Second Level Algorithms

Prof. Palash Dey

Department of Computer Science and Engineering

Indian Institute of Technology, Kharagpur

Week – 11

Lecture 51

Welcome to the 51st lecture of the second-level algorithm course. In this lecture, we will see a linear-time algorithm, a worst-case linear-time algorithm for the selection problem or the order statistics, okay? So, let us begin. So, we have seen in the last class that if we can find a pivot in O(n) time with the guarantee that there exists a constant d between 0 and half strictly more than 0 but less than or equal to half, such that l is between dn and $1-dn$. we can find the k-th smallest element of an n-element integer array for any k in O(n) number of comparisons. Actually, you will take it as homework to verify that the actual time complexity is also O(n). Although, for simplicity, we will only study the number of comparisons, the time complexity of this algorithm is also O(n). So, here also, let us make the number of comparisons instead of time in O(n) or with O(n) comparisons.

So, in this lecture, we will now see the approach. The idea is as follows: we divide the array into $n/5$ groups, where each group contains 5 elements, except the last group, which contains at most 5 elements.

Again, I strongly encourage you to make the simplifying assumption that n is divisible by 5. So that we do not have to worry about these boundary cases, and once you understand the algorithm, it will be clear how to handle the other cases. So, pictorially, this is how it looks like: this is the first group of elements. $A[1]...A[5]$, the second group of elements is $A[6]...A[10]$, and so on. So, we have n/5 groups of elements. Each group has 5 elements exactly, except the last group, which has at most 5 elements. So, this is the first step: we are not doing anything; we are just arranging the elements or grouping the elements in our head. The task now begins with this step: we sort each group of elements using any sorting algorithm of our choice. We can use insertion sort, bubble sort, heap sort—anything we like—because each group has only 5 elements, so sorting each group takes $O(1)$ time each.

Or, instead of time, let us write the number of comparisons as $O(1)$ comparisons each. The total number of comparisons in this step is then $O(n)$ because there are n/5 groups, and each group takes $O(1)$ time. So, the total number of comparisons is $O(n)$. So, assume then that we have sorted each of the groups.

So, this is sorted. Each of these groups is sorted in this order, from smallest to largest. So, $A[1]...A[5]$ is sorted, $A[6]...A[10]$ is sorted, and so on. Now, here is a crucial claim: for each group, find the median. So, for example, the first group's median is a3 after sorting. The median of the second group is $A[8]$, and so on. So, after sorting, they are available. No more comparison is required.

So, let the medians be $m_1, m_2, \ldots, m_{n/5}$. No comparison is performed. in this step. Now, here is the claim: the median of these $n/5$ medians is a good pivot. Let us call it M. Let us call this median M a good pivot in the sense that there exists a fraction d, a constant d, such that after partitioning, the index of the pivot l is between dn and $(1-d)n$. So, let the calculation give us the value of d. So, let us prove this claim. So, first observe that there are. So, for simplicity, let us call it ay x . So, you observe that there are $x/2$ elements in the set $m_1, m_2, \ldots, m_x$ which are less than the median of median let us call M the median of median M. Now, for each such for each such corresponding medians which are smaller than M there are three elements which are smaller than those mediums.

Because a median in it is a median of a group of five elements and if I count that median there are two more elements which are smaller than that. So, for each such medians there are three elements including the median. in its group of 5 elements ok. So, at least 3 times $x/2$ elements are smaller than the pivot m. by similar logic at least. So, this is 3 times x by 10 this is now x is $mn/5$. So, this $3n/10$.

By similar logic, at least $3n/10$ elements are larger than the median of medians M, ok? And this is the definition of a good pivot. So, we can choose d equal to $3/10$. So, let us write here, in particular, at least $3n/10$ of the array A are smaller than M, and at least 3n by 10 elements of the array A are larger than M. So, it is a good pivot, but how will you find that? The master stroke is the last step: we find the median of these medians by calling the function itself recursively. This step performs $T(n/5)$ comparisons.

Hence, the overall time complexity, or the overall or the total number of comparisons performed by the algorithm in the worst case, follows the following recurrence relation. So, because it is a recursive function, there will be a base case. So, assume we will

compute the median by simply sorting it if the number of elements is less than or equal to some constant, say 100.

So, this is then some constant, say c_1, if n is less than or equal to, say, 100. Otherwise, what are we doing? We are first, let us compute the number of comparisons. The first step is grouping.

So, no comparison. The second step is sorting each group, which takes $O(n)$ time. So, suppose then that is at most, say, c_2 of n, that is what $O(n)$ means. This is to sort $n/5$ groups of 5 elements each, $c_2 n$. Then what is the next step?

We are finding the median of each group; there is no comparison involved here. Then we are calling recursively to find the median of these $n/5$ elements. So, that takes $T(n/5)$ number of comparisons, and then after this step. So, this time is, or this number of comparisons the algorithm uses to find a good pivot. Now, once the pivot is found then the recursive algorithm of the last lecture is followed that means, we are performing a recursive call on the suitable part. Now, each part one part will have at or both part has at least $3n/10$ elements that means, the larger part has at most $7n/10$ elements. So, in the worst case this is then $T(7n/10)$. So, this is for the recursive call on the suitable on the suitable sub array on the suitable sub array based on the values of k and l. What was l? It was after the partition subroutine returns l is the index of the pivot.

So, this is the recurrence relation for the number of comparisons In the next lecture we will solve this recurrence relation to see that $T(n)$ is $O(n)$ and then we will also write a pseudocode of this algorithm to ensure that we have all understood the algorithm crystal clear fashion ok. So, let us stop here. Thank you.