1. We have the recurrence relation in terms of big-oh expressions.

BASIS. $T(0) = O(1)$.

INDUCTION. $T(n) = O(1) + T(n-1)$, for $n > 0$.

Here is the recurrence relation now in terms of unknown constants.

BASIS. $T(0) = a$.

INDUCTION. $T(n) = b + T(n-1)$, for $n > 0$.

We shall now solve this recurrence relation. We determine the sequence

$$\begin{aligned} T(n) &= b + T(n-1) \\ T(n-1) &= b + T(n-2) \\ T(n-2) &= b + T(n-3) \\ &\cdots \\ T(1) &= b + T(0). \end{aligned}$$

Now we can substitute simply. We express $T(n)$ in terms of the value of $T(n-1)$ and get

$$T(n) = b + (b + T(n-2)) = 2b + T(n-2).$$

And again, $T(n)$ expressed in terms of the value of $T(n-2)$ to get

$$T(n) = 2b + (b + T(n-3)) = 3b + T(n-3).$$

This should suffice to capture the pattern. We see that, in terms of i, we have

$$T(n) = ib + T(n-i).$$

Where $i = n$, we will have expressed $T(n)$ in terms of the basis $T(0)$. This is

$$T(n) = nb + T(0) = nb + a.$$

Now we express $T(n)$ back in terms of big-oh expressions. The term nb is n proportional to some constant and a is only proportional to a constant. Thus the terms are $O(n) + O(1)$. Therefore the running time of `sum` is $O(n)$.

2. A suitable size measure is the length of the list input.

BASIS. $T(0) = T(1) = O(1)$.

INDUCTION. $T(n) = O(1) + T(n-1)$, for $n > 1$.

Rewriting this in terms of unknown constants we have

BASIS. $T(0) = T(1) = a$.

INDUCTION. $T(n) = b + T(n-1)$, for $n > 1$.

Let us find a pattern. We have

$$T(n) = b + (b + T(n-2)) = 2b + T(n-2).$$

That is good enough. We determine that

$$T(n) = ib + T(n-i).$$

To express $T(n)$ in terms of $T(1)$, substitute i for $n+1$. Then we have

$$T(n) = (n+1)b + T(n-(n+1)) = (n+1)b + T(1) = (n+1)b + a.$$

We have a function proportional to n and a function proportional to a constant. The two terms expressed as big-oh expressions are $O(n) + (1)$, and $O(n)$ is the running time of `find0`.

3. A suitable size measure is $m = n - i$, the number of elements still unsorted.

BASIS. $T(1) = O(1)$.

INDUCTION. $T(m) = O(m) + T(m-1)$, for $m > 1$.

We substitute for constants.

BASIS. $T(1) = a$.

INDUCTION. $T(m) = bm + T(m-1)$, for $m > 1$.

By substitution we try to discover a pattern

$$\begin{aligned}
T(m) &= bm + T(m-1) \\
&= bm + b(m-1) + T(m-2) = b(2m-1) + T(m-2) \\
&= b(2m-1) + b(m-2) + T(m-3) = b(3m-3) + T(m-3) \\
&= 3b(m-1) + b(m-3) + T(m-4) = b(4m-6) + T(m-4) \\
&= b(4m-6) + b(m-4) + T(m-5) = b(5m-10) + T(m-5).
\end{aligned}$$

Now we see the pattern. It is

$$T(m) = b\left(km - \sum_{j=1}^{k-1} j\right) + T(m-k) = b(km - k(k-2)/2) + T(m-k).$$

Let $k = m - 1$. Then we have $T(n)$ expressed in terms of $T(1)$ and also with k in terms of m, which is

$$\begin{aligned}
T(m) &= b\left((m-1)m - \sum_{j=1}^{m-2} j\right) + T(1) \\
&= b((m-1)m - (m-2)(m-2+1)/2) + a \\
&= b((m-1)m - (m-2)(m-1)/2) + a \\
&= b((m-1)(m-(m-2)/2)) + a \\
&= b((m-1)((2m-m+2)/2)) + a \\
&= b((m-1)((m+2)/2)) + a \\
&= b(m-1)(m+2)/2 + a.
\end{aligned}$$

If we multiply we will get a m^2 term. Thus the recursive selection sort program is $O(m^2)$.

4.

BASIS. $T(1) = T(2) = O(1)$.

INDUCTION. $T(n) = O(1) + T(n-1) + T(n-2)$, for $n > 2$.

We substitute for constants.

BASIS. $T(1) = T(2) = a$.

INDUCTION. $T(n) = b + T(n-1) + T(n-2)$, for $n > 2$.

We try to find a pattern

$$
\begin{aligned}
T(n) &= b + T(n-1) + T(n-2) \\
&= b + (b + T(n-2) + T(n-3)) + (b + T(n-3) + T(n-4)) \\
&= b + (b + (b + T(n-3) + T(n-4)) \\
&\quad + (b + T(n-4) + T(n-5))) \\
&\quad + (b + (b + T(n-4) + T(n-5)) \\
&\quad + (b + T(n-5) + T(n-6))).
\end{aligned}
$$

The relation between the inputs of T has the pattern

n-1

n-2

with the next being

n-2

n-3 n-3

n-4

and the next being

n-3

n-4 n-4 n-4

n-5 n-5 n-5

n-6

The pattern to capture so far is that the number of T terms double starting from 2, the number of b terms is one less than the number of T terms, the number of rows and columns increases by 1, the greatest $n-i$ term decreases by 1 and the least $n-i$ term decreases by 2. The number of terms on each row resembles the corresponding binomial coefficient.

Thankfully we do not have to discover a method of computing the coefficients. We need only relate this problem to another. Let k be the least i in all the $T(n-i)$ terms, then we have

$$T(n) = (2^k - 1)b + \sum_{j=k}^{2k} \binom{k}{j-k} T(n-j).$$

However, expressing this equation in terms of basis cases is difficult. Instead, we rely on the pattern above. There are 2^k terms involving T. If we let $k = n-2$, then all the T terms must be basis cases. Therefore we have

$$(2^{n-2} - 1)b + 2^{n-2}a.$$

That is, there are 2^{n-2} basis cases a. We can rewrite this as

$$2^{n-2}(b+a) - b.$$

Thus we have a function proportional to an exponential factor and a function proportional to a constant. Since 2^{n-2} is $O(2^n)$, then the `fibonacci` program is $O(2^n)$.

5. Let n be the number of calls left until the basis is reached.

```
int gcd(int i, int j)
{
    if (i%j == 0) return j;
    else return gcd(j, i%j);
}
```

BASIS. $T(0) = O(1)$.

INDUCTION. $T(r) = O(1) + T(r_{n-1})$, for $r > 0$.

BASIS. $T(0) = a$.

INDUCTION. $T(r) = b + T(r_{n-1})$, for $r > 0$.

The pattern we find is

$$T(r) = kb + T(r_{n-k}).$$

Let $k = n$. Then we shall say that $T(r_0)$ has no calls left before reaching the basis, meaning that it is this particular call that is $T(0)$. Thus we have

$$T(r) = nb + T(0) = nb + a.$$

What we wish to solve now is what n is. This looks like a linear big-oh relation but that is because we obscured what r and n are. Let us trace the calls to gcd, which are

$$gcd(i, j)$$
$$gcd(j, i \mod j)$$
$$gcd(i \mod j, j \mod (i \mod j))$$

We look at the third call. If $j = 1$ then the basis would have been satisfied in the second call. Since the third call is invoked then j must be ≥ 2. We claim (by clairvoyance) that $m = i \mod j$ for $j \geq 2$ leaves $m \leq i/2$ (it is also assumed that $i > j$ under gcd). This process repeats where the first argument m of gcd is at least halved, which is the worst case.

The number of calls n depends on the halving of the original argument i. Hence nb is proportional to $\log i$. Thus the running time $T(r) = nb + a$ is $O(\log i) + O(1)$ which is $O(\log i)$.