CSE 21
Practice Exam for Midterm 1
Winter 2016

This practice exam should help prepare you for the first midterm on Friday, January 29.

1. Sorting and Searching Give the number of comparisons that will be performed by each sorting algorithm if the input array of length n happens to be of the form $[1, 2, ..., n-3, n-2, n, n-1]$ (i.e., sorted except for the last two elements).

Note: On the real exam, you would be given pseudocode for the algorithms, though it is a very good idea to be comfortable with how the algorithms work to save time on the exam. For now, you can refer to the textbook and/or lecture slides for pseudocode.

(a) MinSort (SelectionSort)
Solution: $\frac{n(n-1)}{2}$
MinSort does the same amount of work on all inputs. It does $n-1$ comparisons to find the minimum of the list before swapping it to the front. Then it does $n-2$ comparisons on the next pass to find the minimum of the remaining elements, and so on, until it eventually does 1 comparison to find the minimum of the last remaining two elements. The total number of comparisons is $1+2+\cdots+(n-1)$.

(b) BubbleSort
Solution: $\frac{n(n-1)}{2}$
BubbleSort does the same amount of work on all inputs. It does $n-1$ comparisons on the first pass, then $n-2$ comparisons on the next pass, and so on. The total number of comparisons is $1+2+\cdots+(n-1)$.

(c) InsertionSort
Solution: $\frac{n^2+n-4}{2}$
On this input, InsertionSort will do two comparisons when $j=2$ (comparing a_2 with a_1 and then with itself). It will then do three comparisons when $j=3$, and so on, up to $n-1$ comparisons when $j=n-1$. This is because each new number a_j being inserted into the sorted list is greater than all the numbers that have been inserted before it, and InsertionSort compares from the beginning of the list each time. Finally, when $j=n$, InsertionSort will do $n-1$ comparisons since a_n is bigger than all but one of the elements that have been inserted before it. It takes $n-1$ comparisons (comparing a_n with all the other elements except itself) in order to determine to place a_n in the second to last position. The total number of comparisons is $2+3+\cdots+(n-1)+(n-1)$. We can simplify this sum to $\frac{n^2+n-4}{2}$. Note that this is just one less comparison than the worst case of InsertionSort, which would be if all the list elements were in sorted order. The analysis for the worst case of InsertionSort is done in Example 6 on page 222 of Rosen.

2. Asymptotic Notation For each part, answer True or False, and give a short explanation for your answer. All logarithms are base 2.

(a) $\sqrt{n^3} \in O(n^2)$.

Solution: True. As $n \to \infty$, the limit of $\frac{\sqrt{n^3}}{n^2}$ is 0, which says $\sqrt{n^3} \in O(n^2)$.

(b) $8^{\log(n^2)} \in \Theta(n^6)$.

Solution: True. $8^{\log(n^2)} = (2^3)^{\log(n^2)} = 2^{3\log(n^2)} = 2^{\log(n^2)^3} = 2^{\log(n^6)} = n^6$. Since both functions are the same, they are in the same Θ class.

(c) $\log(n) \in \Omega(\log(\log(n)))$.
Solution: True. By the definition of Ω, $\log(n) \geq 1 \cdot \log(\log(n))$ for all $n > 2$.

(d) If f, g, and h are functions from the natural numbers to the non-negative real numbers with $f(n) \geq g(n)\ \forall n \geq 1$, $f(n) \in \Theta(h(n))$, and $g(n) \in \Theta(h(n))$, then $(f-g)(n)) \in \Theta(h(n))$.
Solution: False. Here is a counterexample: $f(n) = 3n, g(n) = 3n, h(n) = n$.

(e) If f, g, and h are functions from the natural numbers to the non-negative real numbers with $f(n) \in \Theta(h(n))$ and $g(n) \in \Theta(h(n))$, then $(f * g)(n)) \in \Theta((h(n))^2)$.

Solution: True. Since $f(n) \in \Theta(h(n))$ and $g(n) \in \Theta(h(n))$, that means we can find constants $c_1, d_1, k_1, c_2, d_2, k_2$ so that $d_1 \cdot h(n) \leq f(n) \leq c_1 \cdot h(n)$ for all $n > k_1$ and $d_2 \cdot h(n) \leq g(n) \leq c_2 \cdot h(n)$ for all $n > k_2$. If we multiply these inequalities, this means that for all $n > max(k_1, k_2)$, $d_1 d_2 \cdot (h(n))^2 \leq (f * g)(n) \leq c_1 c_2 \cdot (h(n))^2$, which says $(f * g)(n)) \in \Theta((h(n))^2)$.

3. Best and Worst Case Suppose we are adding two n-digit integers, using the usual algorithm learned in grade school. The primary operation here is the number of single-digit additions that must be performed. For example, to add 48 plus 34, we would do three single-digit additions:

1. In the ones place, add $8 + 4 = 12$.
2. In the tens place, add $4 + 3 = 7$.
3. In the tens place, add $7 + 1 = 8$.

(a) If $n = 5$, give an example of two n-digit numbers that would be a best-case input to the addition algorithm, in the sense that they would cause the fewest single-digit additions possible.

Solution: One example is 11111 plus 22222.

(b) In the best case, how many single-digit additions does this algorithm make when adding two n-digit numbers?

Solution: In each column, there is at least one addition, and the best case is when there is only this one addition (no carrying). Therefore the total number of single-digit additions could be as low as the number of columns, or n .

(c) In the best case, when adding two n-digit numbers, describe the number of single-digit additions in Θ notation.

Solution: n is $\Theta(n)$, so the best case is linear in the number of digits.

(d) If $n = 5$, give an example of two n-digit numbers that would be a worst-case input to the addition algorithm, in the sense that they would cause the most single-digit additions possible.

Solution: One example is 99999 plus 88888.

(e) In the worst case, how many single-digit additions does this algorithm make when adding two n-digit numbers?

Solution: In the rightmost column, there are two numbers to add, so that is one single-digit addition. In all of the other $n - 1$ columns, there could be 3 numbers to add (from carrying), so that is two single-digit additions each. Therefore the total number of single-digit additions could be as high as $1 + 2(n - 1) = 2n - 1$.

(f) In the worst case, when adding two n-digit numbers, describe the number of single-digit additions in Θ notation.

Solution: $2n - 1$ is $\Theta(n)$, so the worst case is linear in the number of digits.

4. Iterative Algorithms and Loop Invariants In the following problem, we are given a list $A = a_1, \ldots, a_n$ of salaries of employees at our firm and two integers L and H with $0 \leq L \leq H$. We wish to compute the average salary of employees who earn between L and H (inclusive), and the number of such employees. If there are no employees in the range, we say that 0 is the average salary. This is an iterative algorithm which takes as input A, L, and H and returns an ordered pair (avg, N) where avg is the average salary of employees in the range, and N is the number of employees in the range.

AverageInRange(A : list of n integers, L, H : integers with $0 \leq L \leq H$)

1. $sum := 0$
2. $N := 0$
3. **for** $i := 1$ to n
4. **if** $L \leq a_i \leq H$ **then**

5. $sum := sum + a_i$
6. $N++$
7. **if** $N = 0$ **then**
8. **return** $(0, 0)$
9. **return** $(sum/N, N)$

(a) State a loop invariant that can be used to show the algorithm *AverageInRange* is correct.

Solution: After t times through the for loop, *sum* is the total salary of those people among $a_1, \ldots, a_t$ who make between L and H, and N is the number of such people.

(b) Prove your loop invariant from part (a).

Solution: For the base case, let $t = 0$. Before the loop starts, *sum* and N are both 0. This is correct because there are no people among $a_1, \ldots, a_t$, as it is the empty set, so N should be 0, and we said that when there are no employees in the range, their average salary is 0.

Now suppose that after t times through the for loop, *sum* is the total salary of those people among $a_1, \ldots, a_t$ who make between L and H, and N is the number of such people.

On the $t + 1$st time through the loop, $i = t + 1$.

Case 1:

If a_{t+1} falls in the range from L to H, then we increment *sum* by a_{t+1} and N by 1. Thus, if the value of *sum* before this iteration was the total salary of those people among $a_1, \ldots, a_t$ who make between L and H, now it is the total salary of those people among $a_1, \ldots, a_{t+1}$ who make between L and H. Similarly, if N before this iteration counted the total number of people among $a_1, \ldots, a_t$ with salaries in the range of L to H and we increased it by one, now N counts the number of such people among $a_1, \ldots, a_{t+1}$.

Case 2:

If a_{t+1} does not fall in the range of L to H, the algorithm does nothing. In this case, the total salary of those people among $a_1, \ldots, a_t$ who make between L and H is the same as the total salary of those people among $a_1, \ldots, a_{t+1}$ who make between L and H. Similarly, the number of such people among $a_1, \ldots, a_t$ is the same as the number of such people among $a_1, \ldots, a_{t+1}$.

(c) Conclude from the loop invariant that the algorithm *AverageInRange* is correct.

Solution: Letting $t = n$ in the loop invariant says that after the algorithm terminates, *sum* is the total salary of those people among $a_1, \ldots, a_n$ who make between L and H, and N is the number of such people. The algorithm returns $(sum/N, N)$, which is the correct average salary and number of people.

(d) Describe the running time of this algorithm in Θ notation, assuming that comparisons and arithmetic operations take constant time. Justify your answer.

Solution: This is a linear time algorithm, $\Theta(n)$. The operations inside the for loop take constant time, and the for loop runs n times. All other work outside the loop is also constant time.

5. Recursive Algorithms In the following problem, we are given a list $A = a_1, \ldots, a_n$ of salaries of employees at our firm and two integers L and H with $0 \leq L \leq H$. We wish to compute the average salary of employees who earn between L and H (inclusive), and the number of such employees. If there are no employees in the range, we say that 0 is the average salary. This is a recursive algorithm which takes as input A, L, and H and returns an ordered pair (avg, N) where avg is the average salary of employees in the range, and N is the number of employees in the range.

$RecAIR(A$: list of n integers, L, H : integers with $0 \leq L \leq H)$

1. **if** $n = 0$ **then**
2. **return** $(0, 0)$
3. $B := a_1, a_2, \ldots, a_{n-1}$
4. $(avg, N) := RecAIR(B, L, H)$
5. **if** $L \leq a_n \leq H$ **then**

6. **return** $((avg * N + a_n)/(N+1), N+1)$
7. **else**
8. **return** (avg, N)

(a) Prove by induction on n that for any input, the algorithm correctly returns the average salary and number of employees in the range.

Solution: For the base case, if $n = 0$, the algorithm returns 0 for the average and 0 for the number of employees in the range. Both are correct, because we said that when there are no employees in the range, the average salary is 0.

Now suppose the algorithm returns the correct average salary avg and number of employees N when there are $n-1$ employees. When there are n employees, either the nth employee is in the salary range from L to H or not.

Case 1:

If a_n falls in the the salary range L to H, the algorithm returns the total number of employees in the range as $N+1$. Assuming by the inductive hypothesis that N of the first $n-1$ employees are in the range, then this is correct. Also, the algorithm returns for the average $(avg * N + a_n)/(N+1)$ in this case. Assuming by the inductive hypothesis that avg is the average salary of the first $n-1$ employees in the range, and N is the number of such employees, then the total salary of all of the first n employees who fall in the range is $avg * N + a_n$ and the number of such employees is $N+1$, so the average salary comes from dividing these numbers.

Case 2:

If a_n does not fall in the the salary range L to H, then the number of employees among the first n who are in the range is the same as the number of employees among the first $n-1$ who are in the range. By the inductive hypothesis, we assume this number is N, which is what the algorithm returns in this case for the number. Also, the algorithm returns for the average avg, which is also correct by the inductive hypothesis. This is true because if a_n is not in the range, the average salary among the first $n-1$ employees in the range is the same as the average salary among the first n employees in the range, and this number is avg.

(b) Write down a recurrence for the time taken by this algorithm, assuming that comparisons and arithmetic operations take constant time. Assume also that removing an element from a list (line 3) takes constant time.

Solution: $T(0) = c,\ T(n) = T(n-1) + d$, where c and d are constants

(c) Use your answer from part (b) to determine the running time of this algorithm in Θ notation. Justify your answer mathematically.

Solution: We can do this by unraveling or by guess and check. Let's do it by unraveling.

$$\begin{aligned} T(n) &= T(n-1) + d \\ &= T(n-2) + 2d \\ &= T(n-3) + 3d \\ &\vdots \\ &= T(n-k) + kd \\ &\vdots \\ &= T(0) + nd \\ &= c + nd \end{aligned}$$

We choose a value of $k = n$ to end at the base case of $T(0)$.

The result is a linear function in n, $c + dn$, so the algorithm is $\Theta(n)$. Notice that it's the same order as the iterative version in the previous problem, so neither is substantially more efficient than the other.

(d) Write down a recurrence for the time taken by this algorithm, assuming that comparisons and arithmetic operations take constant time. Assume now that removing an element from a list (line 3) takes linear time.

Solution: $T(0) = c,\ T(n) = T(n-1) + dn$, where c and d are constants

(e) Use your answer from part (d) to determine the running time of this algorithm in Θ notation. Justify your answer mathematically.

Solution: We can again do this by unraveling or by guess and check. Let's do it by unraveling.

$$\begin{aligned}
T(n) &= T(n-1) + dn \\
&= T(n-2) + d(n-1) + dn \\
&= T(n-3) + d(n-2) + d(n-1) + dn \\
&\vdots \\
&= T(n-k) + d(n-k+1) + d(n-k+2) + \cdots + d(n-1) + dn \\
&\vdots \\
&= T(0) + d(1 + 2 + \cdots + (n-1) + n) \\
&= c + dn(n+1)/2
\end{aligned}$$

We choose a value of $k = n$ to end at the base case of $T(0)$.

The result is a quadratic function in n, since the higest power if we expand $c + dn(n+1)/2$ is n^2, so the algorithm is $\Theta(n^2)$.

6. Solving Recurrences Suppose a function f is defined by the following recursive formula, where n is a positive integer.

$$f(n) = f(n-1) + 2n - 1,\ f(1) = 6$$

Use any method we learned in this class to get a closed-form formula for $f(n)$.

Solution: By the unraveling method, we get

$$\begin{aligned}
f(n) &= f(n-1) + 2n - 1 \\
&= f(n-2) + 2(n-1) - 1 + 2n - 1 \\
&= f(n-3) + 2(n-2) - 1 + 2(n-1) - 1 + 2n - 1 \\
&\vdots \\
&= f(n-k) + 2(n-k+1) + 2(n-k+2) + \cdots + 2(n-1) + 2n - k \\
&\vdots \\
&= f(1) + 2(2 + \cdots + (n-1) + n) - (n-1) \\
&= 6 + 2(n(n+1)/2 - 1) - n + 1 \\
&= 6 + n(n+1) - 2 - n + 1 \\
&= n^2 + 5.
\end{aligned}$$

Here we are using $k = n - 1$ so that we reach the base case of $f(1)$. Our closed-form formula is $f(n) = n^2 + 5$.