

Solutions to Practice Problems for Homework 2

1. First the median between 15, 8, and 3 is computed, and swapped into the last element of the subarray, leading to the array: $\langle 15, 7, 9, 3, 4, 10, 8 \rangle$. So 8 will be the pivot element. Next 15 (the first element moving forwards from the left ≥ 8) and 4 (the first element moving backwards from the right < 8) are swapped giving the array: $\langle 4, 7, 9, 3, 15, 10, 8 \rangle$. Next 9 (the next element moving forwards from the left ≥ 8) and 3 (the next element moving backwards from the right < 8) are swapped giving the array: $\langle 4, 7, 3, 9, 15, 10, 8 \rangle$. Finally, the pivot element 8 is swapped with 9 (the leftmost element ≥ 8) to lead to the partitioned array $\langle 4, 7, 3, 8, 15, 10, 9 \rangle$.
2. Suppose that on average each transaction is out of its proper sorted position by at most c positions for some constant c . This means that over all $n - 1$ iterations of insertion sort, the number of comparisons is at most $(c + 1)n$ (with one final comparison used to recognize the element is in order) and at most cn swaps are made. This gives a running time of $\Theta(cn) = \Theta(n)$. Thus, insertion sort is the best choice since mergesort has worst case $\Theta(n \log n)$ time complexity, and quicksort has expected case time complexity $\Theta(n \log n)$.
3. Let the random variable X be the index of the minimum element. X takes on value i (for $i \in \{0, 1, \dots, n - 1\}$) with probability $1/n$. Thus

$$E[X] = \sum_{i=0}^{n-1} i \cdot \text{Probability}(X = i) = \sum_{i=0}^{n-1} \frac{i}{n} = \frac{1}{n}(0 + 1 + \dots + n - 1) = \frac{n(n-1)}{2n} = \frac{n-1}{2}.$$

4. Let the random variable X be the number of times that $\mathbf{A}[i] \neq \mathbf{x}$ is executed.

- (a) Here X can only take on one of 3 values: 1 (occurs with probability $1/2$ when \mathbf{x} is in $\mathbf{A}[0]$), 2 (occurs with probability $1/4$ when \mathbf{x} is in $\mathbf{A}[1]$) and n (occurs with probability $1/4$ when \mathbf{x} is not in \mathbf{A}). Thus

$$E[X] = 1/2 \cdot 1 + 1/4 \cdot 2 + 1/4 \cdot n = n/4 + 1$$

- (b) Here X can take on values between 1 and n each with probability of $1/(2n)$ when the search is successful and with probability of $1/2$ it takes on a value of n (an unsuccessful search). So

$$E[X] = \left[\sum_{i=1}^n \frac{i}{2n} \right] + n \cdot \frac{1}{2} = \frac{1}{2n}(1 + 2 + \dots + n) + n/2 = \frac{1}{2n} \frac{n(n+1)}{2} + \frac{n}{2} = 3n/4 + 1/4$$

- (c) Here when the search is successful, X takes on value i with probability $(1/2)^i$ for $i = 1, \dots, n$ and when the search is unsuccessful, X takes on value n (and this occurs with probability $1/(2^n)$). Thus

$$E[X] = \left[\sum_{i=1}^n i \cdot (1/2)^i \right] + n \cdot \frac{1}{2^n} = 2 - \frac{1}{2^{n-1}} - \frac{n}{2^n} + \frac{n}{2^n} = 2 - \frac{1}{2^{n-1}}$$

5. Let the random variable X be the number of times that `ptr.value != x` is executed.

We have that:

position of item searched for	number times <code>ptr.value != x</code> done	probability
1	1	1/2
2	2	1/4
⋮	⋮	⋮
i	i	$1/(2^i)$
⋮	⋮	⋮
$n - 1$	$n - 1$	$1/(2^{n-1})$
n	n	$1/(2^{n-1})$

So expected number of times `ptr.value != x` is executed is $\left(\sum_{i=1}^{n-1} i/(2^i)\right) + n/(2^{n-1})$.

Applying formula that $\sum_{i=1}^k \frac{i}{2^i} = 2 - \frac{1}{2^{k-1}} - \frac{k}{2^k}$ with $k = n - 1$ yields that expected number of times `ptr.value != x` is executed is:

$$2 - \frac{1}{2^{n-2}} - \frac{n-1}{2^{n-1}} + \frac{n}{2^{n-1}} = 2 - \frac{2}{2^{n-1}} + \frac{1}{2^{n-1}} = 2 - \frac{1}{2^{n-1}}$$

So you on average you the number of list items traversed approaches 2 as n approaches infinity.

6. The adversary will begin by creating the following $n + 1$ possibilities for the array A . First for each $i \in \{0, 1, \dots, n - 1\}$ a sorted array is created in which $A[i] = x$ (and all other elements are different than x). Finally, a sorted array is created that does not contain x . Let L be a list holding these $n + 1$ possible values for A . For example, when $n = 6$ and $x = 10$ the list L could contain the 7 possible inputs:

$$\langle 10, 11, 12, 13, 14, 15 \rangle, \langle 9, 10, 11, 12, 13, 14 \rangle, \langle 8, 9, 10, 11, 12, 13 \rangle, \langle 7, 8, 9, 10, 11, 12 \rangle, \\ \langle 6, 7, 8, 9, 10, 11 \rangle, \langle 5, 6, 7, 8, 9, 10 \rangle, \langle 4, 5, 6, 7, 8, 9 \rangle.$$

Whenever the algorithm asks to compare x to $A[i]$, the adversary examines all arrays in L and counts how many have $x < A[i]$, how many have $x = A[i]$, and how many have $x > A[i]$. It will answer in accordance to the answer that occurs most frequently (with ties broken arbitrarily). Observe that $x = A[i]$ could only occur once. Thus the most frequent answer must occur in at least $\lceil (n - 1)/2 \rceil$ of the arrays in L . From this it follows that the adversary can guarantee that $|L| \geq 2$ until $\lceil \log_2(n + 1) \rceil$ comparisons have been made. Since a correct algorithm cannot be done until $|L| = 1$, it follows that any correct algorithm for this problem must execute at least $\lceil \log_2(n + 1) \rceil$ statements leading to a $\Omega(\log n)$ lower bound.

Note: A slightly simpler argument argues that the most frequent answer occurs in at least $\lceil |L|/3 \rceil$ of the arrays in L . This leads to a weaker (though asymptotically equivalent) lower bound of $\lceil \log_3 n \rceil$.

7. An array can be placed into the list L for each distinct answer. Observe that each answer can be viewed as a string that is a sequence (possibly empty) of “1”s corresponding to elements less than x and y (inclusive), followed by a sequence (possibly empty) of “2”s corresponding to elements between x and y , and finally a sequence (possibly empty) of “3”s corresponding to elements greater than x and y . Each such bit string is defined by the positions where you switch from 1 to 2 and where you switch from 2 to 3. Hence we must count the number of ways to arrange 2 dividers with n array elements. Equivalently, we must count the number of ways to position 2 dividers among $(n + 2)$ slots. Hence the adversary can create a list L of $C(n + 2, 2) = (n + 2)(n + 1)/2$ inputs with distinct answers among with the algorithm must force the adversary to eliminate all but one.

The adversary answers each question based on the majority of the elements in L . So with each comparison the adversary can guarantee that at least $|L|/2$ of these possible inputs remain. Thus the number of comparisons that the adversary can force any algorithm to make before $|L| = 1$ is $\lceil \log_2(n + 2)(n + 1)/2 \rceil = \lceil \log_2(n + 2) + \log_2(n + 1) - 1 \rceil$. So any comparison-based algorithm for this problem takes $\Omega(\log n)$ time.