**Divide and conquer algorithm**

* Divide and conquer procedure:

1. breaking the problem into non-overlapping subproblems of the same type.
2. recursively solving those subproblems.
3. combining the results

* For running time deduction, remember it is used by propotional series to get the result for all the levels from 0 to log2^n
* Master Theorem:

If (for constants a > 0, b > 1, d >= 0), then:

* Selection sort running time O(n^2)
* Merge sort is a kind of divide and conquer algorithm,

T(n) <= 2\*T(n/2) + O(n), (smaller or equal is because each recursion n will reduce,

Running time O(nlog(n)).

* A comparison based sorting algorithm sorts objects by comparing pairs of them.

Examples: selection sort, merge sort

Lemma: Any comparison based sorting algorithm perform Ω(n\*logn) comparisons in the worst case to sort n objects

* Non-comparison sort, example: counting sort. All integers are form 1 to M, the running time O(n + M).
* Quick sort, running time on average: O(n\*logn), (notice here it is on average not at least)
* Unbalanced split, if T(n) = T(1/10 \* n) + T(9/10 \* n) + n, running time still O(n\*logn)
* If all elements are equal, then running time O(n^2), so we adopt quick sort 3 which split elements into 3 groups.
* Quick sort quiz: **What is the amount of additional memory that regular Quick Sort uses (besides the array being sorted) in the worst case?**

In the worst case, the array is always divided into a part of size 1 and a part with all the other elements, and the recursion depth in this case will be O(n). Recursion needs O(1) additional memory for each call, so in the worst case Quick Sort will use O(n) additional memory. However, by using tail recursion elimination we can make Quick Sort use no more than O(logn) additional memory. See the [lecture](https://www.coursera.org/teach/algorithmic-toolbox/author/outline/lecture/D5xqU/video-subtitles) with the final remarks about Quick Sort.

Assignments:

1. For binary search,

(1) use mid = left + (right – left) / 2, here let right be vector.size() – 1, Termination condition if (right < left) or while (left <= right)

1. Divide and conquer method:
2. Confirm left majority and right majority
3. If either of them is not -1, count the number in merged array to check if over n /2.
4. Recurrence

HashMap method：

* Create unordered\_map to store all values.
* Count each keys mapped value.

Note: Unordered maps implement the direct access operator (operator[]) which allows for direct access of the mapped value using its key value as argument.

And that’s why it is easy for us to count the number of each element in the vector.

<http://www.cplusplus.com/reference/unordered_map/unordered_map/>

1. 注意，如下方法是错误的，他在此种组合就失败了：66127863，

for (int i = l + 1; i <= r; i++)

{

if (a[i] < x)

{

m1++;

m2++;

swap(a[m1], a[i]);

}

else if (a[i] == x)

{

m2++;

swap(a[m2], a[i]);

}

}

swap(a[l], a[m1]);

正确采取Dijkstra 3-way partitioning,

<https://algs4.cs.princeton.edu/lectures/23DemoPartitioning.pdf>

1. In merge process, let i is used for indexing left sub-array and j for right sub-array. At any step in merge(), if a[i] is greater than a[j], then there are (mid – i) inversions. because left and right subarrays are sorted, so all the remaining elements in left-subarray (a[i+1], a[i+2] … a[mid]) will be greater than a[j]

<https://www.geeksforgeeks.org/counting-inversions/>

1. See professors’ hints,

<https://www.coursera.org/learn/algorithmic-toolbox/discussions/all/threads/QJ1jK9wNEeWdPBL2iFTrAw/replies/Ihiw4txhEeWK5g7mfcS2Xw/comments/oyAMaeIiEeWyqwpvChh66Q>

将所有元素按pair(pos, char)进行排序，起始点（start, l）,终点（end, r）, 点（pos, p）

1. O(n\*logn^2), <https://www.geeksforgeeks.org/closest-pair-of-points-using-divide-and-conquer-algorithm/>

O(n\*logn), <https://www.geeksforgeeks.org/closest-pair-of-points-onlogn-implementation/>