Second Level Algorithms

Prof. Palash Dey

Department of Computer Science and Engineering

Indian Institute of Technology, Kharagpur

Week – 11

Lecture 52

Welcome to the 52 lecture of the second-level algorithm course. In this lecture, we will finish our study of the linear-time selection algorithm. So, let us begin. So, in the last class, we described the linear-time selection algorithm and showed how the running time is big O of n. In this class, let us see the pseudocode of the linear-time selection algorithm. So, it is a recursive procedure.

So, let us call the function select, which takes an integer array of size n and is supposed to return the k-th minimum element of the array. So, we will ensure that k is between 1 and n. First, the base case: if n equals 1, then k must also be 1, and in this case, we return a[1]. Else, what do we do? We divide the array into groups of size 5 and sort them, sort the subarray A[5i + 1]. So, this subarray we sort for all i from 0 to n/5 - 1. So, this takes linear time. So, now we know the medians of each subarray; for example, the median of the first subarray is at A[3], the second subarray is at A[8], and so on. So, we recursively compute the median of these medians.

for that we use the select function itself and pass this set of numbers which is n by So, these are $n/5$ numbers and we want to find the median that means, $n/10$th smallest element ok. Once we get a median of this medians we use it to pivot to partition with respect to this And suppose the partition subroutine returns the index of the pivot which is suppose it is l. Now if l is equal to k then we know that the pivot is the kth smallest element then we return m the pivot else if l is greater than k then the kth smallest element of a 1 to n is the k-th smallest element of a 1 to $l-1$.

So, in this case we return using recursive call on the left subarray. we find the k-th smallest element of a 1 to n by finding the $(k-l)$-th smallest element of the right subarray. So, that is it. So, now let us say number these steps suppose this is 1.

So, let us see which step takes how much time. step 1 takes $O(1)$ time, step 2 takes $O(1)$ time. So, 1 and 2 takes this time. In step 3 we are sorting $n/5$ arrays each of size at most 5. So, each sorting step takes $O(1)$ time because the number of elements we are sorting is at most 5. So, step 3 takes $O(n)$, step 4 we are recursively calling the function with n by 5 element to find the median. So, step 4 takes $n/5$. In step 5 we are partitioning the array A with n integers with respect to a partition which takes $O(n)$ time in the worst case step 6 is $O(1)$. In step 7 we are calling the function recursively on an array of size and in step 8 we are calling the function recursively on an array of size $n-l$. So, thereby the recurrence solution that we get is $T(n)$ is $O(1)$. is n equal to 1 otherwise it is $T(n/5)$ let us write less than equal to $T(n)$ is less than equal to $T(n/5)$ plus 1 of 7 or 8 gets called not both of them. So, in worst case this is at most So, we have seen that the median of median if you look at recall our picture that this is the how the array elements were arranged pictorially is $n/5$ and whatever is the median at least $n/10$ medians are less than the median of median m and for each such median 3 elements are less than. So, the size of one part is at least $3n/10$. So, the size of the other part is at most $7n/10$. So, that is how I get $7n/10$ Plus $O(n)$, this is otherwise, and we have seen in the last class how by solving this recurrence relation we can prove that T(n) is $O(n)$. So, now let us solve this recurrence relation. There are various methods for solving recurrence relations, and the method we will use is the substitution method. So, we will solve this recurrence relation by the substitution method, ok. So, to apply the substitution method, we first need to get rid of this big O notation.

So, $O(1)$ means at most some constant C, $O(n)$ means at most some constant C n, and we can assume without loss of generality that the constants are the same by choosing the higher of the constants because of this less-than equality. So, we have the following recurrence relation: T(n) is less than or equal to C if n equals 1, otherwise T(n) is less than or equal to T(n/5) plus T(7n/10) plus C otherwise, ok. Now, let us claim that T(n) is less than or equal to T the exact constant d. We will choose after we do the calculation. Our calculation will help us determine what d we should choose. So, d, of course, has to be greater than C. Let us pick d greater than or equal to C. So, the base case holds. T(1) is less than or equal to C, which is given, which is less than or equal to D. Now, the inductive are given the recurrence relation T(n) is less than or equal to T(n/5) plus T(7n/10) plus By the induction hypothesis, we will assume that T(n) is less than or equal to dn for all n less than n, for all integers 1 to $n-1$.

So, this is the inductive hypothesis. In particular, the inequality $T(n) \leq d\, n$ holds for $T(n/5)$ and $T(7n/10)$. By substituting, what we get is $T(n) \leq d\, n/5 + 7\, d\, n/10 + c\, n$. So, now, we will pick d such that the right-hand side is less than $d\, n/5 + 7\, d\, n/10 + c\, n \leq d\, n$, that is $d/10 \geq c$, that is $d \geq 10\, c$. So, we pick any constant $d \geq 10\, c$. Now, we choose d to be equal to 10 c.

With this particular choice of d, because c is a constant, d is also a constant. With this choice of d, we have shown that $T(n) \leq d\, n$. Hence, $T(n)$ is $O(n)$. In particular, you can prove the following recurrence relation: $T(n)$ is $T(n) \leq O(1)$ if $n \leq$ some constant, say d; otherwise, $T(n) \leq c_1\, n + T(c_2\, n) + T(c_k\, n) + O(n)$, otherwise where $c_1 + c_2 + ... + c_k < 1$, then $T(n)$ is $O(n)$. This proof you can take as homework. Another interesting homework is the following: Instead of grouping the integers of the array into sizes of 4 or 5, we group the integers of the array into any size.

less than 5, then the worst case runtime of the selection algorithm is not big O of n ok. Another interesting homework is instead of grouping the integers of the array into groups of size 5 if we group the integers of the array into groups of any size more than 5, then the worst case runtime of the selection algorithm continues to be $O(n)$ of course, the size must be constant any constant size. So, instead of groups of size 5 you can pick groups of size 6, 7, 8, 9, 10 does not matter the run time of the algorithm in the worst case continue to be $O(n)$. On the other hand, if your group size becomes smaller than 5 say 4, 3 or 2, then the worst case runtime of the new selection algorithm is not $O(n)$ anymore ok. So, this is what we want to study in linear time selection algorithm. In the next lecture we will start another very important topic which is string matching ok. So, let us stop here. Thank you.