1. **reverse(a, a+n);**  // Reverse an array/vector//
2. **remove(v.begin(), v.end(), 'key');** //Remove an element from an array by key//
3. **v.erase(it);** it = position; // Vector erase //
4. **accumulate(v.begin(), v.end(), 0);** // Sum of array//
5. **\*max\_element(v.begin(), v.end());** // Max element// Returns the integer having maximum value
6. **find(v.begin(), v.end(), 'key');** // Find : STL// // Returns a pointer. Index can be fetched as \*it;
7. **binary\_search(v.begin(), v.end(), 'key'**) // Binary\_Search // // Returns true/false if the element is present
8. **count(v.begin(), v.end(), 'key')** // Count // // Returns the integer which is the count of the 'key' present in a container
9. **count(a, a+n, 'key')** // Count in array// // returns a boolean value true/false if the element is present or not
10. **distance(v.begin(), v.end())** // Distance between two pointers // returns integer which is the distance or total number of elements lying between v.begin() and v.end();
11. **replace(v.begin(), v.end(), 'old', 'new');** // Replace // // replaces all occurences of old with new in the array
12. **is\_sorted(a,a+n);** // Check if the array is sorted//
13. **is\_sorted\_until(v.begin(), v.end());** // Check for the range upto which the array is sorted // returns a pointer. to get the index use distance(it, v.begin());
14. **upper\_bound(a, a+n, 'key')-a** // Upper Bound // // returns the index of first element in the array greater than 'key'
15. **lower\_bound(a, a+n, 'key')- a** // Lower Bound // // returns the index of first element in the array less than 'key'
16. **merge(a, a+N, b, b+N, c)** // Merge two sorted ranges/arrays // // c is the array which contains all the sorted elements
17. **includes(v1.begin(), v1.end(), v2.begin(), v2.end());** // Includes a subset// v2 are the indices of subset
18. **set\_difference(a, a+N, b, b+N, arr);** // Difference of two sets// 'a' and 'b' are two sets and 'arr' is the array having set difference;
19. **set\_intersection(v1.begin(), v1.end(), v2.begin(), v2.end(), v3.begin());**
20. **std::set\_union(v1.begin(), v1.end(), v2.begin(), v2.end(); v3.begin());**
21. **set\_symmetric\_difference( v1.begin(), v1.end(), v2.begin(), v2.end(), v3.begin());**
22. **equal (vector\_1.begin(), vector\_1.end(), a)** // Equal check if two arrays/strings/containers are equal// a[] is an array
23. **is\_permutation ( A, A+4, B ) )** // Permutation// https://www.geeksforgeeks.org/stdis\_permutation-c-stl/ where A and B are arrays and we are checking if B is permutation of A or not
24. // Next Permutation// **next\_permutation(s.begin(), s.end());** works for strings and arrays
25. // Previous Permutation**// prev\_permutation(s.begin(), s.end())**