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
```

- 1a. Is $2^{n+1} \in O(2^n)$? Why or why not? .
 - .
- .
- 1b. Is $2^{2^n} \in O(2^n)$? Why or why not?
 - .
- .
- 1c. Is $n^{1.01} \in O(\log^2 n)$?
 - .
 - .
- 1d. Is $n^{1.01} \in \Omega(\log^2 n)$?
 - .
 - .
- 1e. Is $\sqrt{n} \in O((\log n)^3)$?
 - .
 - .
- 1f. Is $\sqrt{n} \in \Omega((\log n)^3)$?
 - .

2. SPARC to Python

Consider the following SPARC code of the Fibonacci sequence, which is the series of numbers where each number is the sum of the two preceding numbers. For example, 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610 ...

```
\begin{array}{l} foo\ x=\\ \text{ if }\ x\leq 1\ \text{ then }\\ x\\ \text{ else }\\ \text{ let }(ra,rb)=(foo\ (x-1))\ ,\ (foo\ (x-2))\ \text{ in }\\ ra+rb\\ \text{ end } \end{array}
```

- 2a. (6 pts) Translate this to Python code fill in the def foo method in main.py
- 2b. (6 pts) What does this function do, in your own words?

. .

.

3. 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., $longest_run([2,12,12,8,12,12,12,0,12,1], 12) == 3$

- 3a. (7 pts) First, implement an iterative, sequential version of longest_run in main.py.
- 3b. (4 pts) What is the Work and Span of this implementation?

.

• 3c. (7 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.

• 3d. (4 pts) What is the Work and Span of this sequential algorithm?

• 3e. (4 pts) 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?

.

n to to Algorithm nation, f(n) EO(g(n) quotient faster than 2", soit is not O

	is arbitrary since bases differ by a constant factor.
	(2)
	$\lim_{n \to \infty} f(n) = \lim_{n \to \infty} n^{1.01}$
	$\frac{\lim_{n\to\infty}\frac{f(n)}{g(n)}-\lim_{n\to\infty}\frac{n^{1.01}}{(\log_n)^2}$
	Set n=e ^t . Then the ratio be comes: e ^{1.01t} e ^{2.01t} t ² t
	WILL A TO THE COUNTY OF THE CO
100	is unbounded, so there are no constants c>0 and no with
	is unbounded, so there are no constants c> 0 and no with
	n1.01 < c(logn) for all n > no
	Therefore n 1.01 = w ((logn)2), not O ((logn)2).
	1 d. Yes. nº 1.01 E D ((logn)2).
	By definition, $f(n) \in SZ(g(n))$ if $J(c) O$ and n_0 such that $f(n) > cg(n)$ for all $n > n_0$.
	such that f(n) > cg(n) for all n?, no.
	Let f(n) = n and g(n) = (logn) (with any log base; bases
	differ by a constant tactor). We consider the fraction:
	lim (logn)2(0,00)
	let no et then the orio desones to so
	Set n=et. Then:
	$(\log n)^c = t^2$ $0 (t \rightarrow \infty)$, since an
	exponential dominates any polynominal. Hence there exists no
	within to got (near) of an actualining of disher
1	(log (n) < 1 for all n > no)
The second second	n - (copal) cut = a copal copal copal
	which implies not 7 (Logn) for all n 7, no
The state of the s	VIBOOK INTERNATIONAL
	THE RIVER OF THE PARTY OF THE P

Taking $c = 1$ yields $n^{1.01} \in \Omega$ ($(\log n)^2$). In fact, $n^{1.01} = \omega$. $n^{1.01} = \omega ((\log n)^2)$.
4.04
n = co(cogn)
1 1 = 1 0 (10) 3
1e. No. In $\notin O((\log n)^3)$.
To be in O, we would need constants c>O and no with
In & c(legn) for all n 7, no.
Let n=et. The inequality becomes et2 < Ct3. We consider
the rate:
to the etter start son and as betalling the set of the
t3 t→∞ (noll)) 10.2
So no such constant c can work. Hence In grows faster than
any polylogarithm, in particular faster than (logn)3.
100 100 100 100 100 100 100 100 100 100
If. Yes. In ((legn) 3).
Definition: $f(n) \in \Omega(g(n))$ if $\exists c > 0$ and n_0 such that $f(n) > c g(n)$ for all $n > n_0$.
+(n) > c g(n) for all n > no.
(on sider the fraction:
$(\log n)^3$
0 t/2
(logn) < 1 for all n >, no,
VIME 2 TOO WILL IN 1/10,
336 A 3 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
which is equivalent to $\sqrt{n} > (\log n)^3$ for all $n > n_0$.
Taking C=1 satisfies the definition. Therefore In tSL((logn))
Taking C=1 satisfies the definition. Therefore In £ D2 ((log n)) and, in fact, In = w((log n)).
a a lista de la color de la co



D. foo(x) returns the x-th Fibonacci number. It uses the standard recurrence Fo = 0, Fq = 1, Fn = Fn-1 + Fn-2 for n/2. The base case returns x when x < 1. Otherwise, it computes the two previors Fibonacci numbers and add them up. (3) 3b. Work: θ(n). We scantbelist once with O(1) work per element. Span: θ(n). The computation is sequential. 3d. Work: θ(n). Pecurrence I(n) = 2T(n/2) + θ(1) gives linear work by the Muster Theorem. Span: θ. Calls rum one after another in a single thread, so span equals total work. 3e. More and span if we farallelize the two recursive calls work: θ(n). Parallelism does not change the total amount of work. Span: θ(logn). Critical path satisfies S(n) = S(n/2) + θ(1) fince the two halvos run in parallel.	A Title Control of the International States of the States	
Standard recurrence Fo = 0, F1 = 1, Fn = Fn + Fn = for n>, 2. The base case returns x when x < 1. Otherwise, it computes the two previors Fibonacci numbers and add them up. (3) 3b. Work: \theta(n). We scan the list once with O(1) work per element. Span: \theta(n). The computation is sequential. 3d. Work: \theta(n). The computation is sequential. 3d. Work: \theta(n). Fecurrence T(n) = 2T(n/2) + \theta(1) gives linear work by the Master Theorem. Span: \theta. Calls rum one after another in a single. thread, so span equals total work. 3e. Work and span if we farallelize the two recursive calls. Work: \theta(n). Parallelism does not change the total amount of work. Span: \theta(logn). Critical path Satisfies S(n) = S(n/2) + \theta(1)	2	
Standard recurrence Fo = 0, F1 = 1, Fn = Fn + Fn = for n>, 2. The base case returns x when x < 1. Otherwise, it computes the two previors Fibonacci numbers and add them up. (3) 3b. Work: \theta(n). We scan the list once with O(1) work per element. Span: \theta(n). The computation is sequential. 3d. Work: \theta(n). The computation is sequential. 3d. Work: \theta(n). Fecurrence T(n) = 2T(n/2) + \theta(1) gives linear work by the Master Theorem. Span: \theta. Calls rum one after another in a single. thread, so span equals total work. 3e. Work and span if we farallelize the two recursive calls. Work: \theta(n). Parallelism does not change the total amount of work. Span: \theta(logn). Critical path Satisfies S(n) = S(n/2) + \theta(1)	b. foo(x) returns	the x-th Fibonacci number. It uses the
Fo = 0, F ₁ = 1, F _n = F _{n-1} + F _{n-2} for n>, 2. The base case returns x when x \(\frac{1}{2}\). Otherwise, it computes the two previors tibonacci numbers and add them up. (3) 3b. Nork: θ(n). We scan the list once with O(1) work per element. Span: θ(n). The computation is sequential. 3d. Nork: θ(n) Pecumence T(n) = 2T(n/2) + θ(1) gives linear work by the Master Theorem. Span: θ. Calls rum one after another in a single thread, so span equals total work. 3e. Work and span if we parallelize the two recursive calls Work: θ(n). Parallelism does not change the total amount of work. Span: θ(logn). Critical path Satisfies S(n) = S(n/2) + θ(1)		
The base case returns x when x \(\frac{1}{2} \). Otherwise, it computes the two previors \(\frac{1}{2} \) be nacci numbers and add them up. (3) 3b. Nork: \(\theta(n) \). We scan the list once with O(1) work per element. Span: \(\theta(n) \). The computation is sequential. 3d. Work: \(\theta(n) \). The computation is sequential. 3d. Work: \(\theta(n) \) Pecurrence \(\ta(n) = 2T(n/2) + \theta(1) \) gives linear work by the Master Theorem. Span: \(\theta \). Calls run one after another in a single thread, so span equals total work. 3e. Work and span if we farallelize the two recursive calls Work: \(\theta(n) \). Parallelism does not change the total amount of work. Span: \(\theta(logn) \). Critical path satisfies \(S(n) = S(n/2) + \theta(1) \)	Fo = 0, Fy =	=1, Fn = Fn-1 + Fn-2 for n>, 2.
He two prentors to bo nacci numbers and add them up. 3b. Nork: θ(n). We scan the list once with O(1) work per element. Span: θ(n). The computation is sequential. 3d. Worn: θ(n) Pecumence T(n) = 2T(n/2) + θ(1) gives linear work by the Master Theorem. Span: θ. Calls run one after another in a single thread, so span equals total work. 3e. Work and span if we farallelize the two recursive calls Worn: θ(n). Parallelism does not change the total amount of work. Span: Y(logn). Critical path Satisfies S(n) = S(n/2) + θ(1)	The base case	returns x when x { 1. Othernise, it computes
element. Span: $\theta(n)$. The computation is sequential. 3d. Worn: $\theta(n)$ Pecurrence $T(n) = 2T(n/2) + \theta(1)$ gives Linear work by the Master Theorem. Span: θ . Calls run one after another in a single thread, so span equals total work. 3e. Work and span if we farallelize the two recursive calls work: $\theta(n)$. Parallelism does not change the total amount of work. Span: $\theta(n)$. Critical path satisfies $\theta(n) = \theta(n/2) + \theta(1)$		
element. Span: $\theta(n)$. The computation is sequential. 3d. Worn: $\theta(n)$ Pecurrence $T(n) = 2T(n/2) + \theta(1)$ gives Linear work by the Master Theorem. Span: θ . Calls run one after another in a single thread, so span equals total work. 3e. Work and span if we farallelize the two recursive calls work: $\theta(n)$. Parallelism does not change the total amount of work. Span: $\theta(n)$. Critical path satisfies $\theta(n) = \theta(n/2) + \theta(1)$	3	
Span: θ(n). The computation is sequential. 3d. Norn: θ(n) Pecurrence T(n) = 2T(n/2) + θ(1) gives Linear work by the Master Theorem. Span: θ. Calls run one after another in a single thread, so span equals total work. 3e. Work and span if we farallelize the two recursive calls work: θ(n). Parallelism does not change the total amount of work. Span: θ(logn). Critical path Satisfies S(n) = S(n/2) + θ(1)	36. Work: 0(n).	We scan the list once with O(1) work per
3d. Worn: Un) Pecumence $T(n) = 2T(n/2) + \theta(1)$ gives Linear work by the Master Theorem. Span: θ . Calls run one after another in a single thread, so span equals total work. Be. Work and span if we farallelize the two recursive calls worn: $\theta(n)$. Parallelism does not change the total amount of work. Span: $\theta(n)$. Critical path satisfies $\theta(n) = \theta(n/2) + \theta(1/2)$	element.	
Linear work by the Master Theorem. Span: θ . Calls run one after another in a single thread, so span equals total work. Be. Work and span if we farallelize the two recursive calls work: $\theta(n)$. Parallelism does not change the total amount of work. Span: $\theta(\log n)$. Critical path satisfies $S(n) = S(n/2) + \theta(1)$. Span : θ(n).	The Computation is sequential.
Span: θ . Calls run one after another in a single thread, so span equals total work. Be. Work and span if we parallelize the two recursive calls work: $\theta(n)$. Parallelism does not change the total amount of work. Span: $\theta(\log n)$. Critical path satisfies $\theta(n) = \theta(\log n)$	3d wore: (n) P	ecumence T(n)=2T(n/2)+0(1)gres
Heread, so span equals total nork. Be. Work and span if we parallelize the two recursive calls Work: A(n). Parallelism does not change the total amount of work. Span: E(logn). Critical path Satisfies S(n) = S(n/2) + O(1)	linear nork by the	Master Theorem.
Heread, so span equals total nork. Be. Work and span if we parallelize the two recursive calls Work: A(n). Parallelism does not change the total amount of work. Span: E(logn). Critical path Satisfies S(n) = S(n/2) + O(1)	. Span: 0. C	alls run one after another in a single
of nork. Span: O(logn). Critical path satisfies S(n) = S(n/2) +O(1)	thread, so span equ	als total nork.
of nork. Span: O(logn). Critical path satisfies S(n) = S(n/2) +O(1)	20 . Mark and exa	n if we parallelize the two recursive calls
of nork. Span: Ologn). Critical path satisfies S(n) = S(n/2) +O(1)	· worn: ()(n).	Parallelism does not change the total amount
. Span: Ologn). Critical path satisfies S(n) = S(n/2) +O(1)		
since the two halves run in parallel.		. Critical path satisfies S(n) = S(n/2)+O(1
	since the two halves	nun in parallel.

