```python
def find_max(data):
  """Return the maximum element from a nonempty Python list."""
  biggest = data[0]              # The initial value to beat
  for val in data:               # For each value:
    if val > biggest             # if it is greater than the best so far,
      biggest = val              # we have found a new best (so far)
  return biggest                 # When loop ends, biggest is the max
```

1, n, n, 1

$T(n) = 2n + 2 \approx O(n)$

```python
def unique1(S):
  """Return True if there are no duplicate elements in sequence S."""
  for j in range(len(S)):
    for k in range(j+1, len(S)):
      if S[j] == S[k]:
        return False             # found duplicate pair
  return True                    # if we reach this, elements were unique
```

$\frac{n^2}{2} - \frac{n}{2}$

1 <

$\frac{n(n+1)}{2}$

$T(n) = \frac{n^2}{2} - \frac{n}{2} + 1 \approx O(n^2)$

j	k
0	n-1
1	n-2
2	n-3
⋮	⋮
n-2	1
n-1	X

range(1, n)

n-1, n

$1 + 2 + 3 + \cdots + n-2 + n-1$

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

```python
def unique2(S):
  """Return True if there are no duplicate elements in sequence S."""
  temp = sorted(S)               # create a sorted copy of S
  for j in range(1, len(temp)):
    if S[j−1] == S[j]:
      return False               # found duplicate pair
  return True                    # if we reach this, elements were unique
```

n log n

n-1

1 <

$T(n) = n \log n + \cancel{n-1} \cancel{+1} \approx O(n \log n)$ (1, n)

n-1 번

```
      def prefix_average1(S):
        """Return list such that, for all j, A[j] equals average of S[0], ..., S[j]."""
1       n = len(S)
n       A = [0] * n                 # create new list of n zeros
        for j in range(n):
n         total = 0                 # begin computing S[0] + ... + S[j]
          for i in range(j + 1):
n²/2+n/2    total += S[i]
n         A[j] = total / (j+1)      # record the average
1       return A
```

$T(n) = \frac{n^2}{2} + \frac{7n}{2} + 2 \approx O(n^2)$

$\frac{n}{2} + 3n$

range(i)

j	i
0	1
1	2
2	3
$\vdots$	$\vdots$
$n-1$	n

$= \frac{n(n+1)}{2} = \frac{n^2}{2} + \frac{n}{2}$

```
    def prefix_average3(S):
      """Return list such that, for all j, A[j] equals average of S[0], ..., S[j]."""
1     n = len(S)
n     A = [0] * n                 # create new list of n zeros
1     total = 0                   # compute prefix sum as S[0] + S[1] + ...
      for j in range(n):
n       total += S[j]             # update prefix sum to include S[j]
n       A[j] = total / (j+1)      # compute average based on current sum
1     return A
```

$T(n) = 3n + 3 \approx O(n)$

```
def disjoint1(A, B, C):
  """Return True if there is no element common to all three lists."""
  for a in A:
    for b in B:
      for c in C:
        if a == b == c:
          return False        # we found a common value
  return True                 # if we reach this, sets are disjoint
```

n^3

$1 <$

$T(n) = n^3 + 1 \approx O(n^3)$

```
def disjoint2(A, B, C):
  """Return True if there is no element common to all three lists."""
  for a in A:
    for b in B:
      if a == b:              # only check C if we found match from A and B
        for c in C:
          if a == c           # (and thus a == b == c)
            return False      # we found a common value
  return True                 # if we reach this, sets are disjoint
```

n^2

n번 비교

n번만

n^2

$\Rightarrow n^2$

$1 <$

$T(n) = 2n^2 + 1 \approx O(n^2)$

A.

```
    def func(A, n):
1       i = 0
1       s = 0
        while i<n:
n           if i%2==0:
n/2             s += A[i]
n           i += 1
1       return s
```

$T(n) = \frac{5n}{2} + 3$

$\approx O(n)$

B.

```
    def func(A, n):
1       i = 0
1       s = 0
        while i < n:
1           j = i
            while j < n:
n               j += 1
n               i += 1
n               s += A[j]
1           i += 1
1       return s
```

$\frac{i}{0}$ $\underline{j}$

$T(n) = 3n + 5$

$\approx O(n)$

```
def factorial(n):
  if n == 0:
    return 1
  else:
    return n * factorial(n-1)
```

What is the running time of the recursive factorial function?

(Assume)

$T(n)$ = running time of factorial (n)

C = local ops / call

k = recursions

$$\begin{aligned}
T(n) &= C + T(n-1) \\
&= C + C + T(n-2) = 2C + T(n-2) \\
&= 2C + C + T(n-3) = 3C + T(n-3) \\
&= \ldots = kC + T(n-k) \\
&= Cn + T(0) \\
&= Cn + 2 \approx O(n)
\end{aligned}$$

$$\begin{pmatrix} n-k=0 \\ n=k \end{pmatrix}$$

```
def binary_search(data, target, low, high):
  """Return True if target is found in indicated portion of a Python list.

  The search only considers the portion from data[low] to data[high] inclusive.
  """
  if low > high:
    return False                  # interval is empty; no match
  else:
    mid = (low + high) // 2
    if target == data[mid]:       # found a match
      return True
    elif target < data[mid]:
      # recur on the portion left of the middle
      return binary_search(data, target, low, mid - 1)
    else:
      # recur on the portion right of the middle
      return binary_search(data, target, mid + 1, high)
```

(Assume)

$T(n)$ = running time of binary_search (data, target, low, high)

C = local ops / call

k = recursions

$$\begin{aligned}
T(n) &= C + T\left(\frac{n}{2}\right) \\
&= C + C + T\left(\frac{n}{4}\right) = 2C + T\left(\frac{n}{2^2}\right) \\
&= 2C + C + T\left(\frac{n}{2^3}\right) = 3C + T\left(\frac{n}{2^3}\right) \\
&= \ldots = kC + T\left(\frac{n}{2^k}\right) \\
&= c(\log n + 1) + T(0) \\
&= C\log n + C + 2 \approx O(\log n)
\end{aligned}$$

$$\begin{pmatrix} \frac{n}{2^k} = 0 \\ \frac{n}{2^{k-1}} = 1 \\ n = 2^{k-1} \\ \log n = k-1 \\ k = \log n + 1 \end{pmatrix}$$

D.

```python
def func(A, n):
    if n == 0:
        return A[0]
    else:
        if n % 2 == 0:
            return A[n]+func(A, n//2)
        else:
            return A[n]-func(A, n//2)
```

(Assume)

$T(n)$ = running time of func(A, n)

c = local ops / call

k = recursions

$$
\begin{aligned}
T(n) &= c + T\left(\tfrac{n}{2}\right) \\
&= c + c + T\left(\tfrac{n}{4}\right) = 2c + T\left(\tfrac{n}{2^2}\right) \\
&= 2c + c + T\left(\tfrac{n}{2^3}\right) = 3c + T\left(\tfrac{n}{2^3}\right) \\
&= \cdots = kc + T\left(\tfrac{n}{2^k}\right) \\
&= c(\log n + 1) + T(0) \\
&= c\log n + c + 2 \\
&\approx O(\log n)
\end{aligned}
$$

$$
\left(
\begin{aligned}
\frac{n}{2^k} &= 0 \\
\frac{n}{2^{k-1}} &= 1 \\
n &= 2^{k-1} \\
\log n &= k-1 \\
k &= \log n + 1
\end{aligned}
\right)
$$