**Answer (a) :**

// this function will be called as **MOM7QS**(A)

1. **def MOM7QS(int arr):**

2. if(n>1) …(p)

3. A[p]=MomPivot(A[1...n]) **…3** …(q)

4. r=Partition(A[1...n],p) …(r)

5. MOM7QS(A[1...(r-1)]) …(s)

6. MOM7QS(A[(r+1)...n]) …(t)

7. return

8. **def MomPivot(A[1...n]):**

9. if(n<=49) …(a)

10. Use BruteForce to find median & return mom …(a)

11. else …(a)

12. m=**⌈**n/7**⌉** …(a)

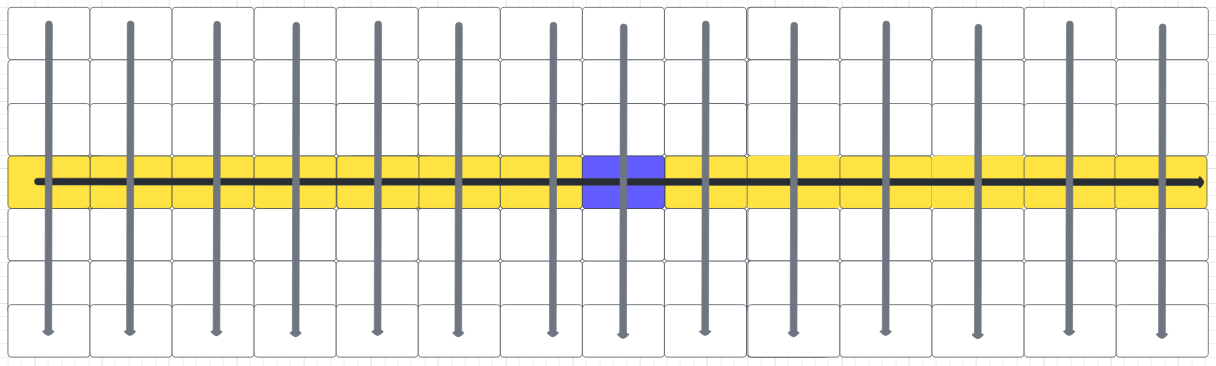
13. for(i=1 to m) …(b)

14. M[i]=MedianOfSeven(A[7i-6,...7i]) //use BruteForce …(b)

15. mom=MomPivot(M[1...m]) …(c)

**Answer (b) :**

As ***Prof.*** *Jeff* pointed out in his analysis of the quadratic worst-case behavior of QuickSelect, that if we could *somehow* magically **choose a good pivot in linear time,** the time taken in line **3** would exponentially decay in the recursion tree analysis. The **MomPivot** method divides the input array into **⌈n/7⌉** blocks, each block containing exactly 7 elements except the last block, as n can be chosen to not be a multiple of 7. Thus, in many cases, there can be an extra deficient block at the end.



We compute the median of these blocks by brute force & gather them into a new array **M[1…⌈n/7⌉]**, and then recursively compute the median of this newly formed array of medians. Let the time complexity of the **MomPivot** method be **T(n).**

**T(n) = Ta + Tb + Tc**

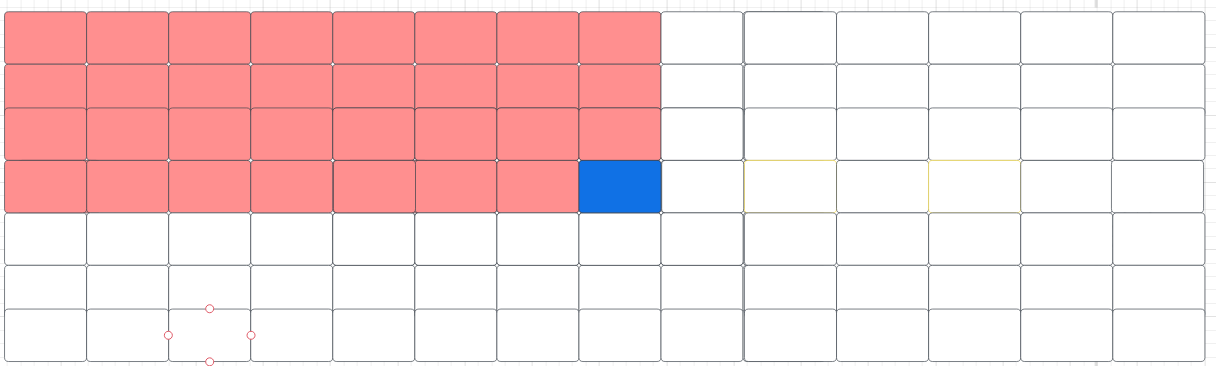
**Ta** is a constant time operation, so complexity is O(1).

**Tb** denotes the entire array **A[1...n]** traversal, so it takes O(n) time**.**

**Tc:** Here, the problem size is recursively reduced to **⌈**n/7**⌉,** so it will take **T(⌈**n/7**⌉)** time.

**Hence**,  **=>**

Finally, we use the median of the block medians (called “**mom**” in the pseudocode above) as the pivot for the **MOM7QS** method to sort the array. The **Mom** is larger



Let the time complexity of the **MomPivot** method be **G(n).**

**G(n) = Gp + Gq + Gr + Gs + Gt**

**Gp:** is a constant time operation, so complexity is **O(1).**

**Gq:** the **MomPivot** method takes **O(n)** time complexity ***[*** *solved later in answer* ***(c) ]***

**Gr:** the **Partition** method also takes **O(n)** time complexity, because it’s a simple for-loop with constant work per iteration.

As illustrated above, each 7-sized column is sorted top down & then we sort the columns by their middle element. In this arrangement, the median-of-medians is the element closest to the center of the grid. Mom is larger than **⌊⌈**n/7**⌉**/2**⌋ - 1 =** n/14 block medians, and each block median is larger than 3 other elements in its block. Thus, mom is bigger than at least **⌊**4n/14**⌋** elements in the input array. *(marked with pink color in the above diagram)*

If the element we’re looking for is larger than mom, our algorithm will throw away everything smaller than mom, including those **⌊**4n/14**⌋** elements, before recursing. Thus, the input to the recursive subproblem contains at most **⌈**10n/14**⌉** elements.

A symmetric argument implies that if our target element is smaller than mom, we discard at least **⌊**4n/14**⌋** elements larger than mom, so the input to our recursive subproblem has at most **⌈**10n/14**⌉** elements.

Thus **(Gs) + (Gt) = G(⌈**10n/14**⌉) + G(⌊**4n/14**⌋)** *[The exact order of* ***Gs*** *&* ***Gt****is symmetric.]*

Thus, **G(n) = O(1) + O(n) +O(n) + G(⌈**10n/14**⌉) + G(⌊**4n/14**⌋)**

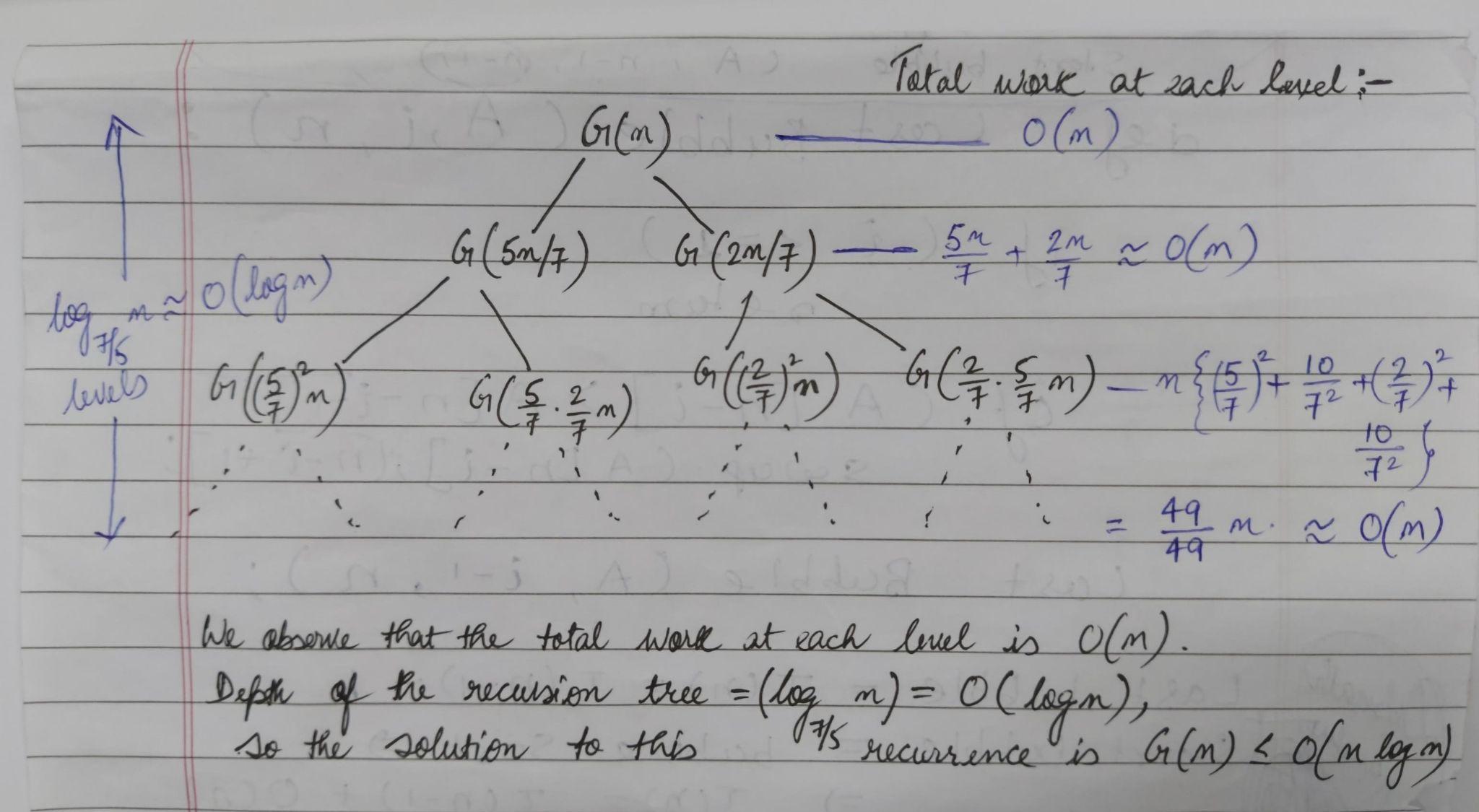
**Or,**

**Answer (c) :**

The solution of the recurrence is **O(n)**

[By Master’s Theorem]

The recurrence can be solved through the following recurrence tree:



Hence, **G(n)** = **O(n log(n))**