete problem. Explain why the polynomial-time algorithm you develop in (c) does not contradict the NP-completeness, referring to the details of the $\leq_P$ reduction in your answer. [*5 marks*]