Problem7.1

a)

Counting\_Sort.cpp

b)

Bucket\_Sort.cpp

c)

Count(A, n, k, a, b){ // A has n numbers from 0 to k, find how many [a, b]

CountArray[k+1]

For i=0 to k

CountArray[i]=0

For j=0 to n

CountArray [ A[j] ] +1

Sum = 0

For i=a to b

Sum +=CountArray[i]

Return Sum

}

d)

Word\_Sort.cpp

e)

For Bucket Sort, the worst case is when all elements in the given sequence are being categorized into a same bucket. In which case the time complexity of Bucket Sort will be **same as time complexity of the inner algorithm that sorts the buckets.**

Since the inner sorting algorithm of Bucket Sort must be a stable sorting algorithm, the potential inner algorithms are:

|  |  |
| --- | --- |
| Algorithm | Worst Case Time Complexity |
| Insertion Sort | **O(N2)** |
| Merge Sort | **O(N log N)** |
| Bubble Sort | **O(N2)** |
| Counting Sort | **O(N + k)** |

In these Sorting Algorithms, the worst Algorithm is Insertion Sort and Bubble Sort.

Let me use Insertion Sort for proving.

Worst-Case scenario: n=10 every element in one bucket

0.19 0.18 0.17 0.16 0.15 0.14 0.13 0.12 0.11 0.10

When inserting the elements into the buckets, it costs n time-costs.

Sort these reversed numbers in the bucket of 0.1 takes time of the Insertion Sort, which is n^2.

Thus, the time complexity for this scenario is n^2 (+n), where n^2 dominates.

Problem 7.2

a)

Hollerith.cpp

b)

This algorithm uses divide & conquer method. **Dividing method is Counting Sort.**

* From the most significant digit the array is being divided into 10 subgroups, each group starts with number 0-9.
* For every subgroup, if the total element is not 1 or 0, divide again, until k times, where k is the number of digits the MAX number has.
* The total time complexity will be O(Nk), where N is the number of elements.

Here is an example shows how this works.

![一些文字和图案

描述已自动生成]()