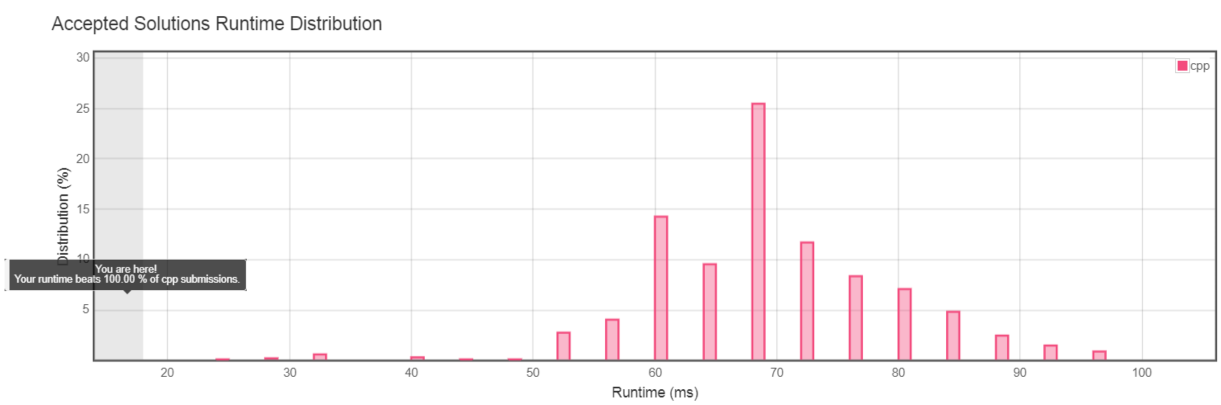
1. counting for each bit. $O(nw)$.
2. for each bit, count the number of 0s and 1s in the array, which needs $O(\log n)$ bits of space to store if the array has length $n$. assume word operation is $O(1)$ after preprocessing (the word operations we want: (1) add two $w_1$-bit numbers, (2) padding, i.e. pad $w_1$ bits with leading 0 to get $w_2$ bits), the counting for each bit can be performed in parallel by word operations.
Use divide and conquer to perform counting, let $f(n)$ denote the running time to get the count for each bit when the array has length $n$. merge the results for two arrays with length $\frac{n}{2}$ needs $O(w \cdot \log n)$ bits, i.e. $O(\log n)$ word operations. Intuitively if the length of the array is small, we can use fewer bits.
$f(n) = 2f(n/2) + O(\log n)$, i.e. $f(n) = \sum_{i=0}^{\log n} 2^{\log n - i} \cdot i = O(n)$.
total time $O(n)$.

actually if we represent the counters by $O(\log n)$ words, where the $j$-th bit of the $i$-th word represent the $i$-th bit of the $j$-th counter, we can construct the word operations we want by $O(1)$ basic word operations (implement an adder which is parallel in bits). The following C++ implementation uses the logarithmic method [1] to reduce the space complexity to $O(\log^2 n)$.

3. If each integer only has $k$ bits, there are only $O(2^k)$ distinct integers, we can use buckets to count each distinct value. Then we can use a slower algorithm on $n' \leq 2^k$ values (in this special case, the values are $0, \cdots, 2^k - 1$, there's an easier $O(2^k)$ algorithm by prefix sum on buckets, because to count on the $w$-th bit, we only need to count on $\frac{2^k}{2^w}$ intervals of the form $* \cdots * 1 \underbrace{* \cdots *}_{w-1}$, the running time is $\sum_{w=0}^{k-1} \frac{2^k}{2^w} = O(2^k)$).
set $k = \Theta(\log n)$ s.t. $2^k \leq n$, running time $O(\frac{w}{k} \cdot (n + 2^k)) = O(\frac{nw}{\log n})$.




# References

[1] Jon Louis Bentley and James B Saxe. Decomposable searching problems i. static-to-dynamic transformation. *Journal of Algorithms*, 1(4):301–358, 1980.