CSE408
Lower bound theory

Lecture # 31

Lower bound

- We always consider the algorithm with smaller order of complexity as the better.
- How can we ensure that is there any faster method available
- We need to find a function g(n) which is a lower bound on the time that algorithm must take, So if f(n) is the time for some algorithm then we can write f(n)=Ω(g(n) where g(n) is lower bound for f(n)

Lower bound theory

- Or we can say f(n) >= c * g(n) for all n>n0
- Where c & n0 are constants
- For many problems it is easy to calculate the lower bound on n inputs
- e.g. consider the set of problems to find the maximum of an ordered set of n integers. Clearly every integer must be examined at least once. So Ω(n) is a lower bound for that. For matrix multiplication we have 2n2inputs so the lower bound can be Ω(n2)

Sorting and Searching

- For all sorting & searching we use comparison trees for finding the lower bound.
- For an unordered set the searching algorithm will take Ω(n) as the lower bound. For an ordered set it will take

- Ω(logn) as the lower bound. Similarly all the sorting algorithms can not sort in less then Ω(nlogn) time so Ω(nlogn) is the lower bound for sorting algorithms.

Oracle and adversary arguments

- Given some computational model, the oracle tells the outcome of each comparison. In order to derive a good lower bound, the oracle tries its best to cause the algorithm to work as hard as it might.
- Take the example of a tournament where the values are called players and the largest value is interpreted as the winner and the second largest is taken as the runner up. So the problem is the similar to finding the largest and the second largest number out of a set of n numbers.

Second largest

- Since we can't determine the second largest element without having determined the largest element, at least n-1 comparisons are necessary. We need to show that there is always some sequence of comparisons which forces the second largest to be fund in (logn)-1 additional comparisons

Tournament theory

- The winner of the tournament has played x matches then there are x people who are candidates for the runner up position. The runner up has lost only once to the winner and other x-1 persons must have lost to one other person. We can produce a oracle which decides the results of matches in such a way that winner plays logn other people.
- In a match between a and b the oracle declares a as the winner if a is previously undefeated and b has lost at least once or if both a and b are undefeated but a has won more match then b. In any other case the oracle can decide arbitrarily as long as it remains consistent.

Winning match

- Corresponding to this tournament imagine drawing a directed graph with n vertices. Each vertex corresponds to one of the n players. Draw a directed edge from vertex b to a iff either player a has defeated b or a has defeated another player who has defeated b. Since for the overall winner there must be an edge from each of the remaining n-1 vertices, it follows that the winner must have played at least logn matches.

Increasing matrix

- M is an nXn integer matrix in which the keys in each row are in increasing order. Consider the problem of finding the position of an integer x in M. Determine a lower bound for the no. of comparisons to be done in the problem.

L
P
U

Thank You !!!