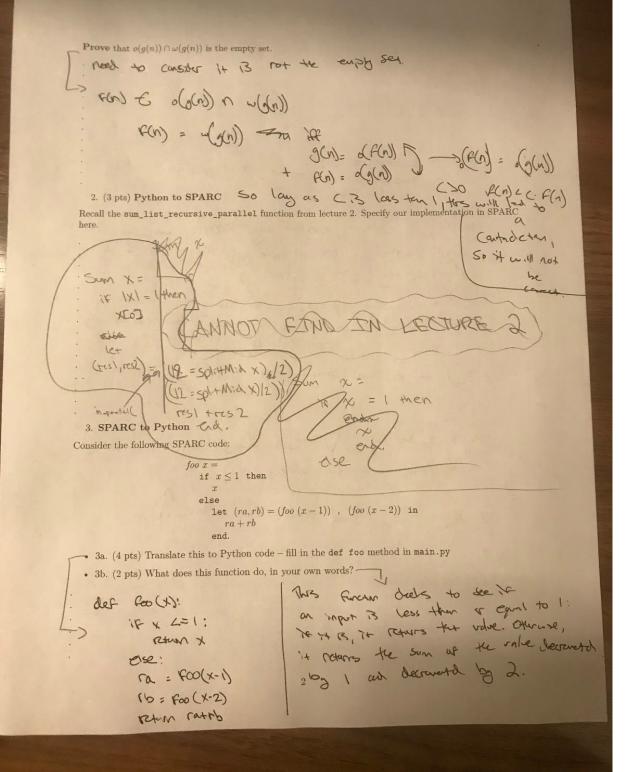
CMPS 2200 Assignment 1 In this assignment, you will learn more about asymptotic notation, parallelism, functional languages, and algorithmic cost models. As in the recitation, some of your answer will go here and some will go in main.py. You are welcome to edit this assignment-01 md file directly, or print and fill in by hand. If you do the latter, please scan to a file assignment-01.pdf and push to your github repository. 1. (2 pts ea) Asymptotic notation tes 2 never eclipses of consent that you can multiply 2n+1 vertically, the will exert a consent that you can multiply 2n+1 • 1a. Is $2^{n+1} \in O(2^n)$? Why or why not? Every two will exert a construction of the second in the will extra \mathcal{J} . 1b. Is $2^{2^n} \in O(2^n)$? Why or why not? (Tes.) you can tole a construct, Say 1000000, where upon on the another construct, Say cas, and who you multipy the set by 2^n and 2^n respectively, 2^n will expire 2^n . (no) tese fireties will contine on their pates for at new cross no nather who cashes they re mypul by. • 1d. Is $n^{1.01} \in \Omega(\log^2 n)$? for the sur reason into a bue. • 1e. Is $\sqrt{n} \in O((\log n)^3)$? no.) At toll ests, In will alway eclose (logn3) for large constant values • 1f. Is $\sqrt{n} \in \Omega((\log n)^3)$?

• 1g. Consider the definition of "Little o" notation:

 $g(n) \in o(f(n))$ means that for every positive constant c, there exists a constant n_0 such that $g(n) \le c \cdot f(n)$ for all $n \ge n_0$. There is an analogous definition for "little omega" $\omega(f(n))$. The distinction between o(f(n)) and O(f(n)) is that the former requires the condition to be met for every c, not just for some c. For example, $10x \in o(x^2)$, but $10x^2 \notin o(x^2)$.



4. Parallelism and recursion Consider the following function: def longest_run(myarray, key) Input: 'myarray': a list of ints 'key': an int Return: the longest continuous sequence of 'key' in 'myarray E.g., longest_rum([2,12,12,8,12,12,0,12,1], 12) == 3 4a. (8 pts) First, implement an iterative, sequential version of longest_run in main.py. • 4b. (4 pts) What is the Work and Span of this implementation? were = (C(2)) return Max((sx) -4c. (8 pts) Next, implement a longest_run_recursive, a recursive, divide and conquer implementation. This is analogous to our implementation of sum_list_recursive. To do so, you will need to think about how to combine partial solutions from each recursive call. Make use of the provided class Result. 4d. (4 pts) What is the Work and Span of this sequential algorithm? longs+_run (nums): I MAKEN+ = Ø IF CHY MANCH range (len(runs)): If nonstill != nons(i-Band cut > MaxCN+: CN+ = MaxCN+ From (max dua maxCN+) CN+= MaxCN+ Prus (nat-ON+) Else: CM = CN++1 • (3 pts) 4e. Assume that we parallelize in a similar way we did with sum_list_recursive. That is each recursive call spawns a new thread. What is the Work and Span of this algorithm? (neol) o Know = nove < work = & o (Mogn)