In the following assume $m = n$.
1. Use Boyer-Moore voting algorithm, maintain the information (number, count) by segment tree. $O(n \log n)$.
2. The information we maintain forms an (associative) semigroup, so it's mergeable. We need to query range sum on a static array, which needs $O(n\alpha(n))$ by Yao [5]. [1] also gives a $\Theta(n\lambda(k, n))$ ($= O(n\alpha(n))$ for our purpose) time and space algorithm, where $\lambda(k, \cdot)$ is the inverse of a certain function at the $\lfloor \frac{k}{2} \rfloor$-th level of the primitive recursive hierarchy. Then check whether the number we find is valid, by computing the number of occurrence of it in the query interval, using persistent array (or vEB tree) in $O(\log \log n)$. We can also solve this in $O(\frac{\log \log n}{\log \log \log n})$ per query by reducing to the static predecessor problem `https://en.wikipedia.org/wiki/Predecessor_problem` [2] (but with $O(n^4)$ preprocessing time; in our case the universe $N = n$). In conclusion we get $O(n \log \log n)$.
We can also get $O(n)$ preprocessing and $O(1)$ per query, see my article here: `https://zhuanlan.zhihu.com/p/79423299`.
3. $O(n \log n)$ preprocessing, $O(1)$ per query [3, 4].
4. If we randomly select $O(\log n)$ elements in the array and check the count in $O(\log \log n)$, we can get w.h.p. $O(n \log n \log \log n)$.
5. We further improve Alg. 4. Suppose the query range has length m. If a random element x has frequency $\leq \frac{m}{4}$ in the range, then we can detect this in $O(1)$ time, by checking that both the previous and the subsequent $\frac{m}{4}$-th occurrence of x is outside the range. In this case, x cannot be the strict majority. For elements with frequency $\geq \frac{m}{4}$ in the range, we need $O(\log n)$ time to compute its frequency (or $O(\log \log n)$ by vEB). We can terminate if we either find an element with frequency $\geq \frac{m}{2}$, or we find two distinct elements with frequency between $\frac{m}{4}$ and $\frac{m}{2}$ (in this case, there's no strict majority). We need to take $O(\log n)$ samples to ensure w.h.p. success, so the running time per query is $O(\log n) \cdot O(1) + O(1) \cdot O(\log \log n) = O(\log n)$. $O(n \log n)$ w.h.p. see my article here: `https://zhuanlan.zhihu.com/p/344219746`.
6. Use persistent segment tree. $O(n \log n)$.
7. Compute the majority for each bit. $O(n \log U + n \log n)$.

References

[1] Noga Alon and Baruch Schieber. *Optimal preprocessing for answering on-line product queries.* Citeseer, 1987.

[2] Paul Beame and Faith E Fich. Optimal bounds for the predecessor problem. In *STOC*, volume 99, pages 295–304. Citeseer, 1999.

[3] Stephane Durocher, Meng He, J Ian Munro, Patrick K Nicholson, and Matthew Skala. Range majority in constant time and linear space. In *International Colloquium on Automata, Languages, and Programming*, pages 244–255. Springer, 2011.

[4] Gonzalo Navarro and Sharma V Thankachan. Optimal encodings for range majority queries. *Algorithmica*, 74(3):1082–1098, 2016.

[5] Andrew C Yao. Space-time tradeoff for answering range queries. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 128–136. ACM, 1982.

Online Majority Element In Subarray

Submission Detail

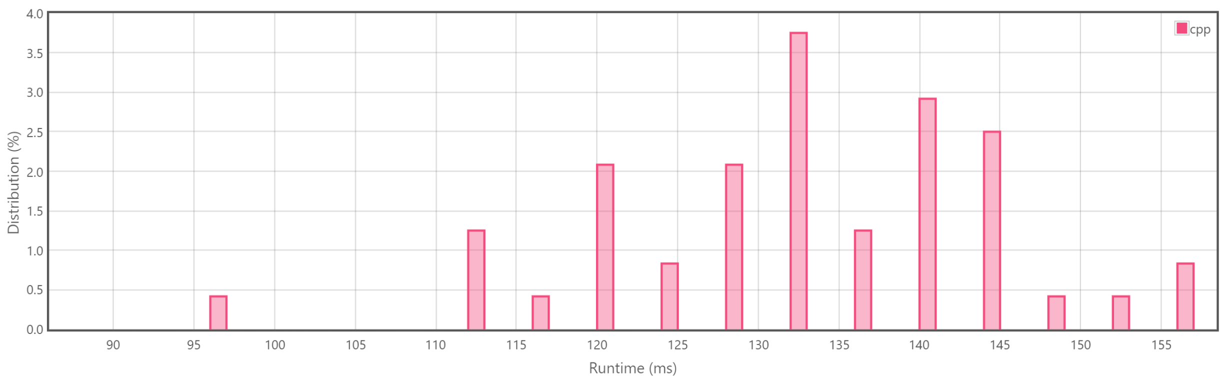
27 / 27 test cases passed.

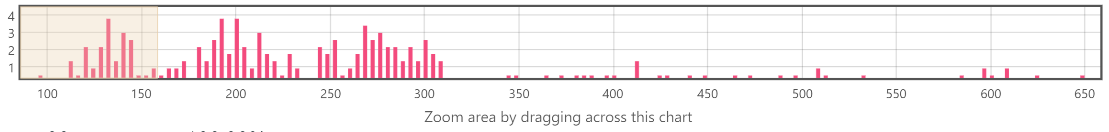
Runtime: **80 ms**
Memory Usage: **61.3 MB**

Status: Accepted

Submitted: **2 minutes ago**

Accepted Solutions Runtime Distribution





Runtime: 80 ms, faster than 100.00% of C++ online submissions for Online Majority Element In Subarray.

Memory Usage: 61.3 MB, less than 97.92% of C++ online submissions for Online Majority Element In Subarray.