

# CSE 101 Homework 8

Brian Masse, Taira Sakamoto, Emily Xie, Annabelle Coles

December 5, 2025

## 1. Verifying Heap

A *3-ary heap* is like a binary heap, except that every internal vertex has three children: *left*, *middle* and *right*. We will use  $v.\text{left}$ ,  $v.\text{mid}$ , and  $v.\text{right}$  to denote the three children. We let  $v.\text{value}$  denote the value stored at  $v$ . We consider only the case of complete trees, so all three sub-trees rooted at the three children will have the same size.

Here's an algorithm that verifies whether a balanced complete 3-ary tree has the min-heap property.

**IsHeap**( $r$  : root of 3-ary heap  $H$ )

- IF  $r$  is a leaf: return **true**
  - IF  $r.\text{value} > r.\text{left}.\text{value}$  return **false**
  - IF  $r.\text{value} > r.\text{mid}.\text{value}$  return **false**
  - IF  $r.\text{value} > r.\text{right}.\text{value}$  return **false**
  - IF **IsHeap**[ $r.\text{left}$ ] == **false** return **false**
  - IF **IsHeap**[ $r.\text{mid}$ ] == **false** return **false**
  - IF **IsHeap**[ $r.\text{right}$ ] == **false** return **false**
  - Return **true**
- (a) Give a recurrence for the time taken by **IsHeap** in terms of the total size,  $n$ , of the heap rooted at  $r$ . Explain your answer.
- (b) Give a time analysis for **IsHeap** by solving this recurrence, up to order. If you use the Master Theorem, be sure to give the values of the three parameters used, and the case it falls under.

## 2. Median of Sorted Lists

The following algorithm finds the median ( $n$ th smallest element) of two sorted lists of distinct integers,  $A[1..n]$  and  $B[1..n]$ . For simplicity, we assume  $n = 2^k$  is a power of 2.

MedianSorted( $A[1..n], B[1..n]$ )

- (a) IF  $n = 1$  THEN Return  $\min(A[1], B[1])$ .
- (b) IF  $A[n/2] < B[n/2 + 1]$
- (c)     THEN Return  $\text{MedianSorted}(A[n/2 + 1..n], B[1..n/2])$ .
- (d) IF  $A[n/2] > B[n/2 + 1]$
- (e)     THEN Return  $\text{MedianSorted}(A[1..n/2], B[n/2 + 1..n])$ .
- (f) IF  $A[n/2] = B[n/2 + 1]$
- (g)     THEN Return  $A[n/2]$ .

Questions

- (a) Give a recurrence for the worst-case time taken by MedianSorted in terms of  $n$ . Explain your answer.
- (b) Give a time analysis for MedianSorted by solving this recurrence, up to order. If you use the Master Theorem, be sure to give the values of the three parameters used, and the case it falls under.

### 3. Taxi Stand

At a taxi stand, there is a line of customers  $C_1, \dots, C_n$  and a line of taxis,  $T_1, \dots, T_m$ . Since taxis are licensed to drive in some towns but not others, each can serve some of the customers but not others. We need to assign each customer to a taxi, and the assigned taxis need to be in order, but not all taxis need to be assigned. We want to minimize the position of the last taxi to be assigned.

For example, if we have three customers  $A, B, C$  and four taxis, 1, 2, 3, 4 where 1 can serve  $A$  and  $C$ , 2 can serve  $A$  and  $C$ , 3 can serve  $B$  and  $C$ , and 4 can serve  $B$  and  $C$ , we could assign  $A$  to taxi 2,  $B$  to taxi 3 and  $C$  to taxi 4.

- (a) Fill out the rest of the problem specification with mathematical expressions clearly defining the constraints and objective:
- Instance: Bipartite graph  $G$  whose left vertices represent customers and right vertices represent taxis, and an edge  $(C, t)$  represents taxi  $t$  being able to serve customer  $C$ , with orderings  $C_1 \dots C_n$  and  $t_1 \dots t_m$  on the two sets of vertices, with  $m > n$ .
  - Solution Format: A list of taxis  $i_1, i_2, \dots, i_n$ , with taxi  $i_j$  assigned to customer  $j$ .
  - Constraints: Each customer must be assigned to a taxi that can drive them to their destination, i.e., (fill in) for every  $1 \leq j \leq n$ . And each customer must be assigned to a later taxi than the previous customer, i.e., (fill in)
  - Objective: (fill in)
- (b) Give a high level greedy strategy that solves this problem. Describe your strategy in words, don't give complete pseudo-code here.
- (c) Illustrate your strategy on the example given in the problem.
- (d) Prove your strategy finds the optimal solution, using any technique we've learned. You can state and prove the main lemmas rather than give the complete proof, such as stating and proving a modify-the-solution lemma but skipping the induction argument.

#### 4. Maximizing inner product

Say we are given two sequences of integers  $x_1, \dots, x_n; y_1, \dots, y_m$  with  $m \geq n$ . We want to find a subsequence of  $y_1, \dots, y_m$  of length  $n$ ,  $y_{i_1}, y_{i_2}, \dots, y_{i_n}$  with  $1 \leq i_1 < i_2 < \dots < i_n \leq m$  that maximizes the *inner product*  $\sum_{1 \leq j \leq n} x_j * y_{i_j}$ .

For example, if  $x$  is 3, 2, 4 and  $y$  is 5, 6, 1, 2, 8, the optimal subsequence is 5, 6, 8 giving objective  $3 * 5 + 2 * 6 + 4 * 8 = 59$ .

Here's a recursive algorithm to find the maximum inner product with a subsequence:

$MIPWS(x_1, \dots, x_n; y_1, \dots, y_m)$

- IF  $n = 0$  THEN return 0
- IF  $n = m$  THEN return  $x_1 * y_1 + MIPWS(x_2, \dots, x_n; y_2, \dots, y_m)$
- Return  $\max(x_1 * y_1 + MIPWS(x_2, \dots, x_n; y_2, \dots, y_m), MIPWS(x_1, \dots, x_n; y_2, \dots, y_m))$ .

Questions

- (a) Show the tree of recursive calls this algorithm makes on the above example. (You can either draw the tree, or list the recursive calls, using indentation to indicate nesting. You can use abbreviations like  $2 - 4, 2 - 5$  to mean calling the algorithm on  $x_2 \dots x_4; y_2 \dots y_5$ )
- (b) Give a recurrence for the total number of recursive calls this algorithm makes, in terms of  $n' = n + m$ .

In the rest of this part, you will give a dynamic programming algorithm for this problem. Your algorithm can be based on the above recursion, or some other case analysis.

- (c) Describe precisely the sub-problems that your dynamic programming algorithm solves. Define an array or matrix to hold these answers.
- (d) Describe the base cases in terms of your array or matrix.
- (e) Give a formula or pseudo-code saying how you compute larger sub-problems in terms of smaller sub-problems.
- (f) Explain the case-analysis that justifies the previous step.
- (g) Assemble the parts above into a dynamic programming algorithm, described in pseudocode. Be sure to specify the order in which sub-problems are solved, and what the final output is.
- (h) Give a time analysis for your dynamic programming algorithm in terms of  $n$ .
- (i) Show the array or matrix your algorithm produces on the example above.