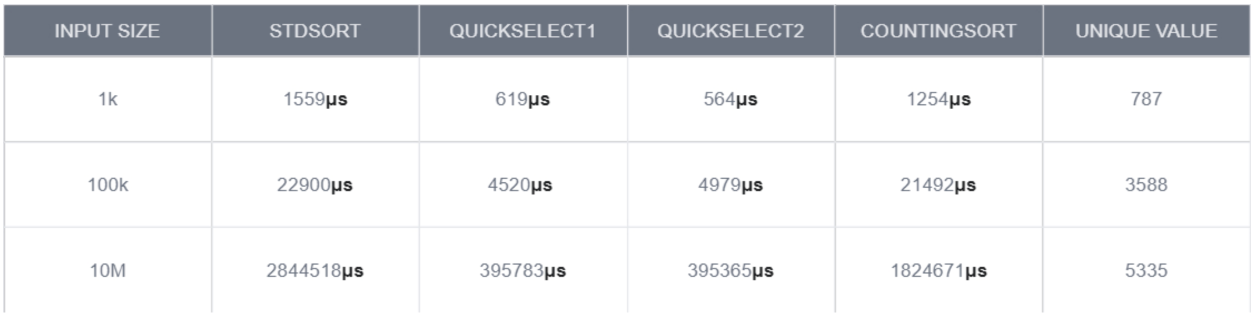
REPORT

1)



INPUT SIZE	STDSORT	QUICKSELECT1	QUICKSELECT2	COUNTINGSORT	UNIQUE VALUE
1k	1559**µs**	619**µs**	564**µs**	1254**µs**	787
100k	22900**µs**	4520**µs**	4979**µs**	21492**µs**	3588
10M	2844518**µs**	395783**µs**	395365**µs**	1824671**µs**	5335

2)

StdSort: Use sort method in STL is O(nlogn) and access each percent tail or min and max in a vector container is O (1), so the time complexity of StdSort is O(nlogn)+O(1)*5 = O(nlogn)

QuickSelect1: choose the middle element as pivot is an O (1), then find first smallest and largest element until all smaller element is placed on left side of pivot and all larger element is place on right side of pivot. And this process has an O(n). After that restore the pivot and it is an O (1). Finally, compare pivot and kth smallest we want to find, to decide either quick select is done or it should be another quick select in left side of pivot or right side of pivot, in worst case we choose a smallest or largest element as pivot and it will lead to n time recursive which is O(n).

To find min and max after the quick select, we only need search on region below 25% and above 75% and store the smallest or largest element, which both is O(n/4) = O(n)

As conclusion the average case of Quickselect1 is O(n) and the worst case of Quickselect1 is 3(O(n))(O(1) + O(n) + O(1)) + 2O(n) = O(n^2) +O(n) = O(n^2) .

QuickSelect2: choose the middle element as pivot is an O (1), then find first smallest and largest element until all smaller element is placed on left side of pivot and all larger element is place on right side of pivot. And this process has an O(n). After that restore the pivot and it is an O (1). The difference is k is not an element, but is it is a vector, so in the comparison, if element in k is equal to pivot + 1, then push to result vector and erase from vector which is O(k) and all other elements in the k is either push to left side vector or right side vector which is O(1). Finally, if check if left side vector or right side vector is empty if is not, call quickselect2 on left side or right side vector is O(n).

As conclusion the average case of Quickselect2 is O(n) and the worst case of Quickselect2 is (O(n)) (O (1) + O(n) + O (1) + O (k)) = O(n^2).

CountingSort: create an unorder map and insert every element from data will have O(n) and then insert all key of unorder map to a vector is dependent on number of unique elements as k, so it is a O(k). Then use stl sort the vector will lead a O(klogk). Finally, assume vector is a counting array and use start and end to record index of duplicate value, so the it is O(k).

As conclusion, time complexity is O(n) + O(k) + O(klogk) + O(k) = O(n + klogk) which k is number of unique numbers.

3)

For StdSort have quicksort as base, if less unique value and more copy in the data, it will lead to the worst case of quicksort which is O(n^2) and to avoid this happening stdsort may switch to heap sort have O(nlogn)

For QuickSelect1 and QuickSelect2 they have similar idea and less unique value and more copy in the data will cause worse case to happen which is O(n^2) which is less efficient.

For CountingSort if less unique value and more copy in the data means more efficient, since O(n + klogk) the k is number of unique value and if all value in data is copies, then it will be O(n) which is more efficient.

4) I don’t find big difference between QuickSelect1 and QuickSelect2 and their performance is best. But it is surprise the CountingSort is slower than I thought it is more close to StdSort but QuickSelect.