NPTEL Questions

CP — July-Nov 2021

Week 2 — Sorting and Searching

1. (4 points) Consider a variation of Bubble Sort known as Cocktail Sort. In cocktail sort we perform two kind of traversals alternatively.

   The first kind of traversal moves from **left** to **right** comparing the adjacent elements. If the element on the left is greater than the one on right, we swap the elements just like in bubble sort.

   The second kind of traversal moves from **right** to **left** comparing the adjacent elements. If the element on the right is smaller than the one on left, we swap the elements. This traversal is the reverse of the first kind of traversal.

   (a) ($\frac{1}{2}$ point) Consider the problem of sorting the array $[2, 3, 4, 5, 6, 1]$, calculate the number of passes required by bubble sort to sort the given array. Ignore any extra passes which do not alter the array.
   A. 6 **B.** 5 C. 4 D. 3

   Explanation: With each bubble sort pass, we see 1 shifts one place to the left. Hence, it takes us 5 passes to sort the array using bubble sort.

   (b) ($\frac{1}{2}$ point) Again, consider the above problem, however now calculate the number of passes required by cocktail sort to sort the given array. Ignore any extra passes which do not alter the array. Note: We take each traversal as a separate pass over the array. **A.** 2 B. 4 C. 6 D. 3
   Explanation: With cocktail sort, the sorting would require just two passes on the data. First kind of traversal followed by the second kind of traversal

   (c) Statement A: There exists an array on which bubble sort takes $O(n)$ time but cocktail sort takes $O(n^2)$.

   Statement B: There exists an array on which cocktail sort takes $O(n)$ time but bubble sort takes $O(n^2)$.

   Choose the correct truth values of the above two statements.

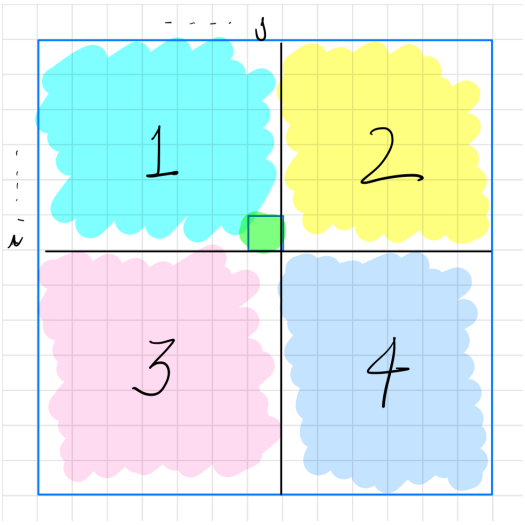
   A. Both statements are true.
   B. Statement A is True and B is False.
   C. Statement A is False and B is True.
   D. Both statements are false.

   Explanation: First, it's simple to see that in $2x$ passes, cocktail sort is able to achieve x bubble sort passes. Hence, we can claim that that in the worst case, cocktail sort performs 2 times more passes than bubble sort to sort a given array. So, if for an array instance, bubble sort takes $O(n)$ time, cocktail sort, in the worst case, will require $O(2n) = O(n)$ time. Hence, the first statement is **False**.
   Consider an array $[2, 3, ..., n, 1]$, to sort this using bubble sort, we would require n passes over the array, corresponding to a time complexity of $O(n^2)$, while cocktail sort can achieve the same in 2 passes, corresponding to a time complexity of $O(n)$. Hence, the second statement is **True**

   (d) What is the worst case time complexity for cocktail sort? A. $O(n)$ B. $O(n \log n)$ **C.** $O(n^2)$ D. $O(n^3)$
   Explanation: In the worst case, cocktail sort would require n passes over the array which amounts to a time complexity of $O(n^2)$

2. (2 points) Suppose you have an n x n matrix, let's call it A, where every row and column is sorted in increasing order. Given a key, we need to determine if it is present in the matrix.

   (a) Suppose we know that the given key is greater than some matrix element, $A[i][j]$. Choose the correct set of regions of the given figure, where the key could lie in.



A. 4
B. $2, 4$
C. $2, 3, 4$
D. $1, 3$

The regions 2, 3, and 4 can contain values that are greater than $A[i][j]$. The key can hence lie in these regions.

(b) By expanding on the idea from above, figure out a Divide and Conquer algorithm to solve this problem - Given a key, determine if it is present in a matrix whose each row and column is sorted in increasing order. Now, find the time complexity of this algorithm.
Hint: Try writing the recurrence relation for the time complexity and then utilise Master's Theorem to get the Time Complexity.

A. $O(n \log n)$
B. $O(\log n^2)$
C. $O(n^{1.42})$
D. $O(n^{1.58})$

Find an appropriate "middle" element of the matrix (the cell at the intersection of the mid-way row and mid-way column). If middle element is the key, we are done. Otherwise, denote the four quadrants of the matrix as TL, TR, BL, BR, respectively, for the top-left, top-right, bottom-left, and bottom-right regions of the matrix relative to the middle element (see the picture above). If middle element is lesser than key then we can search TL, TR, BL. Otherwise, we search TR, BL, BR. Since there are three recursive calls to submatrices of size $(n/2 \times n/2)$, the recurrence relation for the time complexity is:

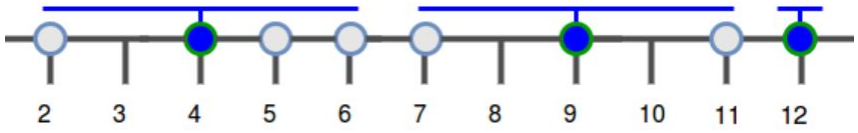
$$T(n) = 3T(n/2) + O(1)$$

Using Master's Theorem, we get the Time Complexity as $O(n^{1.58})$.

3. (4 points) Wonderland is a one-dimensional city with houses aligned at integral locations along a road. The Mayor wants to install radio transmitters on the roofs of the city's houses. Each transmitter has a fixed range meaning it can transmit a signal to all houses within that number of units distance away.

   Given a map of Wonderland and the transmission range, determine the minimum number of transmitters so that every house is within range of at least one transmitter. Each transmitter must be installed on top of an existing house.

   For example, if map of Wonderland = [7,2,4,6,5,9,12,11] and the range of transmission is said to be equal to 2, then, we would have required 3 antennae, at house no.'s 4, 9 and 12, to cover the entire city as is shown in the figure below.



(a) Let's start with solving one more example to build up the intuition of the algorithm to solve this problem. So, if the given map given = [12, 33, 2, 34, 8, 54, 85, 43, 67] and it's said that the transmission range is 19, then, how many radio transmitters will be needed?

A. 1 **B. 3** C. 0 D. 2

Explanation: 3 antennae, at house no.'s 12, 43 and 85 would have been required to cover the entire city.

(b) Consider the following approaches and choose the optimal answer, which will give us the minimum number of radio transmission towers needed to cover the city.

A. Place the transmission tower at the first house in the sorted array, traverse a range of k, then, place a new tower at this position and again, traverse a range of k, and go on, until the end of array is reached.

B. Each tower has a transmission range of $2k$, and the total range to be covered is $max(map) - min(map)$, hence, required number of towers $= \frac{max(map)-min(map)}{2k}$

C. Take the sorted array, find it's mid-element and partition the array into two at the mid-element, check the range of the partitions(P_r) with the given transmission range(t_r); if $t_r > P_r$, we place a tower at the midpoint; else, partition further.

D. Consider the first element of sorted array, say x. Let d be the last house that falls within a distance of $x + k$. This will be the position to place the transmitter. After placing the transmitter here, let y be the last house within a distance of $d + k$. Note that all houses between x and y (inclusive) are covered by the transmitter placed at d. Repeat this process with the remaining houses, continuing until all houses are covered.

Explanation: There are two issues with option A: the first is that we are only using half of any tower's full potential range, so we may use more towers than necessary, and futher, houses may not always be available at the locations we want to place them on. The second option is a potential overestimate — notice that you may have a large range with few houses, and you would not need as many towers as indicated in this option. Option C may also lead to sub-optimal solutions using more towers than necessary. Option D places the transmitter at house as far to the right from the origin house as possible to cover the whole range. The idea here is on utilising the complete range of $2k$ on both the left and the right sides, and this approach turns out to be optimal.

(c) So up till now, you might have established that a transmission tower can cover a range of $2k$, k on the left and k on the right. Look up the following piece of code and conclude if it'll work or not.

```
mapp.sort()
count, i = 0, 0
while i < len(mapp):
    count += 1
    transmissionRange = mapp[i] + 2k

    while i < len(mapp) and mapp[i] <= transmissionRange: i += 1

return count
```

A. It would work perfectly fine, as it correctly takes the maximum range a tower can cover as $2k$, and, hence, will give minimum number of towers required as output.

B. This solution will not work as it assumes that one can always place a transmitter exactly in the middle of two houses that are 2 * k apart, which is not true.

C. This will work with a change of $transmissionRange = mapp[i] + k$

Explanation: Option C actually describes the approach in Option A of part(b), and is hence incorrect. The correctness of Option B is self-explanatory here, making Option A also incorrect.

(d) Based on the discussion up till now, can you come up with the optimum time complexity of the algorithm required to solve this problem?

A. $O(\log n)$ B. $O(n)$ **C.** $O(n \log n)$ D. $O(n^2)$

Explanation: The solution involves sorting[$O(n \log n)$] and then, a single traversal[$O(n)$], so, the overall time complexity will be $O(n \log n)$. Note that you would have to use a two-pointer method to determine d and y as described in part (b) above.