Tutorial 3

Selection: Adversary Arguments

What is an Adversary?

- A method for obtaining worst case lower bounds
- A second algorithm which intercepts access to data structures
- Constructs the input data only as needed
- Attempts to make original algorithm work as hard as possible
- Analyze Adversary to obtain lower bound

Important Restriction

- Although data is created dynamically, it must return consistent results.
- If it replies that x[1]<x[2], it can never say later that x[2]<x[1].

Adversary Lower Bound Technique

- Devise a strategy to construct a worst case input for a correct algorithm.
  - The algorithm is known, i.e. Insertion sort
  - The algorithm in unknown, i.e. comparison-based sorting algorithm
- Guessing Game:
  Z_{100} = {0, 1, …… , 99}, Guess what number in Z_{100} I have in mind?
  - $|L0|=100$, $|L1|\geqslant 50$, $|L2|\geqslant 25$, $|L3|\geqslant 13$, $|L4|\geqslant 7$, $|L5|\geqslant 4$, $|L6|\geqslant 2$, $|L7|\geqslant 1$
  - Worst case lower bound: $\lceil \log_2 100 \rceil = 7$

Design against an adversary

- A good technique for solving comparison-based problem efficiently.
- Should choose comparisons for which both answers give the same amount of information
- Keep the decision tree as balance as possible
- Binary search, merge sort, finding both max and min, finding second-largest

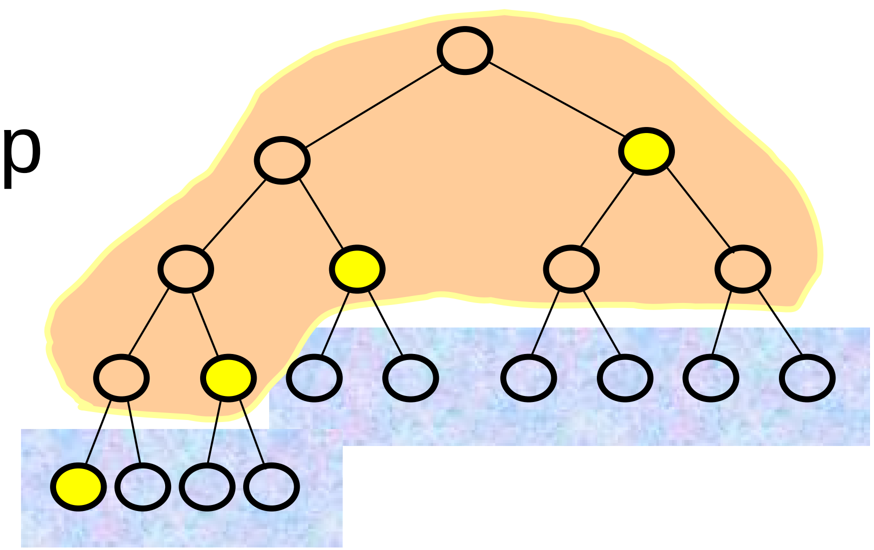
(1) Finding both max and min

- Finding max and min
  - (1) pair up comparison: n/2
  - (2) find largest of the winners: n/2-1,

    find smallest of the losers: n/2-1
  - (3) at least 3n/2-2 comparisons

(2) Finding second-largest key

- Finding second-largest key
  - (1)finding the max of n keys: n-1
  - (2)finding the largest of keys directly lose to max: [lgn]-1
  - (3)at least n+[lgn]-2
- implementation: heap



(3) Finding median

- Selection (Finding median)
  - Divided and conquer approach
- Find a "good" partition?
  - in finding pivot for Quick sort, we have
    - $T(n)=T(q)+T(n-q-1)+\Theta(n)$
    - (1)fixed strategy
    - (2)random strategy
  - for selection
    - $T(n)=T(\max(q, n-q-1))+\Theta(n)$
    - (1)fixed strategy: $\Theta(n^2)$ in the worst case
    - (2)random strategy: [CLRS P189] expected $\Theta(n)$
    - (3) group 5 strategy: $\Theta(n)$ in the worst case
- lower bound (textbook P240): 3n/2-3/2, (2n,3n)

- Questions
- Why select 5 keys as a group? can it be 3,4,6,7,…?

  Yes, we can choose c keys as a group, but we must have c>=5 to run in linear time.

  (Explain why c<5 is not in linear time?)
- Finding the median of 5 elements ?

  (6 comparisons)
- Sorting 5 elements?

  (7 comparisons)

Counting the Number of Comparisons

- Assuming *n*=5(2*r*+1) for all calls of *select*.
- $$W(n) \le 6\left(\frac{n}{5}\right) + W\left(\frac{n}{5}\right) + 4r + W(7r+2)$$

The extreme case: all the elements in A∪D in one subset.

Finding the median in every group of 5

Finding the median of the medians

Comparing all the elements in A∪D with *m**

- *Note: r is about n/10, and 0.7n+2 is about 0.7n*, so

$$W(n) \le 1.6n + W(0.2n) + W(0.7n)$$

$W(n)=1.6n+1.6*(0.9)n+1.6*(0.9)^2n+1.6*(0.9)^3n+\ldots= \theta\ (n)$

Example: Lower Bound for Comparison Sort

- Input: there n! different permutations
- The adversary D maintains a list L
- Adversary Strategy:
  - Initially L contains all n! permutations
  - When an algorithm compares ask a[i] < a[j]?
    - Let L1 be the permutation in L and a[i]<a[j]
    - Let L2 be the permutation in L and a[i]⩾a[j]
    - If |L1|>|L2|, answer “yes”, and let L=L1
    - Else answer “no” and let L=L2
  - At least half of the permutations in L remain
  - The algorithm is done until |L|=1
- So, the number of comparison is at least

$$\lceil \log_2(n!) \rceil \geq \left\lceil \log_2\left(\frac{n}{e}\right)^n \right\rceil = \Omega(n \log n)$$

Ex1: Majority element problem

- A majority element in an array A of size N is an element that appears more than N/2 times. For example, the array

  1,3,2,3,2,3,3 has a majority element 3;

  1,3,2,3,2,4 has no majority element.

  The majority element problem is to find the majority element in an array, output –1 is it does not have one.

- Method 1: Counting the appearance times of each element
- The time complexity is $O(n^2)$

- Method 2
  - (1) Sorting the array in O(nlgn) time
  - (2) Find the longest duplicated element in O(n) time
- Thus the complexity of the algorithm is O(nlgn)

- Method 3: (linear solution)
- Assume n is even, we find the candidate majority element as follows: we pair up element A[2i-1] with A[2i], for I=1,2,…n/2, for each pair, if two elements are equal, put the element into array B, else discard both of them. B is the candidate set, where |B|<=n/2. we have the following claim.

- Claim: if n is even, e is the majority element of A[1..n] and B is the elements which survived the above procedure, then B has a majority element which is equal to e.
- proof: Suppose that k is the number of pairs created by the above procedure, in which both elements are equal to e. Suppose, further, that L is the number of pairs created by the procedure which contain unequal elements. Clearly, |B|=n/2-L. Moreover, since e appears in A at least n/2+1 times it must hold that 2k+L>=n/2+1. This implies

$$k \geq \frac{n/2-L}{2}+\frac{1}{2} \Rightarrow k \geq \frac{|B|}{2}+\frac{1}{2}$$

Hence e is a majority element of B.

- If n is odd:
  - If the first N-1 elements have a majority, then the status of the last element to be a candidate or not cannot change the fact.
  - If no majority element emerged in the first N-1 elements, the last element could be a majority.

- It is not hard to design an algorithm based on the above. We use find_candicate to find the candidate majority elements, and use check_candidate to verify it.

```
int A[N]; // Set up the initial data of this array
int B[M]; // An extra space to store candidates, where M is at most N/2+1
int N = sizeof(A) / sizeof(A[0]);

int Majority( int A[], int N )
{
    int i, Number_of_Candidates = 0;

    // Check the base case of recursion
    if ( N <= 2) {
       for ( i = 0; i < N ; i++ )
             if ( Check_Candidate( A[i] ) == 1 ) return A[i];
       return 0;
    }

    // Compare two consecutive elements in array A
    for ( i = 0 ; i < N ; i += 2 )
    {
        if ( i+1 < N )              // Does the second element exist?
             if ( A[i] == A[i+1] )
                  B[Number_of_Candidate++] = A[i];
    }
    if ( (N/2)*2 < N ) B[Number_of_Candidates++] = A[N-1];
    return Majority( B, Number_of_Candidates );
}

int Check_Candidate( int Candidate )
{
    int i, Count=0;
    for ( i = 0 ; i < N ; i++ )
          if ( A[i] == Candidate )
               Count++;
    if ( Count > N/2 ) return 1;
    else return 0;
}
```

- Analysis of Method 3:
- Time complexity:
- $T(n)=T(n/2)+o(n)$, use Master Theorem, is $O(n)$
- Space usage: $O(n)$

Ex2: Weighted Selection Problem(P246)

- For n distinct elements x_1, x_2,…,x_n
  - Positive weights $w(x_1)$, $w(x_2)$,…,$w(x_n)$
  - Let W= $\Sigma_{i=1\text{->}n} w(x_i)$
  - Let constant C, 0<C⩽W
  - Find the number x_j so that

$$\sum_{x_i < x_j} w(x_i) < C$$

$$w(x_j) + \sum_{x_i < x_j} w(x_i) \geq C$$

- Solution:
- X={x1,x2,…xn}, W={w1,w2,…wn}
- wSelection(X,C)
  - a=selectionMedian(X) //runs in O(n)
  - X1={xi: xi<a} //runs in O(n)
  - X2={xi: xi>a}
  - m= $\Sigma_{\text{xi in X1}} w(x_i)$
  - If m<C and m+w(a)>C then return a
  - Else if m<C return wSelection(X2,C-m-w(a))
  - Else return wSelection(X1,C)
- Analysis: T(n)=T(n/2)+ o(n), use Master Theorem, is O(n)