Yan Zhu

1.a Yes

$2^{n+1} \in O(2^n)$

$2 \cdot 2^n \le C 2^n$

$2 \cdot 2^n / 2^n \le C \cdot 2^n / 2^n$

$2 \le C$

if $C \ge 2$, the inequality is satified.

thus, $2^{n+1} \in O(2^n)$. Yes

1.b No

$2^{2n} \in O(2^n)$

$\lim_{n \to \infty} \frac{2^{2n}}{2^n} = \infty$

when $n \to \infty$ 2^{2n} increase faster then 2^n, and there is no C will satisfy it, so No.

1.c. No

$n^{1.01} \in O(\log^2 n)$

$\lim_{n \to \infty} \frac{n^{1.01}}{\log^2 n}$

$\log \frac{(n^{1.01})}{(\log^2 n)} = \log c n^{1.01} - \log c (\log^2 n)$

$= 1.01 \cdot \log^n - 2 \log c (\log^n)$

$\log^n$ is much fast than $\log c (\log^n)$

when $n \to \infty$, $n^{1.01}$ grow faster then $\log^2 n$, even with any C. So No.

1.d Yes.

$n^{1.01} \in \Omega(\log^2 n)$

Use $\lim_{n \to \infty} \frac{n^{1.01}}{\log^2 n}$

$\log c n^{1.01} - \log c (\log^2 n)$

$= \log 1.0^n - 2 \log c (\log^n) \to \infty$

$\log^n$ is much fast than $\log c (\log^n)$

so $\frac{n^{1.01}}{\log^2 n} \to \infty$

since the result is ∞, it proved $n^{1.01} \in \Omega(\log^2 n)$.

So, Yes.

1.e **No.**

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

$n^{\frac{1}{2}} \leq \log^3 n$

$\lim_{n \to \infty} \frac{n^{\frac{1}{2}}}{\log^3 n}$

$\log(n^{\frac{1}{2}}) - \log(\log^3 n)$

$= \frac{1}{2}\log(n) - \log(\log n) \to \infty$

since $\log n$ grows faster the $\log(\log n)$, $\sqrt{n}$ grows faster the $\log^3 n$. the answer is No.

1.f **Yes.**

$\sqrt{n} \in \Omega(\log^3 n)$

$n^{\frac{1}{2}} \geq \log^3 n$

$\lim_{n \to \infty} \frac{n^{\frac{1}{2}}}{\log^3 n}$

$\log(n^{\frac{1}{2}}) - \log(\log^3 n)$

$= \frac{1}{2}\log(n) - 3\log(\log n) \to \infty$

Since $n^{\frac{1}{2}}$ grows faster then $\log^3 n$, so Yes.

1.g

$o(g(n)) \cap w(g(n))$

$\downarrow$ means $\qquad \downarrow$ means

$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0 \qquad \lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$

A: a $f(n)$ that can fit both 0 and ∞ is impossible, because the $\lim_{n \to \infty} \frac{f(n)}{g(n)}$ cannot represent two different incresment rate. thus $o g(n) \cap w(g(n)) = \emptyset$

For example:

$f(n) = n^2 \qquad g(n) = \log n$

$\lim_{n \to \infty} \frac{n^2}{\log^2} \to \infty$

$n^2 \notin O(\log n) \qquad n^2 \in w(\log n)$

2.b

If x is 0 or 1, the function directly returns x (i.e., it returns 0 when x is 0 and returns 1 when x is 1).
For any other value of x greater than 1, the function calls itself recursively with x - 1 and x - 2 as arguments, adds the results of these two calls together, and returns the sum.
Basically, it calculates the Fibonacci number of a given int X.

3. b.

O(1)

work:
max_run =0 1
run_=0 1
loop(n):
(if f mylist[i] == key 1
run+=1 1
if run>nax_run 1
max_run =run 1) 4n
return max_run 1
1+1+4n+1 =3+4n
O(1)=4n+3

The work and span are both O(n).

Work: The algorithm traverses the entire list, and performs a series of constant operations each time, resulting in the total workload.

Span: Since all operations of the algorithm are executed sequentially, so is the span.

3.d

Operation Count Overview
. . if len(mylist) == 0:

Operations: 1
return Result(0, 0, 0, False)

Operations: 1
if len(mylist) == 1:

Operations: 1
if mylist[0] == key:

Operations: 1
return Result(1, 1, 1, True)

Operations: 1
return Result(0, 0, 0, False)

Operations: 1
mid = len(mylist) // 2

Operations: 1
left_result = longest_run_recursive(mylist[: mid], key)

Operations: W(n/2)
right_result = longest_run_recursive(mylist[mid:], key)

Operations: W(n/2)
left_len = left_result. left_size

Operations: 1
if left_result. is_entire_range and mylist[mid] == key:

Operations: 2
left_len += right_result. left_size

Operations: 1
right_len = right_result. right_size

Operations: 1
if right_result. is_entire_range and mylist[mid-1] == key:

Operations: 2
right_len += left_result. right_size

Operations: 2
cross_len = 0

Operations: 1
if mylist[mid-1] == key and mylist[mid] == key:

Operations: 2
cross_len = left_result. right_size + right_result. left_size

Operations: 1
max_len = max(left_result. longest_size, right_result. longest_size, cross_len)

Operations: 1
entire_range = left_result. is_entire_range and right_result. is_entire_range

Operations: 1
return Result(left_len, right_len, max_len, entire_range)

Operations: 1

$O(1) = 23$

The $T(n)$ of the recursive Algorithm:

$T(n) = 2T(n/2) + O(1)$, $W(n) = O(n)$

since every element is in the recursive.

$S(n) = O(n)$

Because the process is liner, it depends on the result of last recursive.

3.e

$W(n) = O(n)$

$S(n) = O(\log n)$

$W(n)$ is same as 3 d.

But since we are using parallel computing, leading to the $O(\log n)$.