CS330 – Selection Analysis and Lower Bounds

Purpose: We will examine algorithms that select an item in a list that has a desired property, like the minimum, maximum, or k^{th} largest or smallest for some k. Again, once we have an algorithm we also want to know if this is an optimal algorithm and therefore we will develop some techniques for finding the lower bound.

Knowledge: This activity will help you become familiar with the following content knowledge:

- Possible selection algorithms
- Using adversary arguments for showing lower bounds

Activity: With your group perform the following tasks and answer the questions. You will be reporting your answers back to the class in 90 minutes.

1. Consider this algorithm for finding the maximum value in an array:

```
int FindMax(int[] list,int low, int high){
   int max = low;
   for(int i=low+1;i<=high; i++)
      if (list[i]>list[max])
           max = i;
   return max;
 }
```

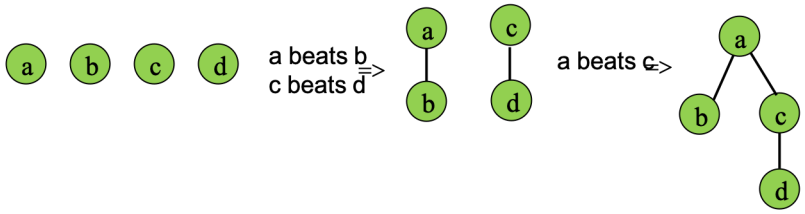
   What is the cost for a list of size n? Do you think this is optimal?

2. We will think about the problem of finding the max as a series of contests (comparisons) where the max value of the comparison wins the contest. The maximum in the list will be the ultimate winner. We will try a lower bound proof for this problem.:

   The winner must compare against all other elements, so there must be $n - 1$ comparisons.

   What is the flaw in this argument? Can you give an example where the max need not be compared to all other elements?

3. We can look at the process of comparison by using "partially ordered sets" or posets. Suppose four opponents have matches to determine the winner. This process may be pictured as posets:



   Ignoring symmetries, draw four other posets that find a winner for this problem. (Be careful not to draw extraneous or unneeded comparisons).

4. To find the max, we must build a poset having one max and $n-1$ losers, starting from a poset of n singletons. We wish to connect the elements of the poset with the minimum number of links.

   Why can we now argue that this will require $n-1$ comparisons? We now have a lower bound proof!

5. Suppose we want to find the 2^{nd} largest value. One algorithm would be to find the max, discard it and then find the max of what is left.

   What is the cost? Do you think this is optimal?

6. Again, we want a lower bound proof for the 2^{nd} max problem. Here is an attempt:

   Anyone who lost to anyone who is not the max cannot be second. So, the only candidates are those who lost to max. Findmax might compare max to $n-1$ others. Thus, we might need $n-2$ additional comparisons to find the second max.

   What is wrong with this argument?

   Hint: There is something called the "necessary fallacy". In this case it would say that our algorithm does something, therefore all algorithms solving the problem must do the same thing.

7. Here is a "divide and conquer" approach to finding the 2^{nd} max:

   Break the list into two halves and then use Findmax on each half. Compare these two winners and use Findmax on the winner's half for locating the 2^{nd} in that half. Finally compare that 2^{nd} to the winner of the other half to get the overall 2^{nd} max.

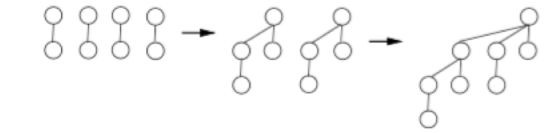
   What is the cost? Do you think this algorithm is optimal?

8. If dividing in half improves things what would be the cost if we broke the list into four pieces instead? What about eight pieces?

9. The only candidates for the 2^{nd} max are losers to the eventual winner. A data structure to maintain that information efficiently is called a binomial tree. A binomial tree of height m has $2m$ nodes organized as:

   (a) a single node, if $m = 0$, or

   (b) two height $m - 1$ binomial trees with one tree's root becoming a child of the other.



   If there are n nodes in the binomial tree, how many candidates are there for 2^{nd} max?

10. So here is another algorithm for 2^{nd} max:

    Build the binomial tree and then compare the $\log n$ children of the root for the 2^{nd} max.

    What is the total cost?

11. A technique used in some lower bounds proofs is an "adversary". The algorithm asks the adversary for information about the input. The adversary may never lie but can choose input to make the algorithm work as hard as possible.

    Imagine that the adversary keeps a list of all possible inputs. When the algorithm asks a question, the adversary answers and crosses out all remaining input inconsistent with that answer. At any point, the adversary is permitted to give any answer that is consistent with at least one remaining input.

    Explain how an adversary would make an algorithm work as hard as possible in a game of Hangman? Remember that the word is not fixed and the adversary gets to pick the input and hangman answers as the algorithm is performed as long as those picks are consistent.

12. We will apply an adversary to the 2^{nd} max problem:

    At least $n - 1$ values must lose at least once so this will involve $n - 1$ compares. Have k direct losers to the winner which must be compared as possibilities for the 2^{nd} max. Therefore, overall there must be at least $n + k - 2$ comparisons.

    What question are we asking for a lower bound proof?

13. Call the "strength" of an element $L[i]$, the number of elements that $L[i]$ is known to be bigger or equal than. If $L[i]$ has a strength a, and $L[j]$ has a strength b, then the winner of a contest between them will have strength $a + b$.

    The adversary wants to minimize the rate at which an element gets stronger. So on a contest between two candidates, which one will the adversary pick as the winner? Why?
    Hint: Consider what will happen to the strengths if $a \geq b$. Which pick will cause the smallest increase?

14. So from the result above, at each contest an element's strength will at most double. After k comparisons an element's strength is less than or equal to 2^k.

    What is the lowest value we can have for k as losers to the max? What does that say about the optimality of the binomial tree algorithm?

Complete the following assignments to be submitted for grading. Each should be done individually but you can consult with a classmate to discuss your strategies.

Assignment 1:
Consider finding the median of three order-able elements A, B, and C:

1. Draw the possible posets of three elements (don't label them) after two comparisons to demonstrate that any algorithm must perform at least three comparison in the worst case.
2. Design an optimal algorithm to find the median.
3. Draw the decision tree for your algorithm and explain how the decision tree gives the worst case cost.

Criteria for Success: You have three things: a lower bound proof with posets, an optimal algorithm, and a decision tree.

Assignment 2:
Consider determining if at least three of five integer values are non-zero using only operations of testing for equality to zero. Give an adversary strategy to force **any** algorithm to examine all of the five values. Explain why your adversary strategy works regardless of what comparison algorithm is used.

Criteria for Success: You have a clear explanation of the adversary strategy for choosing the input value at each comparison to force any algorithm into worst case behavior.

Assignment 3:
Use an adversary argument to prove a lower bound (an optimal worst case behavior) for any algorithm which determines if there are any duplicates in a *sorted* array of size n. In other words, how many comparisons can the adversary force **any** algorithm to perform and what strategy does the adversary use? (Remember that the adversary's answers must be consistent.)

Criteria for Success: You have a clear explanation of the adversary strategy on choosing the input to force any algorithm into worst case behavior.