Second Level Algorithms

Prof. Palash Dey

Department of Computer Science and Engineering

Indian Institute of Technology, Kharagpur

Week – 10

Lecture 49

Welcome to the 49-th lecture of the second-level algorithms course. In the last lecture, we started order statistics. Let us continue that topic in this lecture also. So, let us recall the problem definition of order statistics: given an integer array A of size n and an integer k, find the k-th smallest integer of the array A. We have seen in the last class that for k equal the number of comparisons needed is $n-1$. We have seen an algorithm that finds the minimum with $n-1$ comparisons—the natural algorithm—and we have also said that it is tight without proof. Then we were studying the case k equal to That means finding the second minimum, and towards that, we introduced the idea of pairing the elements, comparing the pair of elements, and then comparing the winners, thereby building what we called a search tree, and we observed that the root element is the minimum integer of the array A, OK. So, by comparing A_1 and A_2, their parent is the minimum of A_1 and A_2. So, every internal node is labeled with the minimum of its two children. And we also observed that the number of internal nodes is $n-1$, which is the number of comparisons needed to find the minimum.

Now, we will we want to find the second minimum and a crucial observation is that the height ceiling of $\log_2 n$ ok. So, for example, if $n=2$ then it has only height is 1 and n equal to 3 height is 2, n equal to 4 height is 2, and so on. The height of a rooted tree is the number of edges is the maximum number of edges in any path from the root to leaf ok. Now, a crucial observation is that the minimum element Again I would suggest that do not get bothered about the existence of unique minimum or not what if there are duplicates. When you are first understanding any algorithm you assume that there are no duplicates make simplifying assumptions understand the algorithm and then you will be able to handle the cases when there are duplicates. So, again so, assume that the input array A does not have duplicate elements. So, all elements are distinct. So, the minimum second minimum all are unique and you can take this as a homework to modify if needed this algorithm to work for arrays with repetition. A crucial observation is that the minimum element defeats log n other elements indeed wherever the minimum is it has to reach to the root and the length of lift to root path is $\log_2 n$. So, that many elements it defeats to reach the root and more importantly the second minimum must be one of the elements one of the log n elements defeated by the minimum element ok. So, to find the second minimum, it is enough to find the minimum $\log_2 n$ elements defeated by the minimum element. So, you have to find the minimum of this $\log_2 n$ element and this can be done by $\log_2 n-1$ comparison. Hence both the minimum and the second minimum element of an n element array can be computed with $n-1+\log_2 n-1$ which is $n+\log_2 n-2$ comparisons. which is a substantial improvement from the naive $2n-3$ comparisons. This is a substantial improvement from the $2n-3$ comparisons that the naive algorithm ok.

Next, let us see another interesting problem slightly related to order statistics, but not exactly. We will come back to the general order statistics problem. So, this problem is called the max-min problem. Here, the input is again an integer array of size n, and the output should be the maximum and minimum of the array A. Again, let us see what the naive algorithm does. The naive algorithm first finds the minimum of the array A with $n-1$ comparisons and then finds the maximum of the remaining $n-1$ elements with $n-2$ comparisons. Hence, the naive algorithm makes a total of $n-1+n-2$, which is $2n-3$ comparisons to compute the minimum and the maximum simultaneously. of an n-element integer array A. Now, we will see an algorithm which improves the number of comparisons from $2n-3$, and the idea is to again pair up the elements. We pair up the elements into $n/2$ pairs.

We perform a comparison between the pair of elements to find the minimum. and the maximum of every pair of elements. OK, that means you pair up $(A_1, A_2), (A_3, A_4)$, and so on. Then, with one comparison, you compute the minimum of every pair and the maximum. Now, we know there is an observation: observe that the minimum of the array must be present or must be the minimum of $n/2$ minimums. Again, I would strongly recommend that you assume that n is an even integer. So that you do not have to worry about floor or ceiling. First, understand the algorithm. Once you have understood the algorithm, it will be an easy exercise for you to change $n/2$ with floor or ceiling of $n/2$, depending on the situation or where it is appropriate.

Similarly, the maximum of the array A must be the maximum of. So, this should be floor or ceiling. must be the maximum of $n/2$ maximums, OK. These can be computed.

with $n/2-1$ comparisons each. Hence, the total number of comparisons performed by the algorithm to simultaneously find the minimum and the maximum of an n integer array is first $n/2$ comparisons for the pairs $n/2-1$ comparisons to find the maximum of maximums and minimum of minimums. So, this $n/2-1$, which is a substantial improvement from $2n-3$ number of comparisons performed by the naive algorithm. OK. So, let us stop here. Thank you.