R-1.7 Order the functions by the big-Oh notation.
Group these functions that are big-Theta of one another.

\[ \frac{1}{n}, 2^{100}, \log \log n, \frac{\log n}{n^{0.01}}, 5n, 3n^{0.5}, 2 \log n, 5n, \]
\[ n \log n, 6n \log n, 2n \log n, 4n^{3/2}, 4 \log n, \]
\[ n^2 \log n, n^3, 2^n, 4^n, 2^{2n} \]

R-1.27 A = \{ -2, -4, 3, -1, 5, 6, -7, -2, 4, -3, 2, 7 \}
Max suffix subarray: 1, 3, -1, 5, 6, 7
Sum = 13

R-1.28 Find amortized running time of an operation
in a series of n add operations on an initially empty table implemented with an array
such that the capacity increment parameter is always maintained to be \( \lceil \log (n+1) \rceil \), where \( n \) is the no. of elements in the array.

- Suppose the initial array size is 1 and we begin with \( n \) adds.
- Let \( k = \lceil \log (n) \rceil \). Consider the last \( n/2 \) adds.
- For \( n/2 \leq k < n \), we know \( k-1 \leq \lfloor \log (k+1) \rfloor \leq k \)
Thus, for any expression executed in the last \( n/2 \) adds, we extend the array by either \( k-1 \) or \( k \) elements.
Let the number of expansions from \( \frac{n}{2} \) to \( n \) is \( E \), then
\[
\frac{n}{2k} \leq E \leq \frac{n}{2(k-1)}, \quad \text{or} \quad E = \Theta\left(\frac{n}{k}\right)
\]

Let the total cost of the last \( \frac{n}{2} \) adds be \( S \). Then
\[
\left(\frac{n}{2}\right)E + \frac{n}{2} + (k-1)E + \ldots + E(k-1) \leq S \leq \left(\frac{n}{2}\right)E + \frac{n}{2} + kE + \ldots + E_k
\]

Or equivalently
\[
\left(\frac{n}{2}\right)(E+1) + (k-1)E(E+1) \leq S \leq \left(\frac{n}{2}\right)(E+1) + kE(E+1)
\]

After simplification
\[
\left(\frac{n}{2}\right) + (k-1)E(E+1) \leq S \leq \left(\frac{n}{2}\right)(E+1) + kE(E+1)
\]

Since \( E \) is \( \Theta\left(\frac{n}{k}\right) \), we conclude that \( S \) is \( \Theta\left(\frac{n^2}{k}\right) \). And the amortized complexity of the last \( \frac{n}{2} \) operations is \( \Theta\left(\frac{n}{k}\right) \).

Let \( T \) be the total cost of all \( n \) operations. Then \( S \leq T < 2S \). The amortized complexity of all \( n \) operations is \( \Theta\left(\frac{n}{k}\right) \).

\[K = \lfloor \log_2(n) \rfloor\]
R-129 Describe a recursive algorithm for finding both
the minimum and maximum elements in an
array A of n elements. Your method should
return a pair (a, b) where a is the minimum
element and b the maximum. What is the
running time?

```
Algorithm MinMax(A)
Input: Array A of n elements
Output: (a, b)
```

```
a = A[0]
b = A[0]
for i = 1 to (n-1) do
    \[ a = \min (a, A[i]) \]
    \[ b = \max (b, A[i]) \]
\]
end for
```

```
\[ a = \min (A[0], MinMax(A[1:n-1])) \]
\[ b = \max (A[0], MinMax(A[1:n-1])) \]
return (a, b)
```

Running time = O(n²)

C-1.1 Modify MaxSubFastest so it returns j and k,
the indices of the max subarray.

```
Input: Array A of n elements
Output: Max subarray sum of array A, j and
k where j and k are in the max subarray.
```

```
M₀ = 0
for t = 1 to n do
    Mₜ = max (0, Mₜ₋₁ + A[t]²)
    m = Mₜ
end for
```

for \( t = 1 \) to \( n \) do
\[ m = \text{max} \& m, M \]
\[ k < t \]
\[ \text{sum} \leftarrow m \]
while \((\text{sum} \neq 0)\)
\[ \text{sum} \leftarrow \text{sum} - A[k \cdot j] \]
\[ k = k - 1 \]
\[ j = j + 1 \]

return \( m, j, k \)

C-13 What is the amortized running time of the operations in a sequence of \( n \) operations:
\[ P = P_1P_2 \ldots P_n \] if the running time of
\[ P_i \] is \( O(c_i) \) if \( i \) is a multiple of 3,
and a constant otherwise?

\[ P = P_1P_2 \ldots P_n \]
\[ a + a + 3c + a + a + 6c + a + a + 9c + \ldots n \]
Assume \( n \) is a multiple of 3

Then
\[ \frac{2na + (3c + 6c + \ldots + n)}{3} \]
\[ = \frac{2na + 3c}{3} \left( \frac{1 + 2 + \ldots + n}{3} \right) \]
\[ = O(n^2) \]
C-1.14 An n-degree polynomial \( p(x) = \sum_{i=0}^{n} a_i x^i \), where 
\( x \) is a real number and each \( a_i \) is a constant.

\( O(n^2) \) - time method for computing \( p(x) \) for
a particular \( x \).

Algorithm: Polynomial \( p(x) \)
Input: Array \( A \) representing each \( a_i \) in \( p(x) \).
Output: Value of \( p(x) \).

\[ p \leftarrow p + (A[i] \times \text{pow}(x, i)) \]

\[ p(x) = a_0 + x(a_1 + x(a_2 + x(\ldots + x(a_{n-1} + x_{n})) \ldots)) \]

Number of multiplications = \( n + n - 1 + \ldots + 3 + 2 + 1 = \frac{n(n+1)}{2} = O(n^2) \)

Number of additions = \( \# n = O(n) \).

C-1.24 Suppose that each row of an \( n \times n \) array \( A \) consists of 1's and 0's such that, in
any row \( i \) of \( A \), all the 1's come
before any 0's in that row.

Describe an \( O(n) \) method for finding
the row of \( A \) that contains the most 1's

Algorithm: Max's
Input: \( n \times n \) array \( A \) of 1's and 0's, such that
in any row, all 1's come before any 0's
Output: Row \( i \) number that has maximum \( 1's \)
\( n = 0 \)
\( c = 0 \)

\[ \text{flag} \leftarrow \text{true} \]

\[ \text{while} \ (n' = n \land c' = n) \ do \]

\[ \text{if} \ (\text{flag} = \text{true}) \]

\[ \text{if} \ (A[c] = 0) \]

\[ \text{flag} \leftarrow \text{false} \]

\[ c = c + 1 \]

\[ \text{else} \]

\[ \text{if} \ (A[c] = 1) \]

\[ \text{flag} = \text{true} \]

\[ \text{else} \]

\[ n = n + 1 \]

---

**C-130**

Consider an implementation of the extensible table where, when the table's capacity is reached, we copy the elements into a new array with \( 2 \times \text{TableSize} \) additional cells.

Show that performing a sequence of \( n \) odd operations takes \( \Theta(n^3) \) time.

Let \( a_i \) be the size of the array after the \( i \text{th} \) expansion. Then \( a_0 = 1 \) and \( a_i = a_{i-1} + \lceil \frac{a_{i-1}}{2} \rceil \)

We prove by induction that

\[ a_i = 1 + (i+1)^2 \]

\[ a_{i+1} = (i+1)(i+2) \]

**Base Case**

\[ i = 0: \ a_0 = (0+1)^2 = 1 = a_0 \]

\[ i = 1: \ a_2 = (0+1)(0+2) = 2 = a_1 \]
Induction Hypothesis:
\[ a_{2i} = (i+1)^2 \]
\[ a_{2i+1} = (i+1)(i+2) \]

Inductive Case:
\[ a_{2i+2} = a_{2i+1} + \sqrt{a_{2i+1}} = (i+1)(i+2) + \sqrt{(i+1)(i+2)} \]
\[ a_{2i+2} = (i+1)(i+2) + (i+2) = (i+2)^2 \]
\[ a_{2i+3} = a_{2i+2} + \sqrt{a_{2i+2}} = (i+2)^2 + \sqrt{(i+2)^2} \]
\[ a_{2i+3} = (i+2)^2 + (i+2) = (i+2)(i+3) \]

Thus, \[ a_{2i} = (i+1)^2 \] and \[ a_{2i+1} = (i+1)(i+2) \].

From
\[ a_{2i} = (i+1)^2 \] and \[ a_{2i+1} = (i+1)(i+2) \], we conclude
that \( (j+1)^2 / 4 \leq a_j \leq (j+2)^2 / 4 \) by letting
\[ j = 2i \text{ or } 2i+1. \]

Let \( k \) be the final array size. Then \( a_k \leq n \leq a_k \),
so that \( k^2 / 4 \leq n \leq (k+2)^2 / 4 \) or \( 2 \sqrt{n \cdot \frac{k^2}{4}} - \frac{k^2}{4} \leq k \leq 2 \sqrt{n} \).
That means the number of expansions \( k \) is \( \Theta(n^{1/2}) \).

Let \( S \) be the total cost of copying elements
from old arrays into new arrays during
\( k \) expansions.

Then \[ S = a_0 + a_1 + a_2 + \ldots + a_k. \]
Using
\[ (j+1)^2 / 4 \leq a_j \leq (j+2)^2 / 4 \]
we have
\[ (1^2 + 2^2 + \ldots + k^2) / 4 \leq S \leq (1^2 + 2^2 + 3^2 + \ldots + (k+1)^2) / 4 \]
\[ \text{On, } S \text{ is } \Theta(k^3) \text{ or } \Theta(n^{3/2}), \text{ because } k \text{ is } \Theta(n^{1/2}). \]

The total cost of \( n \) adds is
\[ n + S = \Theta(n^{3/2}) \]
and the amortized cost is \( \Theta(n^{1/2}) \).
Given an array A of n positive integers, find the longest subarray A such that all its elements are distinct.

What is the running time of the method?

Algorithm longest_subarray(A)
Input: Array A of size n
Output: A[i...k] such that A[i...k] is the longest repeated subarray in A.

max_length = 0
for i = 0 to n-1
    length = 0
    for j = i to n-1
        if (A[j] = A[j+1])
            length = length + 1
        if (length > max_length)
            max_length = length
            j = i
            k = j + 1
return A[i...k]

Running time O(n^2)