

1. (a) In given sequence of n elements, $\{a_1, a_2, \dots, a_n\}$, store minimum and maximum value as a_1, a_1 . And linearly compare with next elements.
 If $a_1 < a_2$, then maximum is a_2 and if $a_1 > a_2$, then minimum is a_2 from first index of sequence to last index of sequence.
 And so on, when the stored minimum and maximum values are a_m, a_M and we are to compare a_i ,
 if $a_i < a_m$, change a_m with a_i ,
 if $a_i > a_M$, change a_M with a_i ,
 else, don't change the stored data.
 if we repeat the loop until when $i = n$, we get the minimum and maximum elements of the given sequence.

In for-loop, executing two comparisons about $(n - 1)$ times.

So, The number of comparisons of sequence is $2(n - 1)$.

(b) set pivot index, middle of sequence = (length of sequence) / 2
 And then, divide two parts of sequence; first to pivot, pivot to last.
 Compute minimum and maximum elements of each sequence recursively and compare these two minimum and maximum values.

recursive equation is,

$n = 2^k$ (k : nonnegative integer)

$$C(2) = 1$$

$$C(n) = 2C(n/2) + 2$$

$$= 2^2 C(n/4) + 2^2 + 2$$

$$= 2^3 C(n/8) + 2^3 + 2^2 + 2$$

$$= 2^{k-1} C(2) + \sum_{i=1}^{k-1} 2^i = n/2 + \sum_{i=1}^{k-1} 2^i = n/2 + 2^k - 2 = n/2 + n - 2$$

$$= 3n/2 - 2$$

2. (a)

with the function given in the exercise,
 assume that,

$T(n)$ = runtime of the function with input nonnegative integer n

then, because the given function is recursive to call $fib(n - 1)$ and $fib(n - 2)$,

$$T(n) = T(n - 1) + T(n - 2) + 1$$

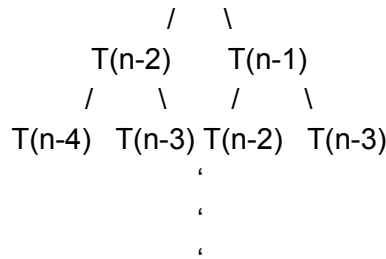
(1 for adding $fib(n - 1)$ and $fib(n - 2)$)

$$T(0) = 0$$

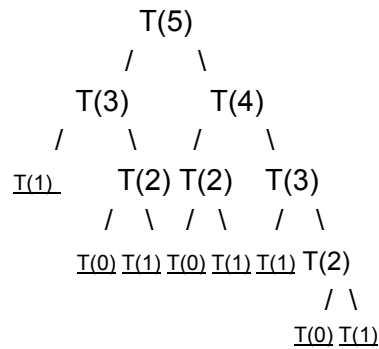
$$T(1) = 0$$

if we draw the $T(n)$ by tree,

$T(n)$



to explain with $T(5)$,



The function recursively calls until the k of $T(k)$ is 0 or 1.

$$\begin{aligned}
 T(n) &= T(n-1) + T(n-2) + 1 = (T(n-2) + T(n-3) + 1) + (T(n-3) + T(n-4) + 1) + 1 \\
 &= T(n-2) + T(n-3) + T(n-3) + T(n-4) + 3 = \dots
 \end{aligned}$$

like in exercise, the number $T(n)$ is counted with only additions, the each additions are between the leaves of the graph.

Then, if we draw the tree until every leaf is either $T(1)$ or $T(0)$, for all n ,

$$T(n) = (\text{number of outer leaves on the tree from } T(n)) - 1$$

let's say $(\text{number of outer leaves on the tree from } T(n)) = f(n+1)$

suppose $f(n) = \text{fib}(n)$

(i) if $n=0$, $n=1$,

$$T(0) = f(0+1) - 1 = \text{fib}(1) - 1 = 1 - 1 = 0,$$

$$T(1) = f(1+1) - 1 = \text{fib}(2) - 1 = 1 - 1 = 0$$

$$T(n) = f(n+1) - 1$$

(ii) suppose that when $n \leq k$,

$$T(k) = f(k+1) - 1,$$

then

$$T(k+1) = T(k) + T(k-1) + 1 = (f(k+1) - 1) + (f(k) - 1) + 1$$

$$= (f(k+1) + f(k)) - 1 = f(k+2) - 1 = f((k+1)+1) - 1,$$

so, by strong induction when $f(n) = \text{fib}(n)$

$$\therefore T(n) = f(n+1) - 1$$

and, the height of the tree is n .

then,

$$f(n+1) \leq 2^{n-1}$$

$$f(n+1) = O(2^{n-1})$$

$$f(n+1) = O(2^n)$$

$$T(n) = f(n+1) - 1 = O(2^n)$$

so,

$$f(n+1) = O(2^n)$$

$$\therefore T(n) = O(2^n)$$

further more, we found out that

$$T(n) = \text{fib}(n+1) - 1$$

and we know the general solution of fibonacci numbers,

$$\text{fib}(n) = \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^n - \frac{1}{\sqrt{5}} \left(\frac{1-\sqrt{5}}{2} \right)^n,$$

when n is enough big,

$$\text{fib}(n+1) \approx \left(\frac{1+\sqrt{5}}{2} \right)^n * \text{fib}(n)$$

$$\text{fib}(n) = \theta \left(\frac{1+\sqrt{5}}{2} \right)^n \approx \theta(1.6^n)$$

$$T(n) = \text{fib}(n+1) - 1 \approx \theta(1.6^{n+1}) = \theta(1.6^n)$$

so, when n is enough

$$\therefore T(n) = \theta(1.6^n)$$

$$(b) \text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)$$

When function $\text{fib}(n)$ is called, first it calls $\text{fib}(n-1)$. And all process of $\text{fib}(n-1)$ is completed, then $\text{fib}(n-2)$ is called. So considering it, $\text{fib}(n-1)$ calls $\text{fib}(n-2)$, $\text{fib}(n-2)$ calls $\text{fib}(n-3)$... , $\text{fib}(2)$ calls $\text{fib}(0)$ and because $\text{fib}(0)$ is last one, $\text{fib}(2)$ calls $\text{fib}(1)$.

These process stored at memory stack as $\text{fib}(n)$, $\text{fib}(n-1)$, $\text{fib}(n-2)$, $\text{fib}(n-3)$, ..., $\text{fib}(1)$, $\text{fib}(0)$.

Then, compute reverse order, $\text{fib}(2) = \text{fib}(2) + \text{fib}(1)$. So stack pops $\text{fib}(1)$, $\text{fib}(0)$ and store $\text{fib}(2) = 2$. $\text{fib}(3) = \text{fib}(2) + \text{fib}(1)$, so $\text{fib}(3)$ calls $\text{fib}(1)$ and pops $\text{fib}(2)$, $\text{fib}(1)$ and store $\text{fib}(3) = 3$. And so on.

The stack pops two elements and store one repeatedly. So needed memory doesn't exceed $n+1$ ($\text{fib}(0)$ to $\text{fib}(n)$). As a result, It can say $\Theta(n)$.

(c) def fib_linear(n):

 if n == 0: return 0

 elif n == 1: return 1

 else:

 a = 0

```
b = 1
for i in range(n):
    a, b = b, a+b
return a
```

in the progress,
fib_linear(n) has 1 addition in each for loop, and
fib_linear(n) has n loops inside.
so it has linear runtime; $O(n)$
and in fib_linear(n),
the data is repeatedly being written on a, b and overlapped.
so the data space doesn't grow.
fib_linear(n) has its constant space $O(1)$.

∴ linear runtime $O(n)$, constant space $O(1)$.