

Question 1. [13 MARKS]

Short answer questions. You do not need to justify your answers unless indicated.

Part (a) [1 MARK]

The height a 2-3-4 tree depends on the number of keys in each node. Let h be the height of a 2-3-4 tree with n keys, then $low \leq h \leq high$. What are low and $high$?

Solutions.

$low = \log_4 n$ and $high = \log_2 n$.

Part (b) [1 MARK]

Is there a **best-case** scenario where `INSERT(key k)` for a 2-3-4 tree with n keys has complexity less than the worst-case complexity? In either case, what is this complexity?

Solutions.

No, insert will always belong to $\mathcal{O}(\log n)$.

Part (c) [1 MARK]

What is the **worst-case** time complexity of `DECREASE-PRIORITY(key k, priority p)` for a min heap of size n ?

Solutions.

$\mathcal{O}(\log n)$.

Part (d) [1 MARK]

Is there a **best-case** scenario where `INSERT(key k)` for a max heap of size n has complexity less than the worst-case? In either case, what is this complexity?

Solutions.

$\mathcal{O}(1)$ if we are inserting a key smaller than the parent nodes of the heap.

Part (e) [3 MARKS]

The *distance* between any two vertices in a graph is the length of the shortest path between them. The *diameter* of a graph is the maximum distance between two vertices in the graph. Give an algorithm to find the diameter of a graph. State the complexity of your algorithm.

Solutions.

Do BFS from every vertex remembering the length of the longest path and endpoints. Take the max of all longest paths, this is the diameter. Takes $\mathcal{O}(n^2 + nm)$ time because BFS takes $\mathcal{O}(n + m)$ and we run it n times.

Part (f) [3 MARKS]

Given a weighted graph with n vertices and m edges where all the edge weights are 10, give an efficient algorithm to find the minimum spanning tree. You may refer to any algorithm discussed in class. What is the complexity of your algorithm?

Solutions.

DFS/BFS $\mathcal{O}(m + n)$. This will find a MST since the edge weights are all the same and is more efficient than Prim's.

Part (g) [3 MARKS]

Does the order of operations matter when doing a sequence of deletions on a 2-3-4 tree? Justify your answer. (HINT: try deleting keys from a 2-3-4 tree containing 6 keys.)

Solutions.

Yes. Consider a tree on 6 elements (1..6) with 3 and 5 in the root. Deleting 4 and 6 results in a different tree than deleting 6 then 4.

Question 2. [12 MARKS]

Recall the ADT PLOT from assignment 1. An object of the ADT is a set of data points. Each point P is a co-ordinate (P_x, P_y) in the plane. The operations of the ADT are:

DELETE(S, P): Deletes the point P from the set S . If P was not in S then this operation does nothing.

ADD(S, P): Adds the point P to the set S .

MIN_VALUE(S, x_1, x_2): Returns the minimum value P_y such that $x_1 \leq P_x \leq x_2$ and P is a point in S . If there is no point P such that $x_1 \leq P_x \leq x_2$ then return 0.

In the following questions, you will be given a variation of the PLOT ADT and asked to give an implementation. You will be graded on the space and time complexity as well as the simplicity of your data structure. You may alter data structures from assignments or examples from class or come up with an entirely new one.

Part (a) [6 MARKS]

In the assignment, we assumed that the x co-ordinate for each point is unique (*i.e.*, no two points have the same x value). Now consider when **multiple points** with the same x co-ordinate or y co-ordinate are allowed. Describe a data structure to implement this ADT with the operation **MIN_VALUE**(S, x_1, x_2) replaced by a new operation, **MAX_VALUE**(S, x) described below.

MAX_VALUE(S, x): Returns the point P such that $P_x = x$ and P_y is maximum.

Make sure you carefully describe each *operation* and its *complexity* (**ADD**(S, P), **DELETE**(S, P), and **MAX_VALUE**(S, x)). You do not need to give the details of methods discussed in class, however you should state their complexity.

Solutions.

One solution is to use a 2-3-4 tree of 2-3-4 trees. The internal 2-3-4 tree must have a pointer to the max value. Then **ADD** and **DELETE** are both $\mathcal{O}((\log n)^2)$ time and are the standard 2-3-4 tree methods except if the item being added/deleted is the max y value for the given P_x , then we need to update the pointer to the max leaf in $\mathcal{O}(\log n)$ time. **EXTRACT_MAX** then simply requires searching for the x value and then returning the key pointed to by the root, so $\mathcal{O}(\log n)$ time.

Note, another solution is to make each item in the 2-3-4 tree a max heap. Implementing **ADD**(S, P), **DELETE**(S, P) consists of using the 2-3-4 insert(x)/delete(x) methods to locate the node with key x followed by the heap insert(P_y) and heap delete(P_y). Notice that locating P_y in the heap takes linear time. The complexity of **ADD**(S, P) $\mathcal{O}(\log n \cdot \log k)$ where k is the number of points with x coordinate equal to P_x . Since k can be $\mathcal{O}(n)$ this can be as bad as $\mathcal{O}((\log n)^2)$. The complexity then of **DELETE**(S, P) is $\mathcal{O}(\log n \cdot (k + \log k))$. If $k \in \mathcal{O}(n)$ then this is $\mathcal{O}(n)$. **MAX_VALUE** requires finding x and then **EXTRACT_MAX** in the heap which can be done in $\mathcal{O}((\log n)^2)$ time.

Part (b) [6 MARKS]

Suppose we now *add* the additional operation `GET_MAX_MODE(S)` to the ADT of part (a) where `GET_MAX_MODE(S)` is defined as follows:

`GET_MAX_MODE(S)`: Return the y value such that the number of points in S with $P_y = y$ is maximum.

Describe a data structure to implement this ADT. Make sure you explain how the other operations (`ADD(S, P)`, `DELETE(S, P)`, `MAX_VALUE(S, x)`) are affected. You may alter your data structure from (a) or give an entirely new one.

Solutions.

Add a max heap whose items are the pairs (y, n) where y is a P_y value and n is the number of occurrences of P_y in S . `GET_MAX_MODE(S)` can be achieved by just calling `GET_MAX()` on the heap. Each time a point (P_x, P_y) is inserted in the data structure from (a), simply call `INCREASE_PRIORITY(P_y, 1)`. If P_y isn't in the heap yet, just add a new item $(P_y, 1)$ to the heap. Similarly, for delete, just call `DECREASE_PRIORITY(P_y, 1)`. Complexity for `GET_MAX()` is constant but now `INCREASE_PRIORITY` and `DECREASE_PRIORITY` are not as efficient (linear time) as we need to search the heap for the location of P_y .

Can improve the complexity by adding a 2-3-4 tree of P_y values, which have 2 way pointers to the location of P_y in the heap—then the complexity is logarithmic.

... this page is for extra space.