Scotty Felch

CSCI 305

4 February 2015

HW #2

1a. When *found* is **true**, then the value for x is

A[low] + A[high] == x.

This is the only way for the control flow of the code to enter the while loop beginning on line 5:

((low < high) AND NOT found)

and the if statement on line 6:

if (A[low] + A[high] == x).

1b. For notation purposes, let A be the original array, and S be set of eliminated possibilities (the two tails of the array).

We can assume that at the start of each iteration, S is still the set of all eliminated possibilities, and *found* == false.

Initiliazation: Low = 0, high = n-1, and S = {null}.

Assuming there is a winning combination in the array, then

P(win) = P(low wins) * P(high wins).

= (1/n) * (1/n-1) = 1/(n(n-1))

Maintenance: Low >= 1, high <= n-2

S = [0 .. low-1] and [high+1 .. n-1], by initial assumption.

For every iteration through this algorithm, while *found* == false, either *low* will increment or *high* will decrement. This grows the size of S, while narrowing down the possibilities remaining in A that could be the winning match. This process continues for each iteration until either we find the match and set found == true, or run out of options and enter the following Termination case.

Termination: Low = high + 1, high = low -1, and S = A.

P(win) = 0, since at this point we've failed.

2a. A = [2, 3, 8, 6, 1].

Inversions = (2,1), (3,1), (8,6), (8,1), (6,1)

2b. What array with elements from the set {1, 2, ..., n} has the most inversions?

A reverse ordered array.

How many inversions does it have?

$\sum_{i=1}^{n} n - i \quad => \quad \frac{n^2+n}{2}$ (by Gaussian identity)

2c. What is the relationship between running time of insertion sort and the number of inversions in the input array?

- $\theta(n^2)$ is worst case of insertion sort.

- Every inversion is an operation that must be performed.

- n inversions translates to $\sum_{i=1}^{n}(n-1)$ operations that must be performed in addition to overhead costs.

- $\sum_{i=1}^{n}(n-1)$ simplifies to $\frac{n^2+n}{2}$.

- This makes sense because $\frac{n^2+n}{2}$ translates to $\theta(n^2)$ when you drop off the low order terms.

2d. An algorithm that determines the number of inversions in any permutation on n elements in $\theta(n \lg n)$ worst case time is a simple modification of the Merge Sort algorithm on p. 31 of the text. The algorithm is already tracking every occurrence of an inversion as part of the sorting process, this occurs when the else block on line 16 is hit. However you can't just add in a counter, since the sorting process intermediate steps would cause some inversions to be not counted.

The two changes two make would be to change lines 16-17 to:

else A[k] = L[i]

j = j + 1

inversion_counter = inversion_counter + 1

3a. I used the problem in textbook on page 121 for reference on this, it is essentially the same format.

X_i = I{you guess correctly}

= {1 if card guess i is correct, 0 if card guess i is incorrect

n = total cards

$X = X_1 + X_2 + ... + X_n$ (X is the total of all trials completed)

E[X] = how many guesses correct in total

$E[X_i]$ = P(i^{th} guess is correct) = $\frac{1}{n}$

$E[X] = E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} \frac{1}{n}$ = $\frac{1}{n}$*n = $\frac{n}{n}$ = 1 correct guess

3b. $E[X_i]$ = P(i^{th} guess is correct) = $\frac{1}{n-i+1}$ because your # of cards to draw from decreases each draw

$E[X] = E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} \frac{1}{n-i+1} = \sum_{i=1}^{n} \frac{1}{i}$ = ln n + O(1)