

Homework 2

February 4, 1999

Due Date: February 11

Write up all solutions clearly, concisely, and legibly. Don't forget about the practice problems (with solutions) on the course web page (under Homeworks).

Required Problems:

1. (7 pts) In the last homework you used the definitions of O , Ω , and Θ to prove or disprove statements. To be sure that everyone understands these notations, indicate whether each of the following is true or false. You are to assume that for all n , $f(n) \geq 0$ and $g(n) \geq 0$, and that both $f(n)$ and $g(n)$ cannot decrease as n increases. In other words, assume that $f(n)$ and $g(n)$ are time complexities from algorithms where n is the input size.

Give a one or two sentence explanation. No proofs are required here. However, I recommend you think about how you would prove your answer is correct since I may ask you to do this on Exam 1.

- (a) $5n + 2 = O(n)$
- (b) $5n + 2 = O(n^2)$
- (c) $\frac{1}{2}n^2 + n \log_2 n = \Omega(n \log_2 n)$.
- (d) $2n^2 + 7n = \Theta(n^2)$
- (e) $3n^2 + (n \log n)/2 = \Omega(n \log n)$
- (f) if $f(n) = O(g(n))$ then $f(n) + g(n) = \Omega(f(n))$
- (g) if $f(n) = \Omega(g(n))$ then $f(n) + g(n) = \Theta(f(n))$
- (h) if $f(n) = \Theta(g(n))$ then $2^{f(n)} = \Theta(2^{g(n)})$
- (i) if $f(n) = \Omega(g(n))$ then $f(n) + g(n) = O(g(n))$

2. (8 pts) Suppose that you are to design an algorithm to find the median of n numbers. You have thought of 5 divide and conquer algorithms that have time complexities as given by the following recurrences (for each, $T(1) = \Theta(1)$). Which algorithm(s) are fastest as n grows large? Clearly justify your answer.

Algorithm A: $T(n) = 4T(n/2) + \Theta(1)$

Algorithm B: $T(n) = 4T(n/4) + \Theta(n)$

Algorithm C: $T(n) = 3T(n/2) + \Theta(\log n)$

Algorithm D: $T(n) = T(9n/10) + \Theta(n)$

Algorithm E: $T(n) = 2T(n/2) + \Theta(n \log n)$.

3. (10 pts) Derive an exact solution to the following recurrence AND use induction to prove your solution is correct. (If you do not know induction, you can get partial credit by showing that your solution is correct for $n = 1, 2, 3, 4$ and 5 .)

$$T(1) = 1, \quad T(n) = T(n-1) + cn \text{ for } n \geq 2$$

4. (10 pts) Consider the following sorting algorithm:

```

void sort(A,p,r){                \\ sorts A[p..r]
  if (p < r) {
    for (i = p; i<= r-1; i++){   \\ has the loop invariant that for
      if (A[i] > A[r])           \\ p <= j <= i-1, A[j] <= A[r]
        swap A[i] and A[r]
    }
    sort(A,p,r-1)
  }
}

```

- (a) If you were to make the call `Sort(A,0,A.length-1)` would `Sort` succeed in sorting `A`? If not, give an input `A` for which it would fail. If it is correct then provide a brief proof of correctness.
- (b) What is the asymptotic time complexity of `Sort` as a function of n where n is the number of items in array `A`? Show your work.
5. (15 pts) In this problem you will design and analyze a divide-and-conquer algorithm to find the sum of the elements of an array `A` of size n . You are REQUIRED to divide the input into two subproblems each of roughly $n/2$ elements.
- (a) Give detailed *pseudo-code* for your algorithm (i.e. roughly the level of detail provided in lecture for quicksort). You may find it useful for your procedure to take as input the array `A` and indices `p` and `r` and return the sum of $A[p] + \dots + A[r]$. Hence, your initial call would have $p = 0$ and $r = n - 1$.
- (b) Give a recurrence expressing the time complexity of your algorithm and then solve it to obtain the asymptotic time complexity of your procedure as a function of n .
6. (15 pts) 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 $\frac{1}{2^i}$ and the probability that x is in position n is $\frac{1}{2^{n-1}}$. (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.

Challenge Problems:

Only work on this after you have completed the required problems.

7* (5 pts) The following program determines the maximum value in an unordered array $A[0..n-1]$.

```
1  $max = -\infty$ 
2 (int  $i = 0; i \leq n - 1; i++$ )
3     if  $A[i] > max$                 ▷ Compare  $A[i]$  to  $max$ 
4     then  $max = A[i]$ 
```

What is the expected number of times the assignment in line 4 is executed? (Assume that all numbers in A are drawn randomly from the interval $[0,1]$.) You may find it useful to let s_1, s_2, \dots, s_n be n random variables, where s_i represents the number of times (0 or 1) that line 4 is executed during the i th iteration of the **for** loop. Since $s = s_1 + s_2 + \dots + s_n$, by linearity of expectations $E(s) = E(s_1) + \dots + E(s_n)$.