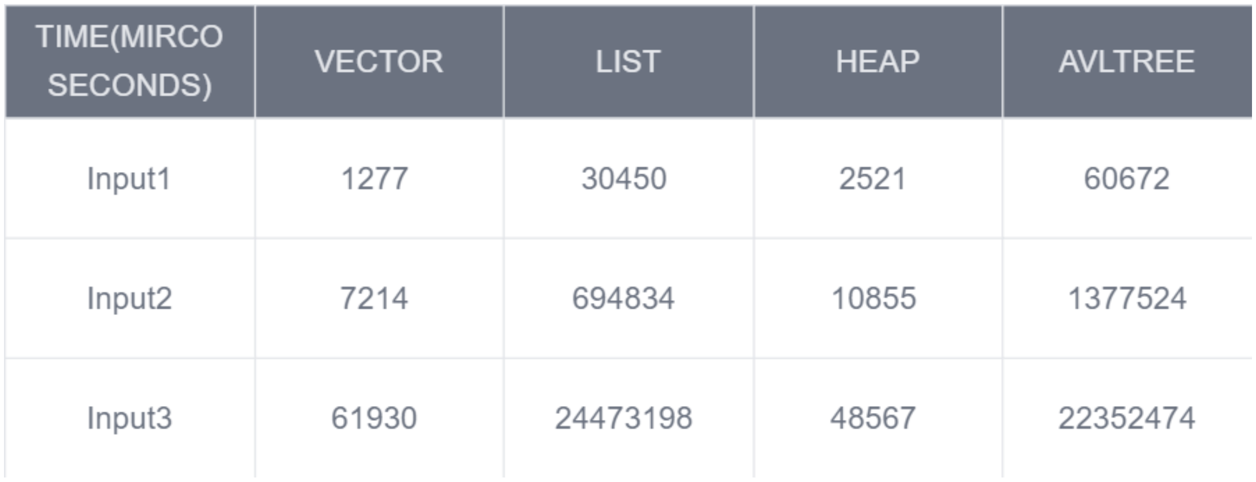
1.



TIME(MIRCO SECONDS)	VECTOR	LIST	HEAP	AVLTREE
Input1	1277	30450	2521	60672
Input2	7214	694834	10855	1377524
Input3	61930	24473198	48567	22352474

2.

VectorMedian and ListMedian: The creation of empty vector or list to store numbers from instruction is O (1). The for loop to iterate every instruction is O (N), and N is the number of instructions. In the for loop there is two cases, insert and pop. For insert, find the position of vector is O (log m) and m is the size of the contain numbers and insert an element to vector or list other than end position is O(m), insert to end of a vector or list is O (1). Combine together is O (m+1) = O (m). For pop median, insert median to end of output vector or list is O (1) and remove for vector or list contain the numbers is O (m).

For any cases m is less than n, for worse case there is only insert instruction m can be considered as N. So the worst case of VectorMedian and ListMedian is O(N*((logN+N)+(1+N))=O (N^2). For tight bound of VectorMedian and ListMedian is only pop median instruction, so m can be consider as 1 then tight bound of VectorMedian and ListMedian is Ω (N*((1+1)+(1+1)))= Ω (N)

HeapMedian: The creation of empty min heap and max heap is O(1). Use for loop to iterate every instruction is O(n) and n is the number of instructions. In the for loop there is two cases, insert and pop. For insert, either insert to maxheap or minheap is O(log m) and m is the size of the max heap or minheap and check and balance the maxheap always store the median require a O (log m). for pop median, the insert to back of a vector and delete of a heap is both O(1) and check and balance the maxheap always store the median require a O (log m).

Similar to VectorMedian and ListMedian, the m is less than n, so the for worst case m consider as n, so worst case of HeapMedian is O(N*((log N+log N)+(1+ 1 + log N))) =

O(N log N). And for tight bound m can be considered as 1, so Ω(N*((1+1)+(1+1+1))= Ω(N)

TreeMedian: The creation of two empty avltree is O(1). Use for loop to iterate every instruction is O(n) and n is the number of instructions. In the for loop there is two cases, insert and pop. For insert, either insert to either avltree is O(log m) and m is the size of the either avltree and check and balance of two avltree store right value require a getsize of avltree is O (m) and insert, remove, find max and find is O(log m). so it is O(m+log m)=O(m). The pop median require findMax and deletion of avltree require O(log m) and insert an element to back of vector is O(1), then check the avltree is O(m).

M is less than N. And for worst case the m can be considered as N, so the worst case of treeMedian is O(N*((log N + logN + N)+(logN + 1 + logN + N)))=O(N^2). For tight bound the m can be considered as 1. Ω(N*((1+1+1)+(1+1+1))= Ω(N).

3.

vectorMedian and listMedian: As predicted, these methods exhibit quadratic time complexity (O(n^2)) in a ratio of (1:5.6:48 for vector and 1: 22.8:803.7 for list) due to the linear operations (insert, erase) on vectors and lists.

heapMedian: The observed time complexity matches the expected O(nlog n) in a ratio of (1: 4.3: 19.3) due to efficient heap operations with logarithmic time (1: 4.3: 19.3) complexity for insertion, deletion and resizing.

treeMedian: The AVL tree-based approach in treeMedian demonstrates the expected O(N^2) time complexity in a ratio of (1: 22.7: 368.4), the treeMedian use both linear operation (for loop and getSize) and logarithmic time operator of insertion and deletion.

4.

For addition observation, we can find even the logic of vectorMedian and listMedian is similar and they both have O(N^2), but listMedian take much more time than vectorMedian, and it is interesting to see that for input1 and input 2 the HeapMedian is take time more than VectorMedian, but input3 the HeapMedian take less time than vetorMedian, which is prove O(nlogn) is growing slower than O(n^2).