CMPS-2200 Algorithms – Fall 24

9/24/24

Practice Midterm

Name: _____

Student ID: _____

- Put your name on the exam.
- You are allowed to bring limited notes (5 pages) and a calculator.
- You are not allowed to use the internet.
- Explain your reasoning in every answer.
- Logs are base 2 when the subscript are omitted.

1) Asymptotic ranking		10
2) Big Oh		10
3) Recurrence		10
4) Turing Machines		15
5) Finite State Machines		10
6) Divide and Conquer		15
7) Sorting		15
8) Map/Filter/Reduce		10
9) P vs NP (could be different on exam)		5
Bonus		10
Total		100

1 Asymptotic Ranking

List the following functions from smallest to biggest asymptotically. Indicate which pairs of functions are asymptotically equal.

$$n, \qquad \frac{n}{\log(n)}, \qquad \log(\log(n)+n), \qquad 2^n,$$
$$1.01^n, \qquad n^{1.01}, \qquad \frac{4n^2-3}{n+9}, \qquad 14, \qquad \log(n!)$$

Solution:

$$14, \log(\log(n)+n), \frac{n}{\log(n)}, n = \frac{4n^2}{n+9}, \log(n!), n^{1.01}, (1.01)^n, 2^n$$

2 Big Oh

1. Use the definition of Θ to prove that if $f, g, h : \mathbb{N} \to \mathbb{N}$ satisfy that $f \in \Theta(g)$ and $g \in \Theta(h)$ then $f \in \Theta(h)$.

   Solution: Since $f \in \Theta(g)$, there exists $n_f \in \mathbb{N}$ and constants $c_1, c_2 \in \mathbb{R}_{>0}$ such that for all $n > n_f$, $c_1 g(n) \leq f(n) \leq c_2 g(n)$.

   Since $g \in \Theta(h)$, there exists $n_h \in \mathbb{N}$ and constants $d_1, d_2 \in \mathbb{R}_{>0}$ such that for all $n > n_g$, $d_1 h(n) \leq g(n) \leq d_2 h(n)$. When $n > \max(n_f, n_g)$, all inequalities hold. We can combine them using the transitive property for inequalities to conclude that

   $$c_1 d_1 h(n) \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \leq c_2 d_2 h(n).$$

   Since $c_1 d_1$ and $c_2 d_2$ are in $\mathbb{R}_{>0}$ and $\max(n_f, n_g) \in \mathbb{N}$, this shows that $f \in \Theta(h)$.

2. Give a counterexample or prove the following claim: If $a(n) \in \Theta(f(n))$ and $b(n) \in \Theta(g(n))$, then $a(n)b(n) \in \Theta(f(n)g(n))$.

   Solution: Since $a \in \Theta(f)$, there exists $n_a \in \mathbb{N}$ and $c_1, c_2 \in \mathbb{R}_{>0}$ such that for all $n \geq n_a$ such that $c_1 f(n) \leq a(n) \leq c_2 f(n)$. Similarly, there exists $n_b \in \mathbb{N}$ and $d_1, d_2 \in \mathbb{R}_{>0}$ such that for all $n > n_b$, $d_1 g(n) \leq b(n) \leq d_2 g(n)$. If we multiply these two inequalities (noting that all terms are positive) we find that for $n \geq \max(n_a, n_b)$, we have $c_1 d_1 f(n)g(n) \leq a(n)b(n) \leq c_2 d_2 f(n)g(n)$. Since $c_1 d_1, c_2 d_2 \in \mathbb{R}_{>0}$, this shows that $a(n)b(n) \in \Theta(f(n)(n))$.

3. Use limits to determine the asymptotic relationship between $2^{\sqrt{n}}$ and $2^{\frac{n}{\log(n)}}$. Show your work.

   Solution: We use the limit theorem and L'Hôpital's rule. We treat $\log(n) = \log_2(n)$ like the natural log, since it makes differentiating easier and is only different by a constant factor.

$$\begin{aligned}
\lim_{n\to\infty} \frac{2^{\sqrt{n}}}{2^{\frac{n}{\log(n)}}} &= \lim_{n\to\infty} 2^{\sqrt{n}-\frac{n}{\log(n)}} \\
&= 2^{\lim_{n\to\infty} \sqrt{n}-\frac{n}{\log(n)}} \\
&= 2^{\lim_{n\to\infty} \frac{\sqrt{n}\log(n)-n}{\log(n)}} \\
&= 2^{\lim_{n\to\infty} \frac{\frac{\log(n)}{2\sqrt{n}}+\frac{1}{\sqrt{n}}-1}{\frac{1}{n}}} \\
&= 2^{\lim_{n\to\infty} \frac{1}{2}\sqrt{n}\log(n)+\sqrt{n}-n} \\
&= 2^{-\infty} \\
&= 0
\end{aligned}$$

So $2^{\sqrt{n}} \in o(2^{n\log(n)})$.

Note: You could skip from the second line to the last two.

Note: The questions on this section should be more in-depth about how asymptotic notation is defined. It could be about little oh, little omega, etc. I could give two slightly different formulations of the definitions and ask if the differences are significant.

3 Recurrence

Solve the following asymptotic recurrences by any method.

- $W_A(n) = 3W_A(\frac{n}{2}) + n^3$
- $W_B(n) = 2W_B(\frac{n}{\sqrt{2}}) + n^2$
- $W_C(n) = 5W_C(\frac{n}{25}) + 1$

Solution: We use the brick method, which is also the Master Theorem.

1. This recurrence is root-dominated, so $W_A(n) \in \Theta(n^3)$.
2. This recurrence is balanced, so $W_B \in \Theta(n^2 \log(n))$.
3. This recurrence is leaf-dominated so $W_C \in \Theta(n^{\log_{25}(5)}) = \Theta(n^{\frac{1}{2}})$.

Note: I may ask about a recurrence where the brick method does not immediately apply. See the homework for some examples.

4 Turing Machines

Describe a Turing machine on the alphabet $\{0,1\}$ that expects a string in $s \in \{0,1\}^\star$ and returns the string s interleaved with 0's.
For example, if the initial input is 011100101101#, where # is the blank symbol, then the output should be $\underline{0}0\underline{1}0\underline{1}0\underline{1}0\underline{0}\underline{0}0\underline{1}0\underline{0}\underline{1}0\underline{1}0\underline{0}0\underline{1}\#$. Every other symbol has been underlined to emphasize that these underlined symbols are the string s.

Solution:
I am naming the states by numbers. The transition function is described in the form $x, y; R, (j)$, which means that if you read x, write y, move the head to the right, and change the state to j.

1. (Go right once) 1,1;R,(2) 0,0;R,(1) #, #;Halt
2. (Go right, check for blank) #, #; L(12) 0,0;R(3) 1,1;R(3)
3. (Go right until blank) 0,0;R (3) 1,1;R(3) #,#; L(4)
4. (copy and move right) 0,0;R(5) 1,1; R (6) #, #; R (8)
5. (paste 0 to right) 0, 0; L (7) 1, 0; L (7)
6. (paste 1 to right) 0,1;L(7) 1,1;L(7)
7. (move left and loop) 0,0;L(4) 1,1;L(4)
8. (Copy to the left once) 0,#;L(9) 1,#; L(10)
9. (paste 0 to the left) #, 0; R (11)
10. (paste 1 to the left) #, 1; R (11)
11. (Restart to 1) #,#,R (1)
12. (Step left) 0,0;L(13) 1,1; L(13)
13. (Step left again) #,#; L(14)
14. (check nonblank, move right) 0,0;R(15), 1,1;R(15) #,#;Halt
15. (Write 0) #,0;L(12)

Idea: Move the string to introduce blanks between the letters. (states 1-11) Then replace all the blanks with 0's. (states 12-15)

1. Find the end of the input
2. Move every letter to the right by 1.
3. Move the leftmost letter to the left by 1.
4. Repeat on the first letter of the contiguous part of the string.
5. You're done repeating when the contiguous part has length 1. You only move right once before encountering a #. This is how you know to go to the next phase of the algorithm. At this point, the symbols $0, 1$ and blanks alternate and the head is at the rightmost nonblank symbol.
6. Take two steps left and verify that the string has not ended. If the string has ended, halt. Otherwise, move right again and overwrite the blank with a #.

Notes:

1. I have a particular problem that in mind that is simpler than this. Like this problem, you will be asked to design a Turing machine that performs a certain task.
2. You are free to introduce new symbols in these sorts of problems, beyond just $1, 0, \#$.
3. You can get significant partial credit for describing how you go about designing the Turing machine, as long as it's clear that you understand what a Turing machine is. If short on time, the explanation in bullets 1-6, together with the explicit (and correct) claim that each step can be implemented with a few Turing machine steps, would earn at least 80% of the points.
4. To get full points, the you need to provide the information about the states and transition function in some form, preferably using circles and labeled arrows, as in the notes.

5 Finite State Machines

Describe the finite state machine over the alphabet $\{a, t, g, c\}$ that recognizes the regular language $((at)^*(gt)^*c)^*$.

Solution:

1. a:2, g:4, c:1 (Accept state)
2. t:3
3. a:2, g:4, c:1
4. t:5,
5. c:1, g:4

The numbers label the states. The transition function away from the state is listed to the state's right. The letter tells you which state to go to next. If a letter is encountered that is not mentioned, then machine will reject the string.
Note: Check that this is correct. It's usually clearer to draw a diagram, like in the notes. I may ask you a question that could involve character groups. No weird lookback expressions. I could ask you to write the regular expression associated to a given machine.
Know the regular expression things: -parenthesis -conacatenation -[] (or) -Kleene star

6 Divide and Conquer

An inversion of a list l of integers is a pair of indicies, $i < j$, such that $l[i] > l[j]$.
For example, the list $[5, 10, 3, 9, 1, 2, 3]$ has 14 inversions. These inversions are the pairs of indicies
$\{(0,2),(0,4),(0,5),(0,6),(1,2),(1,3),(1,4),(1,5),(1,6),(2,4),(2,5),(3,4),(3,5),(3,6)\}$, so there are 14 of them.

- Describe a divide-and-conquer algorithm that counts the number of inversions of a list of integers. Your algorithm should be $o(n^2)$.
- Analyze the work and span of your algorithm.

Solution: Split the list in half and calculate the number of inversions in each half. We must also account for the inversions between the two halves.
Sort both halves. For each item a in the left half, find the last b_a in the right half so that $a > b_a$. Add 1 plus the index of b_a to our count of inversions.
The algorithm has the following recurrence:
$W(n) = 2W(\frac{n}{2}) + n\log(n)$.
The combine step takes $\Theta(n\log(n))$, since the limiting step is the time that it takes to sort the left and right halves. Note that counting the inversions between both lists can be implemented in constant time with a loop through the left half, because both lists are sorted and the indicies b_a are easily seen to be increasing as a increases.
Using the Master Theorem, we can see that this recurrence solves to $\Theta(n\log(n)\log(n))$.
The span obeys the recursive equation
$S(n) = S(\frac{n}{2}) + \log(n)$, since it is possible to sort in logarithmic span. (Ok if you say $(\log(n))^2$, since these are the parallel sorting algorithms we saw). The Master Theorem shows that this is $(\log(n))^2$ Span.

7 Sorting/searching

Consider the following parallel version of bubblesort.

1. Compare each even-indexed item with the next item, swapping them if they are out of order. Do this in parallel.
2. Compare each odd-indexed item with the next item, swapping them if they are out of order. Do this in parallel.
3. Repeat steps 1 and 2 until the items are sorted.

Analyze the worst-case work and span of this parallel version of bubblesort.

Solution:
We showed in a homework that any sorting algorithm that involves only swapping adjacent elements has $O(n^2)$ work in the worst case. The worst case scenario is when the items are in reverse order.
To analyze the span of the parallel version, observe that after steps 1 and 2, each item will be closer to its sorted position by at least one index. Therefore, after n repetitions of steps 1 and 2, the list will be sorted. The span is $O(n)$. This is the best possible result, because in the worst case, the first item will be largest and will need to traverse the whole list, thereby requiring $O(n)$ repetitions of steps 1 and 2.
Note: There will be some question about sorting that involves analyzing a modification of the algorithms that we studied.
Could ask about counting sort, the information bound, etc.

8 Map/Filter/Reduce

A descent of a sequence l is an index i such that $l[i] > l[i+1]$.
Use the map/filter/reduce paradigm to calculate the number of descents in a sequence.
Calculate the work and span of your algorithm.
You can use the Python syntax.

Solution 1: Call the input list *my_list* and call its length n. We can use the list comprehension.

```
sum([1 for index, item in enumerate(my_list)
     if index<n-1 and my_list[index+1]<item])
```

The work of this algorithm is $O(n)$. To see this, the list comprehension can be implemented with $O(n)$ work with a loop through *my_list*. The sum can also be implemented in $O(n)$ work with a loop of length n where each step performs constant work.
List comprehensions can be implemented in $\Theta(\log(n)$ span, and summing can be accomplished in $\log(n)$ span. Solution 2:

1. Zip *my_list* with *my_list*[1 :] (*my_list* without its first element). Call the result *Zip_list*.
2. Filter *Zip_list* so that it contains only the pairs p that satisfy $p[0] > p[1]$. Call the result *Filtered_list*.
3. Map *Filtered_list* using the constant function. Call the result *Mapped_list*.
4. Reduce *Mapped_list* using addition (sum the list). Output the result.

The work of this algorithm is $\Theta(n)$, because it consists of 4 steps that all take $\Theta(n)$ work. The span of steps 1 and 3 is constant. The span of steps 2 and 4 is $\Theta(\log(n))$. So the total span is the sum of these, which is $\Theta(\log(n))$.
Note: It's a bit unclear what functions you're allowed to use. Here are some that you should know.

1. Map
2. Filter
3. Reduce
4. Scan (Accumulate)
5. Zip
6. Enumerate
7. Collect (Groupby)

8. Flatten
9. Chain
10. Length

9 P vs NP

The classes P and NP are formulated as decision problems, meaning that they must have yes/no answers. Technically, this means that the travelling salesman problem is not an NP problem, because the result is a number. What is the justification for defining NP problems as decision problems?

Solution:
You can use binary search by repeatedly asking "Is the answer to the travelling salesman problem greater than c or less than c for various values of c. This allows you to convert numerical problems like travelling salesman to decision problems without much performance loss.
Note: This question in the exam is meant to be a 'general knowledge question.' There's no calculating, I'm just asking you to recall a significant fact that was mentioned.