DSA Assignment 4

Sricharan Vinoth Kumar, Roll No: 2024112022

Q1. Weighted Median

Brief of Question:

Given n items represented by 2 arrays V and W, each item having an integer *value*, represented by V[i] and a positive *weight*, represented by W[i], derive an algorithm to find the weighted mean of the n items.

The weighted mean is defined as x ∈ S, where S is the collection of items, such that,

$$\begin{aligned} w(S_{<x}) &\leqslant \frac{w(S)}{2} \\ w(S_{>x}) &\geqslant \frac{w(S)}{2} \end{aligned} \tag{1}$$

Where,

$$\begin{aligned} w(A) &= \text{Sum of the weights of all the elements in set A} \\ A_{<x} &= \text{Set of all elements in set A whose value is strictly less than x} \\ A_{>x} &= \text{Set of all elements in set A whose value is strictly greater than x} \end{aligned}$$

Algorithm:

Assuming we know the number of items, n,

1. First, let us define a comparision algorithm,
   For two items a and b,
   - *If value of a is greater than value of b, then* ***a is greater than b.***
   - *If value of a is equal to that of b, and weight of a is greater than that of b, then* ***a is greater than b.***
   - *Otherwise* ***b is greater than a.***
2. Now using the two given arrays V and W, create an array Q of {value, weight} pairs. The i^{th} pair of the array will represent the i^{th} item.

3. Partition Q into subarrays of size 5 and sort each subarray. Let the collection of these subarrays be $\{S_1, S_2, S_3, \ldots, S_k\}$. (k = $\lceil \frac{n}{5} \rceil$)

4. For each subarray $S_r, \forall\ r \in \{1, 2, 3, \ldots, k\}$,

   (a) Iterate through S_r and sum the weights of each element. Let this sum be w_t.

   (b) Iterate again through S_r and find the minimum possible index i such that $\forall\ k < i,\ \sum_{k=1}^{i} weight(S_r[i]) \geqslant \frac{w_t}{2}$. This index i represents the weighted mean of S_r.

   Represent this weighted mean item of each subarray S_r as m_{S_r}.

5. Repeat from step 3, now for the array M = $\{m_{S_1}, m_{S_2}, m_{S_3}, \ldots, m_{S_k}\}$, ie, recursively apply the algorithm on M, and find the weighted median of M. Let it be m_{S_h}. We can use m_{S_h} as a pivot to find the actual weighted mean.

6. In the original array, calculate the total weight of elements with value less than m_{S_h}, i.e, $w(S_{<m_{S_h}})$ and calculate the total weight of elements with value more than m_{S_h}, i.e, $w(S_{>m_{S_h}})$.

   If $w(S_{<m_{S_h}}) = w(S_{>m_{S_h}})$, m_{S_h} is the required weighted mean. Else repeat the algorithm on the partition whose total weight is more, since that partition will contain the actual weighted mean. Remove the elements on the partiton with smaller total weight from the set of probable pivot points, ie, mark them so that they are not considered in the pivot point algorithm.

Time Complexity:

Sorting each subarray = 5
Finding the weighted mean of each subarray = $\lceil \frac{n}{5} \rceil \times 5$ (There are $\lceil \frac{n}{5} \rceil$ subarrays)
Find the weighted mean of the median array = $T\left(\lceil \frac{n}{5} \rceil\right)$
Finding $w(S_{<m_{S_h}})$ and $w(S_{>m_{S_h}}) = n$
Recursing the algorithm on the parition with greater weight = $T\left(\frac{7n}{10}\right)$ worst case. (Since the pivot selection uses the Median of Median algorithm, we are guaranteed

to reduce 30% of the array in every iteration)

$$\begin{aligned}
\therefore T(n) &= 5 + 5\lceil \frac{n}{5} \rceil + T\left(\lceil \frac{n}{5} \rceil\right) + n + T\left(\frac{7n}{10}\right) \\
T(n) &= O(n) + T\left(\lceil \frac{n}{5} \rceil\right) + T\left(\frac{7n}{10}\right) \\
T(n) &= O(n) + O\left(\lceil \frac{n}{5} \rceil\right) + T\left(\lceil \frac{n}{25} \rceil\right) + O\left(\frac{7n}{10}\right) + T\left(\frac{49n}{100}\right) \\
\implies T(n) &= O(n) + \sum_{i=1}^{\infty} \left(O\left(\lceil \frac{n}{5^i} \rceil\right) + O\left(\left(\frac{7}{10}\right)^i n\right)\right)
\end{aligned}$$

The sum of O(n) complex processes, also has O(n) complexity.

$$\therefore T(n) = O(n) + O(n) + O(n)$$

$\therefore$ **The algorithm has $O(n)$ complexity**

Q2. Finding the Median of Five Distinct Numbers Using a Decision Tree

Brief of Question:

Represent an algorithm to find the Median among an array of 5 unique elements, using a decision tree, where each node represents a comparision between 2 elements.

Algorithm:

Let the array be A and let A[i] be the object at the i-th index. Let the indexing be 1-indexed. Let $\langle x, y \rangle$ represent a comparision between x and y. Also, let each comparision be indexed with (j).
Since there are 5 elements, the median must be exactly greater than and lesser than 2 elements.

1. Find the minimum of each pair (A[1], A[2]) and (A[3], A[4]). (2 comparisions)
2. Find the minimum of those minimums. Remove it from contention since it is smaller than 3 elements. (max(A[1], A[2]), max(A[3], A[4]) and the other minimum). (1 comparision)

3. Pair the other minimum with A[5]. Find the minimum of that pair. (1 comparision)
4. Compare the new minimum with the other minimum. Remove the smaller element since it it smaller than 3 elements (its maximum, the larger minimum and the maximum of the larger minimum) (1 comparision)
5. Compare the lone element with the minimum of the remaining pair (*the larger minimum and the maximum of the larger minimum*, in the previous step). The smaller value is the median. (It is smaller than the maximum of the pair and the larger value, and is greater than the 2 previously eliminated values) (1 comparision)

Decision Tree:

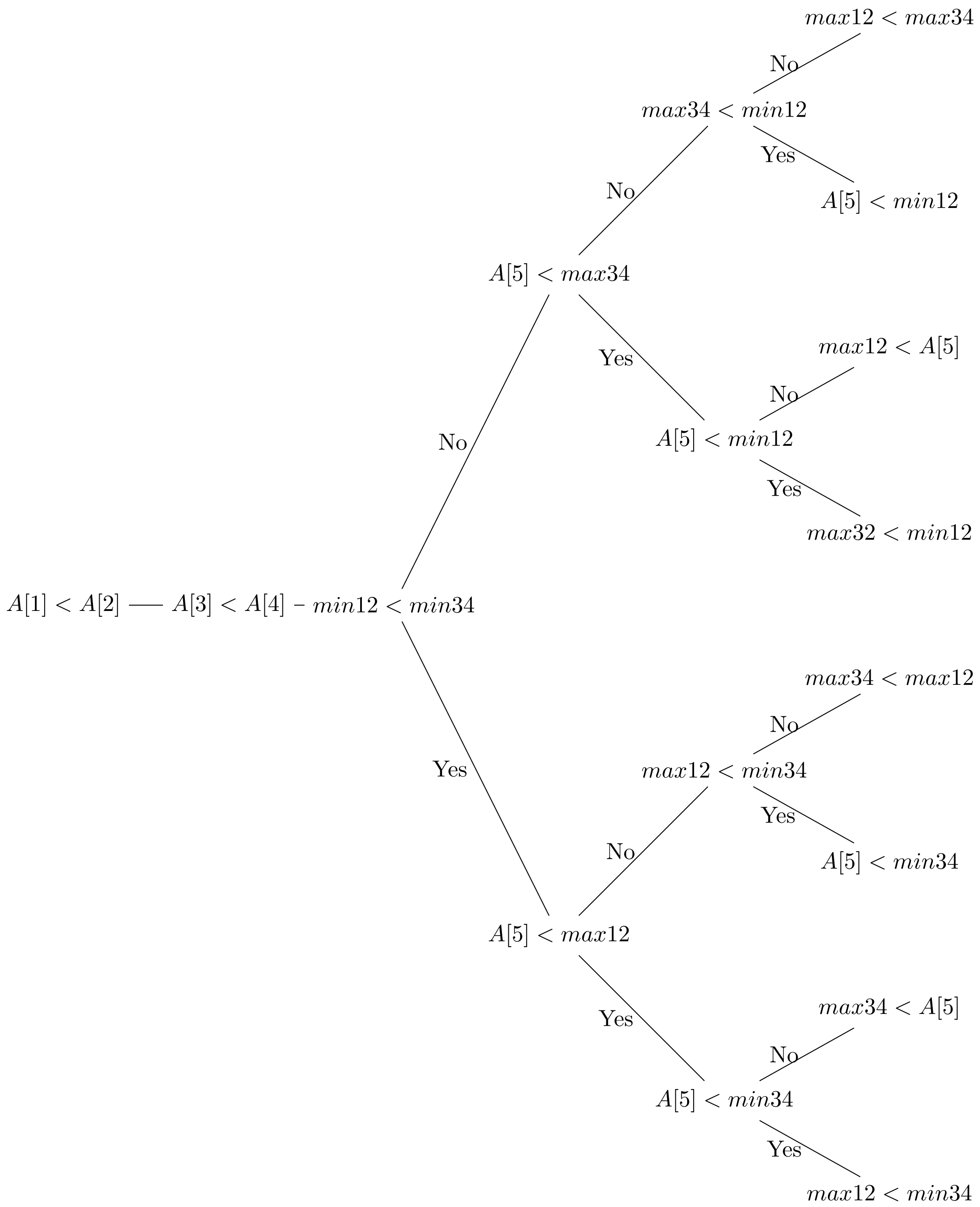
Since it is impractical to represent all the possible combinations as it is in a decision tree, since the total no of nodes will be,

$$\begin{aligned}\text{No of nodes} &= \sum_{i=1}^{6} 2^i \text{ ,(Sum of number of nodes in each level)}\\ &= 2^1 + 2^2 + 2^3 + 2^4 + 2^5 + 2^6\\ &= 2 + 4 + 8 + 16 + 32 + 64\\ &= 126\end{aligned}$$

We will use the following variables to compress the Decision tree.

$$\begin{aligned}max12 &= max\{A[1], A[2]\}\\ min12 &= min\{A[1], A[2]\}\\ max34 &= max\{A[3], A[4]\}\\ min34 &= min\{A[3], A[4]\}\end{aligned}$$

These variables will be defined after the first step of the algorithm.


$A[1] < A[2]$
$A[3] < A[4]$
$min12 < min34$
No
Yes
$A[5] < max34$
$A[5] < max12$
$max34 < min12$
$A[5] < min12$
$max12 < min34$
$A[5] < min34$
$max12 < max34$
$A[5] < min12$
$max12 < A[5]$
$max32 < min12$
$max34 < max12$
$A[5] < min34$
$max34 < A[5]$
$max12 < min34$

Q3. Identifying Top Programmers in a Hackathon

Brief of Question:

Given 100 programmers, such that only 10 programmers can be compared at a time, in a session, with no ties, and the programmers's skills are transitive,

(a) Devise a strategy to find the best programmer using 11 sessions at most.

(b) Prove that it is impossible to find the best programmer using only 10 sessions.

(c) Identity the top 4 programmers using 12 sessions at most.

Answers:

(a) **Algorithm:** Let the programmers be represented as an 100-length array of integers $M = \{P_1, P_2, P_3, \dots, P_{100}\}$, where each integer P_i represents the skill level of the i-th programmer.

1. Divide the array into 10-length paritions $\{\{P_1, P_2, \dots, P_{10}\}, \{P_{11}, P_{12}, \dots, P_{20}\}, \dots, \{P_{91}, P_{92}, \dots, P_{100}\}\}$, ie, $M = \{M_1, M_2, M_3, \dots, M_{10}\}$ where each $M_i = \{P_{10i-9}, \dots, P_{10i}\}$, for $i \in \{1, 2, 3, 4, \dots, 10\}$
2. Run the Session on each parition M_i and get the best player. Let the best player of each parition be P_{M_i}.
3. Combine the best players of each partition into an array $N = \{P_{M_1}, P_{M_2}, \dots, P_{M_{10}}\}$. Since there are 10 paritions, the size of N will be 10. Run a session on N. The best player in N, P_N will be the best programmer in the Hackathon.

Reasoning: Since each P_{M_i} is the best of its partition, we get

$$P_{M_i} \geqslant p \;\forall\; p \in M_i$$

Therefore, for the best in N,

$$\begin{aligned} & P_N \geqslant p \;\forall\; p \in N \\ \implies & P_N \geqslant P_{M_i} \;\forall\; i \in \{1, 2, 3, \dots, 10\} \\ \implies & P_N \geqslant p \;\forall\; p \in M_i \;\forall\; i \in \{1, 2, 3, \dots, 10\} \\ \implies & P_N \geqslant p_j \;\forall\; j \in \{1, 2, 3, \dots, 100\} \end{aligned}$$

$\therefore P_N$ is the greatest element in M. $\implies P_N$ is the greatest programmer's skill level.

(b) To Prove that we need a minimum of 11 sessions to determine the best programmer.

- Assume we have a comparision algorithm than processed the 100 programmers. To make any sort of ordering among these programmers, we need a direct or indirect relation between any 2 programmers. That is,

$$\forall\ P_i, P_j \text{ , either } P_i < P_j \text{ or } P_i > P_j \text{ must be definitively known} \qquad (2)$$

- We can represent this existence of information using a graph. In the graph,

$$\exists \text{ A path between } P_i \text{ and } P_j \text{ iff } \exists \text{ a known relation between } P_i \text{ and } P_j \qquad (3)$$

  Each edge in this graph will represent a comparision between 2 programmers.
- Therefore, from Result (2), for a valid algorithm to find the greatest programmer, the graph of that algorithm must be **fully connected**.
- Since the graph must be fully connected, $\exists$ 2 sessions such that they have a programmer processed in common, since this common programmer is needed to connect the relations obtained in the 2 sessions. There can be (and will be) multiple such pairs. However for this proof, the existence of one such pair is enough.
- If we perform 10 sessions, since we know that atleast 2 sessions must have a programmer in common, the no. of programmers processed/connected will be $100-$(common programmers) which is strictly lesser than 100.
- $\therefore$ The largest connected component of the graph will have < 100 nodes, which implies that **we do not have a fully connected graph**.

$\therefore$ We will need more than 10 sessions to form a fully connected graph.
$\therefore$ A proper algorithm will need more than 10 sessions to find the best programmer.

(c) Amongst the 100 programmers, the top 4 programmers will have the following properties.

1. The 1st place programmer will never lose.
2. The 2nd place programmer can only lose to the 1st place programmer.
3. The 3rd place programmer can lose to the 1st or 2nd programmer only.
4. The 4th place programmer can lose to the above 3 programmers only.

Let us represent these programmers by 1,2,3,4.
If we use the algorithm defined in (a), using the above properties, we can infer the following results:

1. 1 is the best programmer in N. Let 1 belong in partition M_a. (The partitions are defined in (a))
2. 2 must have lost to 1 either in N or in M_a. Therefore, he must be the second best in either N or M_a or the best in any other partition (let it be M_b).
3. 3 must have lost to either 1 or 2 or both. Therefore the possiblilities are:
    i. If 3 lost to 1 and 2 in the partition round, ie, they were all in the same parition. Then 3 must be 3rd in M_a. If 3 lost to 1 and 2 in the final round, he must be 3rd in N.
    ii. If 3 lost to only 1 in the parition round, he must be 2nd in M_a. If 3 lost only to 1 in the final round, he must be 2nd in N.
    iii. If 3 lost to only 2 in the parition round, he must be 2nd in M_b. 3 cannot lose ONLY to 2 in the final round, due to the presence of 1.
    iv. If 3 did not lose in the partition round, he must be the best in his partition. Let that partition be M_c
4. 4 can lose only to the above 3 programmers so the possiblilities are:
    i. If 4 reached the final round, he must be either 4th, 3rd or 2nd in N (depending on whether 2 and 3 reached the final round or not). He must be the best in his partition (let it be M_d).
    ii. If 4 lost only to 3 or 2 or 1 in the partition round, and not the others, then he must be 2nd in M_a, M_b or M_c.
    iii. If 4 lost to both 1 and 2 only, then he must be 3rd in M_a.
    iv. If 4 lost to both 2 and 3 only, then he must be 3rd in M_b.
    v. If 4 lost to all 3,2,1, then he must be 4th in M_a

Using the above results, the most likely candidates for top 4 will be:

- 1st to 4th in M_a
- 1st to 3rd in M_b
- 1st and 2nd in M_c
- 1st in M_d

Which amounts to 10 candidates.
Therefore if we run the first 11 sessions in a similar way to the algorithm in (a) and in the 12th session, we compare the above listed programmers, we will get the true top 4 programmers in a total of 12 sessions.
This will work since the 12th session will collapse all these different possiblilities, since we will know the skill order, which will give us the top 4, amongst these 10 shortlisted candidates, and from the results above, we know that the non-shortlisted candidates cannot be a part of the top 4.