CS 331: Algorithms and Complexity (Spring 2024)
Unique Number: 50930, 50935 50940, 50945

Assignment 1

Due on Thursday, 25 January, by 11.59pm

Problem 1

(a) (3 points)
$O(f_1(n)) = n$
$O(f_2(n)) = n\log\log n$
$O(f_3(n)) = n^2$
We need to find whether $\sqrt{n}$ *or* $\log(n^2)$ *grows faster.*
Solve: $\lim_{n\to\infty} \frac{\sqrt{n}}{\log(n^2)} = \lim_{n\to\infty} \frac{d}{dn}\frac{\sqrt{n}}{\log(n^2)} = \lim_{n\to\infty} \frac{1}{2\sqrt{n}}/\frac{2n}{n^2}$ *(Constants don't matter), this approaches* ∞ *as n increases,* $\therefore$
$O(f_4(n)) = \sqrt{n}$
$O(f_5(n)) = 2^{\sqrt{n}}$
$O(f_6(n)) = 2^{\log n}$
$\therefore$ *the order is:* $f_4(n), f_1(n), f_2(n), f_3(n), f_6(n), f_5(n)$

(b) (3 points)
Given $a_1 = 1$, $a_2 = 8$, *and* $a_n = a_{n-1} + 2a_{n-2}$ *when* $n \geq 3$, *show* $a_n = 3 \cdot 2^{n-1} + 2(-1)^n$ *for all* $n \in \mathbb{N}$.
Define our predicate as $P(n) = a_n = 3 \cdot 2^{n-1} + 2(-1)^n$
Base Cases:

Proof. Let n = 1, then $a_1 = 1$, and $3 \cdot 2^{1-1} + 2(-1)^1 = 1$, which is valid, $\therefore$ P(1) is true.
Let n = 2, then $a_2 = 8$, and $3 \cdot 2^{2-1} + 2(-1)^2 = 8$, which is valid, $\therefore$ P(2) is true. □

Inductive Hypothesis: *Assume for n = k, P(1), P(2), …, P(n) are all true.*

Inductive Step:
Show P(k + 1), aka $a_{k+1} = 3 \cdot 2^k + 2(-1)^{k+1}$.

Proof. We are given this equation: $a_{k+1} = a_k + 2a_{k-1}$.
We can substitute our inductive hypothesis into this equation to get: $a_{k+1} = (3 \cdot 2^{k-1} + 2(-1)^k) + 2(3 \cdot 2^{k-2} + 2(-1)^{k-1})$.
We can factor out a two from the first group to get $2(3 \cdot 2^{k-2} + 2(-1)^{k-1}) + 2(3 \cdot 2^{k-2} + 2(-1)^{k-1})$.

Combine the two groups to get $4(3 \cdot 2^{k-2} + 2(-1)^{k-1})$.
Since $4 = 2^2$, we can add 2 to each exponent to get $3 \cdot 2^k + 2(-1)^{k+1}$.
∴, we have shown that P(k + 1) is true, and by induction, P(n) is true for all $n \in \mathbb{N}$ □

(c) (4 points)
`Algorithm 1(Alice)`

Proof. We need to swap the lines `Report average as sum / count;` and `Increase count by 1;` as in the first iteration, we will divide by 0, which is not the correct average.

□

`Algorithm 2(Bob)`

Proof. First, we show termination.
Since each iteration we read one integer, the set of integers needed to be read is decreased by one each iteration, and since the set is finite, the algorithm will terminate.
Next, we show correctness.
Assume we have to read n integers from the input stream, which we'll call S.
Base Case: n = 1
average is updated to $\frac{(0 \cdot 0 + S_1)}{0+1} = S_1$, which is the correct average, and count is incremented by one which results in count=1.
Then we have average=S_1 and count=1, which is correct.
Inductive Hypothesis: Assume for n = k, average=$\frac{\sum_{i=1}^{k} S_i}{count}$ and count=k.
Inductive Step: Show for n = k + 1, average=$\frac{\sum_{i=1}^{k+1} S_i}{count+1}$ and count=k + 1.
In the loop, we read the integer S_{k+1} and update average to $\frac{(average * count) + S_{k+1}}{count+1}$.
We have to show that $\sum_{i=1}^{k+1} S_i$ = (average * count) + S_{k+1}.
We know that average=$\frac{\sum_{i=1}^{k} S_i}{count}$, ∴ average * count = $\sum_{i=1}^{k} S_i$.
Adding S_{k+1}, we get $\sum_{i=1}^{k+1} S_i$ = (average * count) + S_{k+1}.
∴, we can express $\frac{(average * count) + S_{k+1}}{count+1}$ as $\frac{\sum_{i=1}^{k+1} S_i}{count+1}$, which is the correct average.
count is then incremented by one, which results in count=k+1.
∴, we have shown that the algorithm is correct.

□

Problem 2

(6 points)
For this problem, I would use binary search, since I know that the array is sorted in ascending order.

```
int low = 0, high = n - 1;
while (low <= high) \{
    int mid = low + (high - low) / 2;
    if (E[mid] == x) return mid;
    else if (A[mid] < x) low = mid + 1;
```

```
            else high = mid - 1;
        }
```

Proof. This algorithm terminates since the range of the search is halved each iteration.
Correctness: We are trying to find x $<$ max in E.
Since the array is constructed as [**sorted**|$\mathbf{max}_{size-n}$], and max is greater than all elements of the array up to n, then there are 3 cases in each ith iteration:
Case 1: x $<$ E[i], then since the array is sorted, x is the lower half of the search space.
Case 2: x $>$ E[i], then since the array is sorted, x is the upper half of the search space.
Case 3: x = E[i], then we have found the index of x.

□

Problem 3

Your task is to do the following:

i **(7 points)**

ii **(7 points)**

Proof. □