Version IB/2

EGT3
ENGINEERING TRIPOS PART IIB

---

Thursday 27 April 2023 2 to 4.30

---

**Module 4M26**

**ALGORITHMS AND DATA STRUCTURES**

*Answer not more than **three** questions.*

*All questions carry the same number of marks.*

*The **approximate** percentage of marks allocated to each part of a question is indicated in the right margin.*

*Write your candidate number **not** your name on the cover sheet.*

**STATIONERY REQUIREMENTS**
Single-sided script paper

**SPECIAL REQUIREMENTS TO BE SUPPLIED FOR THIS EXAM**
CUED approved calculator allowed
Engineering Data Book
DPO computer

## 10 minutes reading time is allowed for this paper at the start of the exam.

## You may not start to read the questions printed on the subsequent pages of this question paper until instructed to do so.

## You may not remove any stationery from the Examination Room.

1 (a) Let $L = [x_0, x_1, \cdots, x_{n-1}]$ be a list of 1D point coordinates on a line. Write the function, **pair**$(L)$, which:

(i) finds the pair of distinct points, $i$ and $j$, with the smallest distance, $D(i, j) = |x_i - x_j|$;

(ii) and outputs the indexes of the points of the pair found. The indexes should be outputed in an increasing order and separated with a comma (e.g. "0,3").

Your solution should have its run time complexity strictly smaller than $O(n^2)$. [30%]

(b) Sketch a detailed proof of correctness of your algorithm described in Part (a) and derive its worst-case time complexity. [15%]

(c) Give brief answers to the following questions.

(i) Explain what aspects of computing costs are *not* captured when you specify costs using Big-$O$ notation and why it is still often found useful? [10%]

(ii) Explain what is meant by "amortized" cost analysis, giving an example from the realm of data structures and algorithms where it is helpful. [10%]

(iii) Is it the case that every function of the form $f(n) = An^k$ is asymptotically bounded by $O(2^n)$, given that $A$ and $k$ are constants? Justify your answer. [10%]

(d) You are given an integer list, $L = [x_0, x_1, \cdots, x_{n-1}]$. Write the function, **subsequence**$(L)$, to find the length of the longest strictly increasing subsequence of the list, $L$. You can assume that the elements of the list, $L$, are unique. Note, a subsequence is a list that can be derived from another list by deleting some or no elements without changing the order of the remaining elements. Your algorithm should have run time complexity $\Theta(n^2)$. [10%]

(e) Provide key steps of an algorithm which could solve the task described in Part (d) in run time complexity $O(n \log n)$. [15%]

2 A B-tree of minimum degree $t$ is a data structure with the property that every node may contain at most $2t - 1$ keys. For this question, we will assume that all keys are distinct, that all nodes contain keys but not values, and that all nodes of the B-tree fit into memory (no disk read/writes are required).

(a) Implement a function to search for a given query key, $key$, starting from a given node, $u$, in a B-tree by completing the **search**(*self*, $u$, *key*) function below. If the query key exists in the B-tree, your function should return the node that contains the query key together with the index of the query key among its keys. Otherwise, your function should return `None`. For a B-tree of $n$ keys, your solution should run in $O(t \log_t n)$ time. [35%]

(b) Derive a mathematical expression for the minimum height of a B-tree containing $n$ keys in terms of its minimum degree $t$ and the number of keys $n$. [15%]

(c) In addition to searching for a key according to the value of the key, it is often useful to search for keys by their rank. The rank of a key is its position in the list of all keys stored in the B-tree, arranged in ascending order. For example, if a B-tree contains the keys 2, 4, 7 and 8, the rank of key 2 is 0, the rank of key 4 is 1 and so on.

(i) Describe an algorithm that returns the key with the desired rank. Your algorithm should exhibit $O(n)$ runtime complexity where $n$ is the total number of keys in the tree. [15%]

(ii) Now suppose that we can store an additional non-negative integer counter at each internal node of the B-tree that stores the count of all keys in its subtree (this count includes its own keys). Implement a function, **find_key_with_rank_efficient**($rank$), that makes use of counters to retrieve the key with a rank of $rank$ in a B-tree in $O(t \log_t n)$ time. [15%]

(iii) Describe a modified algorithm that improves the runtime complexity of returning the key with a given rank to $O(\log n)$ irrespective of how $t$ is chosen as a function of $n$. Note, $\log_t n \cdot \log_2 t = \log_2 n$ for $t > 0$. Your algorithm may redefine the meaning of the counter at each node under the constraints that: (i) the counter remains a non-negative integer; (ii) the counter values can be set in a single traversal of the whole tree. [20%]

3 (a) Write the function, **scheduler**($A$), which: (i) takes a list, $L$, as an input. This list in turn contains two lists, the first of which, $T$, stores tasks represented as strings (e.g. "homework"). The second list, $C$, stores pairwise constraints (e.g. "eat-homework") for the order of the tasks (e.g. "homework" should be performed only after "eat"); (ii) finds a schedule which respects all the pairwise constraints.

The schedule is represented as a sequence of tasks separated by "-". If no valid schedule is possible, the function outputs a string, "impossible". The worst case runtime of your solution should be not worse than $O(n + m)$, where $n$ is the number of tasks and $m$ is the number of constraints. Note that a function, **to_adjacency_representation**($T, C$), is provided to convert the task list, $T$, and the constraint list, $C$, into an adjacency list representation. You can ignore the costs of this operation in your reasoning. [35%]

(b) Give brief answers to the following questions.

(i) Derive worst-case time complexity of your proposed algorithm in Part (a). [10%]

(ii) Explain how depth first search (DFS) can be used to find the strongly connected components in a directed acyclic graph. [10%]

(iii) Explain how the longest cycle can be found efficiently in a directed graph in which each node has at most one outgoing edge. [10%]

(c) Answer questions about the graph illustrated in Fig. 1.

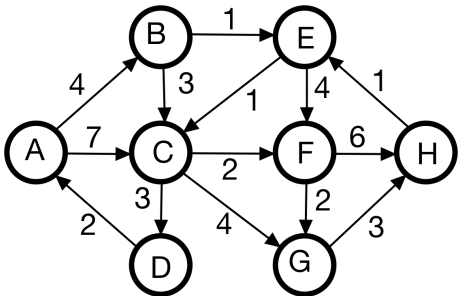


Fig. 1

(i) List vertices in the order in which they would be explored when running the Dijkstra's algorithm, starting from vertex $A$. A vertex is considered explored when *relaxation* operation is performed on all edges outgoing from the vertex in question. [5%]

(ii) List vertices in the order in which they would be explored when running the breadth first search algorithm starting from vertex $A$. A vertex is considered explored

when its color is set to *black*. In the case that there is ambiguity in the order of exploration any valid ordering can be provided. [5%]

(iii) Assume this graph is transformed into an undirected graph by replacing all directed edges with equivalent undirected edges with corresponding weights. Write out the edges of the minimum spanning tree discovered by Primm's algorithm in the same order as they would be added to this tree. The algorithm is run starting from vertex $A$. [5%]

(d) Write the function, **jobs**($L$), which:

(i) takes a list, $L$, as an input. The first element of this list is another list, $S$, containing names of students. The second element is also a list, $V$, containing the names of job vacancies. The third element is a list, $A$, containing job applications represented by the student's name, followed by a dash, "-", and the vacancy name to which this student wants to apply and;

(ii) finds and outputs the maximum number of job vacancies that can be filled. Each student can apply for multiple jobs but can only take at most one job and each job vacancy can be assigned at most to one student.

[20%]

4 Following a longstanding tradition, Jesus College Boat Club committee hosted the Fairbairn Cup in December 2022. This competition involved rowing crews that raced to cover a stretch of the River Cam in the shortest period of time. For each rowing crew participating in the competition, the committee has gathered the crew name and the time taken to complete the course. In order to publish a table that ranks rowing crews by performance, the committee would like to sort the crews according to the time that they took to complete the course.

(a) Provide an implementation of the heapsort algorithm (function **heapsort**($crews$)) that takes as input a list of $n$ rowing crews and returns them sorted by course time in ascending order. Your implementation should employ a max binary heap. The input is an unsorted list of dictionaries of the form [{"crew": <crew_name>, "time": <course_time_in_secs>}, $\cdots$], where <crew_name> is a string and <course_time_in_secs> is an integer. [35%]

(b) Derive a mathematical expression for the height of a binary heap as a function of the number of keys, $n$. [15%]

(c) Prove that construction of a max binary heap has a complexity of $\Theta(n)$, where $n$ denotes the number of keys. [15%]

(d) Suppose that all keys in the input list are identical. What is the Big-O runtime complexity of heapsort with respect to the number of keys, $n$, for this case? Explain your answer with reference to your implementation of heapsort. [10%]

(e) A *stable* sorting algorithm ensures that items that are equal maintain their input ordering in the sorted ordering. By default, heapsort is not a stable sorting algorithm. In the Fairbairns Cup, some crews completed the course in exactly the same time. Jesus College Boat Club has decided to rank crews with equal times according to the order that they appeared in the original unsorted list. By using $O(n)$ additional space, or otherwise, provide an implementation for the **stable_heapsort**($crews$) function below that will perform a stable sort of the rowing crews by course time. [25%]

**END OF PAPER**