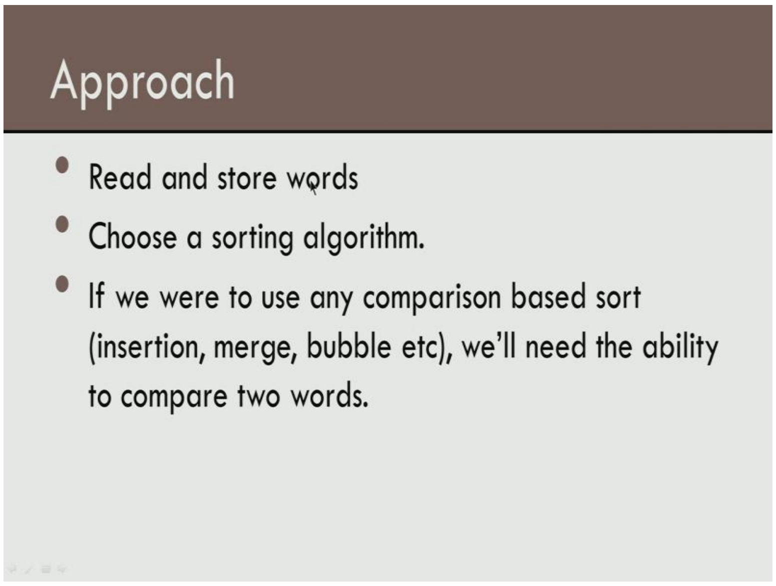
Programming, Data Structures and Algorithms

Prof. N. S. Narayanaswamy
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 35
Algorithms Discussion Sorting Words

In today's lecture we will look at one of the exercises given as a programming assignment in the programming data structure and algorithms course, in the week number 4. So, this is the most challenging programming exercise among the given once, and this is exercise involves writing the program to sort a given set of words. This is the discussion about this programming exercise, and what kind of programs one can write and the arguments of correctness.

(Refer Slide Time: 00:47)

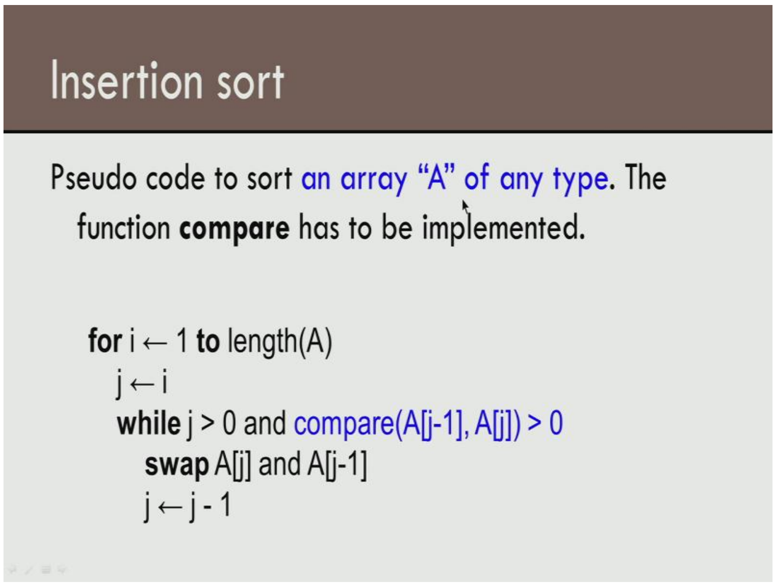


Approach

- Read and store words
- Choose a sorting algorithm.
- If we were to use any comparison based sort (insertion, merge, bubble etc), we'll need the ability to compare two words.

So, the exercise involves a collection of words as given as input, the first step that we will perform is that we will read the words, and store them in an appropriate data structure. In our case it will be that the data structure will be an appropriate array. Then you will choose a sorting algorithm to sort these words, and we will indeed use a comparison based sorting algorithm today, and it is important that if you use a comparison based sorting algorithm, then we should have the ability to compare two given words.

(Refer Slide Time: 01:28)



Insertion sort

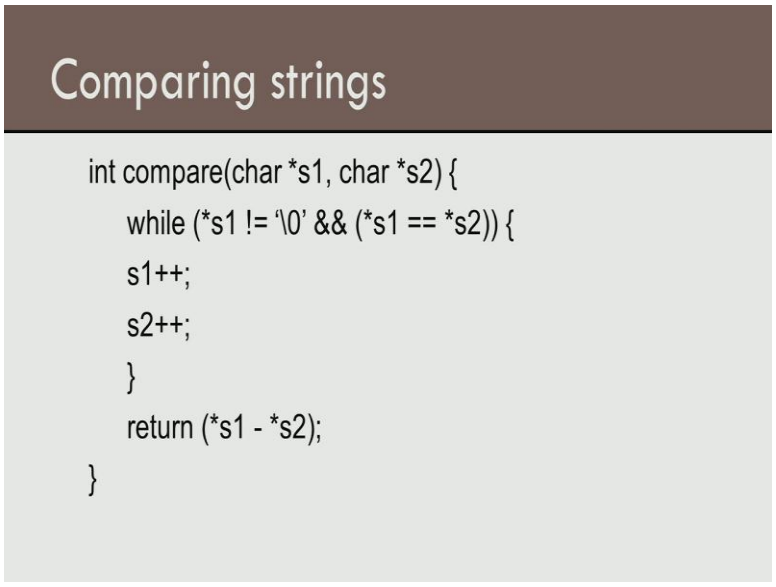
Pseudo code to sort an array "A" of any type. The function **compare** has to be implemented.

```
for i ← 1 to length(A)
  j ← i
  while j > 0 and compare(A[j-1], A[j]) > 0
    swap A[j] and A[j-1]
    j ← j - 1
```

Let us for example choose the insertion sort, and we all know the pseudo code to sort and sort an array which has integers. Here what we do is we look at the array from the first element to the last element, and in the second while loop we sort the first i elements. Recall this we have already study this during the insertion sort lectures, the main invariant of insertion sort is that if you look at the iteration number i, then the first i elements of the array will be sorted at the end of iteration number i. And this is really the pseudo code. Most important think to notice here is there is a comparison made between the elements of the array A, the j minus one th index element in the jth index element, and if you assume that it is an integer A, then conceptually there is very clear understanding a what is the result of a comparison between two integers.

This is highlighted in blue, and if you wanted to use exactly the same insertion sort idea to sort a given list of words, then we must have a clear understanding of how to compare two words and which of given two words are is the larger among the given two words, and whether they are equal and so on and so forth. We need to be able to compare two words. In particular whatever we are going to discuss really does not matter, what the elements of the array are as long as, they are of a single type. So, therefore, we imagine a function call compare, and the result of compare will basically tellers which of the two elements is a larger of the 2. So, let us assume that they elements are A of j minus 1, and A of j. The result of compare tells us that whether as swap must be performed or not. If the result is more than 0 then definitely a swap must be performed, essentially saying that A of j is smaller than A of j minus 1 saying that the two elements must be exchanged, and the result is equal to 0 then no swap is to be effected, because a of j minus 1 is smaller than a of j. So, therefore, what we are going to look at is the implementation of a compare function when the elements of the array are words.

(Refer Slide Time: 04:13)



Comparing strings

```
int compare(char *s1, char *s2) {
    while (*s1 != '\0' && (*s1 == *s2)) {
    s1++;
    s2++;
    }
    return (*s1 - *s2);
}
```

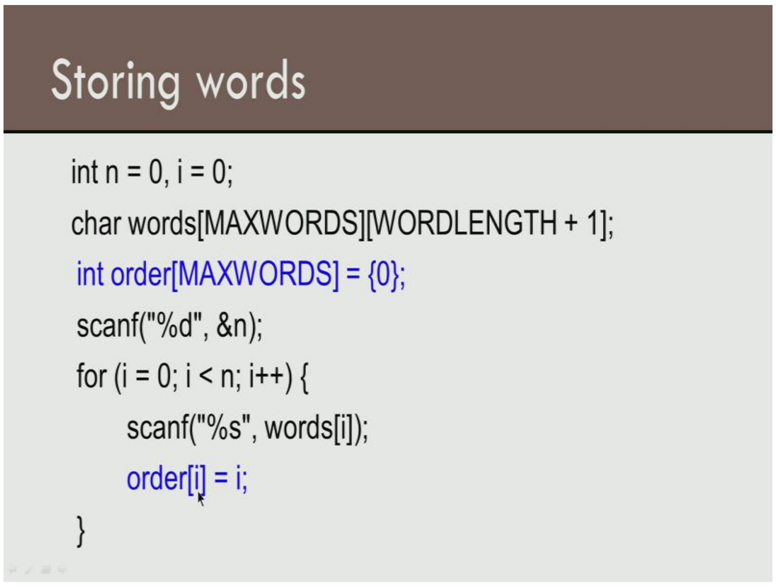
So, let us now look at this small function which is used to compare two strings. Let us recall at s 1 and s 2 are pointers to character arrays, and they are terminated with the null symbol each of them. So, that the contents of both these arrays are consider to be character strings. The output of this function is a 0, if the strings s 1 and s 2 are identical. If s 1 has higher dictionary order then s 2, in other words in the lexicographic ordering or in the dictionary ordering, if s 1 occurs after s 2 then the return value of this function will be a positive value, and if s 1 has a lexicographically lower rank then s 2; that is s 1 occurs or the string s 1 occurs before string s 2 in the lexicographic ordering or in the dictionary ordering, then the return value will be negative.

Therefore, the way of using this function in our sorting algorithm will be that, if s 1 is lexicographically later then s 2 in the dictionary order, then the return value will be greater than 0, and we will effect a swap that is how we are proposing to use the compare function in our sorting algorithm. So, let us just go through this function it is a nice and tricky implementation using function pointers, and the fact that characters can be compared by performing arithmetic on their corresponding integer values. It is very important to remember that the integer value corresponding to a character is it is unique integer code, which is often call the ASCII code use a programming in the C programming language. So, let us look at the compare function. What it does is that there is a while loop which iterates till s 1 reaches the end of string, and the control remains inside the loop as long as the character under process in the string s 1, and the character under process in the string s 2 are identical.

Let us understand this. If the end of string of s 1 is reached or if the character pointed to by s 1 and the character pointed to by s 2 are different, then control will exit from this loop. As long as these conditions both are satisfied, inside the loop the pointer value is just incremented by one unit, in this case by just the address of 1 byte. When control exits from the loop, the check that is performed is to check, if what s 1 and s 2 are pointing to or one and the same or is one smaller than the other or is one larger than the other, and the way is affected is by subtracting the integer value associated with the corresponding on corresponding characters.

Indeed if they are equal and the control is excited; it means that both the strings have come to the come to their ends, and therefore the strings are indeed equal and the return value would be 0. If the return is because of s 1 reaching the end of string, and s 2 not being the end of string, then it means that all of s 1 is contained inside s 2, and the value that would it be return is that s 1 is smaller than s 2 in the lexicographic ordering. And if s 1 the value at the location that is pointed to by s 1 is smaller than the value at the location pointed to by s 2, then this subtraction will return in negative value saying that the s 1 is smaller than s 2. Indeed if it is going to return a positive value then it means that the string s 1 in the dictionary order occurs after the string s 2, and therefore we will use it to exchange to effect an exchange or effect a swap in our usage.

(Refer Slide Time: 09:15)



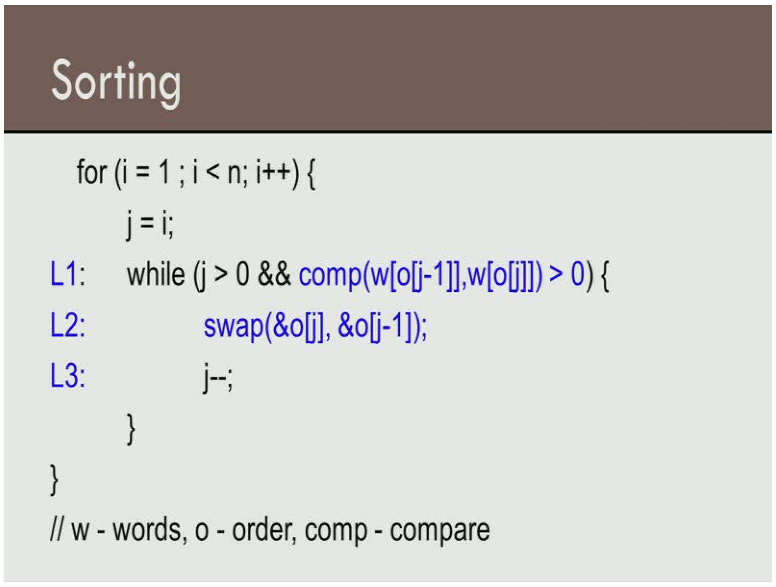
Storing words

```
int n = 0, i = 0;
char words[MAXWORDS][WORDLENGTH + 1];
int order[MAXWORDS] = {0};
scanf("%d", &n);
for (i = 0; i < n; i++) {
      scanf("%s", words[i]);
      order[i] = i;
}
```

So, first we create an array of words, it is a two-dimensional array. The number of rows is max words, which is a constant as given in the input, and the length of the word - the length of the longest word is given and we use one more location and this is only to ensure that we will we will store the words, as strings we will use one more letter or one more character position to store a null. So, now we read the number of words that need to be sorted, this is what the first scanf does and then the for loop here reads the strings one after the other, the words one after the other and source it in the array words.

It is very important here to note that if the words are of length more than word length, then the way we of this program is unpredictable, therefore it is extremely important to ensure that the words that are given as input are only words of exactly word length, the last letter will be used to store a null. Then we have an initialization step of another array call order, and this is the very crucial array which is what we base are insertion sort algorithm on. So, order is another array, there are as many cells in this array, as there are the number of words or the maximum number of words, and all of them i initialize to 0. Then as we read the input words, the order of the ith word is initialize to i, this is just a initialization step and most importantly observe what we are doing, we reading the input words from the input and storing at into the array words. And we are giving a initial ranking which good be wrong which will be wrong on most inputs like unless the words are already sorted, and the ith word is given the order i. And it is this order that will be changed when you perform a insertion sort.

(Refer Slide Time: 11:57)

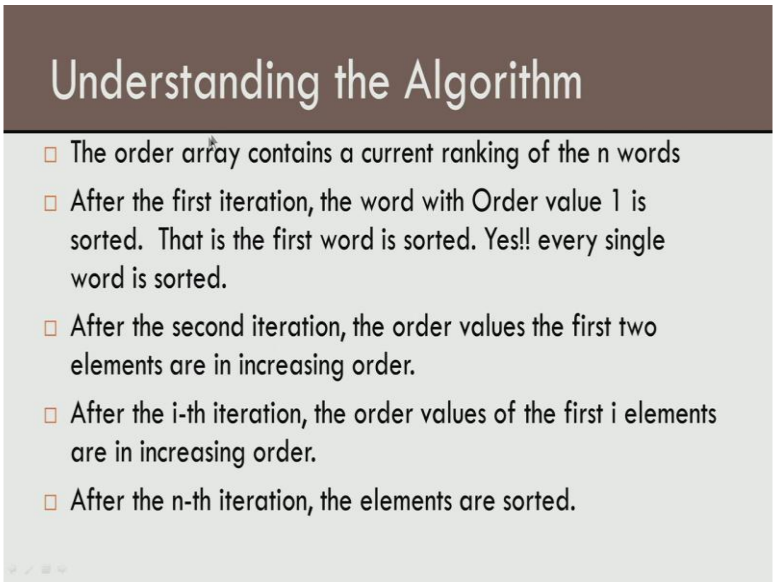


Sorting

```
   for (i = 1 ; i < n; i++) {
          j = i;
L1:       while (j > 0 && comp(w[o[j-1]],w[o[j]]) > 0) {
L2:                 swap(&o[j], &o[j-1]);
L3:                 j--;
          }
}
// w - words, o - order, comp - compare
```

So, now we go to the sorting algorithm itself. For the usage of space appropriately with a int use the word use the space appropriately, we have taken a bit of shortcut; comp stands for the compare function which we talked about earlier, w letter is a shortcut for the words array, and o letter is the shortcut for the order array that we have defined earlier . So, now what we do again like in insertion sort; there are two loops. The outer loop runs over the indices 1 to n, and what we do is we compare the first at the end of this while loop we will guarantee that the prefix of the elements, that is if you look at the ith iteration of the for loop at the end of this while loop, the first i elements will definitely half the order values in ascending order. So, let us look at this comparison in the ith iteration j is set to i, then while j is more than 0 and here j is decremented end at the end of this. We compare the word who is order is j minus 1, and the word who is order is j, and if the comparison gives you value more than 0 which means they must be exchanged, then we swap the words order of j and order of j minus 1. We swap the values of order of j and order of j minus 1, then we decrement j minus 1.

(Refer Slide Time: 13:53)



Understanding the Algorithm

- The order array contains a current ranking of the n words
- After the first iteration, the word with Order value 1 is sorted. That is the first word is sorted. Yes!! every single word is sorted.
- After the second iteration, the order values the first two elements are in increasing order.
- After the i-th iteration, the order values of the first i elements are in increasing order.
- After the n-th iteration, the elements are sorted.

So, let us understand this algorithm. Initially we have a current ranking of the n words, at the end of the first iteration the invariant as we know is that the word with order value one is sorted; that is in this case of first word is sorted, indeed every single word is sorted. After the second iteration, the order values are the first two elements are in increasing order, and after the ith iteration the order values of the first i elements are in increasing order, consequently after the nth iteration all the elements are sorted. Therefore, this is an algorithm to sort a given list of words observe that we have use insertion sort in a very clever way using this order data structure or the order array which whose values are change, and using the comparison function of two words.

Thank you.