

Practice Problems for Homework 2

1. Illustrate the operation of the `Partition` algorithm on the array: $\langle 15, 7, 9, 8, 4, 10, 3 \rangle$ when using the median-of-three partitioning. Indicate where the swaps occur during the algorithm's execution and show the state of the array after each swap.
2. Banks often record transactions on an account in the order of the times of the transactions, but many people like to receive their bank statements with checks listed in order by check number. People usually write checks in order by check number, and merchants usually cash them reasonably promptly. The problem of converting time-of-transaction ordering to check-number ordering is therefore the problem of sorting almost-sorted input. Between insertion sort, mergesort and quicksort explain which one would be best for this problem.
3. An array $A[0 \dots n - 1]$ contains n distinct numbers that are randomly ordered, with each permutation of the n numbers being equally likely. What is the expectation of the index of the minimum element in the array?
4. Consider the problem of searching for x in an array A of n elements:

```
boolean search(n,A,x){  \searches for x in the unsorted array A[0..n-1]
    int i=0;
    while (i <= n-1 && A[i] != x)  \searches for x
        i++;
    if (A[i] == x) return true;
    else return false;
```

You are to give an exact answer for number of times that $A[i] \neq x$ is executed.

Along with the arithmetic sum $\sum_{i=1}^k i = k(k+1)/2$, another useful summation (obtained by integrating both sides of the geometric series $\sum_{i=0}^k x^i = (1-x^{k+1})/(1-x)$ and then setting $x = 1/2$ is

$$\sum_{i=0}^k i(1/2)^i = \sum_{i=1}^k i(1/2)^i = 2 - \frac{1}{2^{k-1}} - \frac{k}{2^k}$$

- (a) Assume that with probability $1/2$, x is in $A[0]$, with probability $1/4$, x is in $A[1]$ and with probability $1/4$, x is not in A .
- (b) Assume that with probability $1/2$, x is not in A and with probability $1/(2n)$, x is position i of the list for $i = 0, 1, \dots, n - 1$.
- (c) Assume that for $i = 0, 1, \dots, n - 1$, that the probability that x is in position i of the array is $(1/2)^{i+1}$ and with probability $(1/2)^n$ that x is not in the array.

5. Here we consider a situation in which calls will be repeatedly made to search for an item in a unsorted list L . Further, you know that items searched for recently are more likely to be searched for again. Thus, to reduce the search time, whenever an item is found it is moved to the front of L . Here's pseudo-code for the search procedure where `ptr.value` gives the value of the item pointed to by `ptr` and `ptr.next` is a reference to the list item after that referenced by `ptr`.

```

boolean search(L,x){           \searches for x in the unsorted list L
  initialize ptr to be a reference to the first item in L
  while (ptr != nil && ptr.value != x)           \searches for x
    ptr = ptr.next;
  if (ptr != nil) {           \if found
    move item referenced by ptr to the front of L
    return true;
  }
  else
    return false;
}

```

You are given that for $1 \leq i \leq n - 1$, that the probability that x is in position i of the list is $(1/2)^i$ and the probability that x is in position n is $(1/2)^{n-1}$. (The first item in the list is item 1, the second item in the list is item 2, and so on with the last item in the list being item n .)

What is the expected number of times that the comparison (`ptr.value != x`) is executed? You are to give an EXACT answer.

Note: Be sure to show your work starting from the definition of expected value.

6. Use the adversary lower bound technique to prove a lower bound on the time complexity of the problem of searching if x is in a sorted array A (of n elements) when using a comparison-based algorithm.
7. Use the adversary lower bound technique to prove a lower bound on the time complexity of a comparison-based algorithm for the following problem: You are given a sorted array A (of n elements) and two elements x and y where $x \leq y$. The algorithm is required to compute how many elements in A are less than both x and y , how many elements of A are between x and y (inclusive), and how many elements of A are bigger than both x and y . Note that x and y are not necessarily in A .