Algoritmos de ordenación y búsqueda
Fundamentos de la programación

Elena G. Barriocanal, Salvador Sánchez

Universidad de Alcalá

Noviembre de 2018

Búsqueda binaria

- Permite buscar un elemento en una colección (v) de manera más eficiente que con una búsqueda lineal.
- Es imprescindible que la colección tenga los elementos ordenados.
- Se basa en buscar sólo en la parte de la colección que puede contener el elemento:
  - Obtener la posición media de la colección (centro).
  - Si el elemento a buscar es menor que v[centro], se busca en la primera mitad de v.
  - Si el elemento a buscar es mayor que v[centro], se busca en la segunda mitad de v.

Búsqueda binaria

Implementación iterativa:

```
def busqueda_binaria_iterativa(v,elem):
  posicion = -1
  encontrado = False
  ini = 0
  fin = len(v)-1
  while (ini <= fin and not encontrado):
    centro = (ini + fin)//2
    if (v[centro] == elem):
      encontrado = True
      posicion = centro
    elif (elem < v[centro]):
      fin = centro-1
    else:
      ini = centro + 1
  return posicion
```

Búsqueda binaria

Implementación recursiva:

```
def busqueda_binaria_rec (v, elem, ini, fin ):
    centro = (ini+fin) // 2
    if (v[ centro ] == elem ):
        pos = centro
    else :
        if ( ini>fin):
            pos = -1
        elif ( elem < v[ centro ]):
            pos = busqueda_binaria_rec (v, elem, ini,centro-1 )
        else :
            pos = busqueda_binaria_rec (v, elem, centro+1, fin )
    return pos
```

Búsqueda binaria

Implementación recursiva con *slicing*:

```
def busqueda_binaria_rec (v,elem):
  """ lista, obj -> int
        OBJ: Busca un elemento en un vector y retorna su posicion,
             -1 si no se encuentra
        PRE: El vector debe estar ordenado y tener al menos un
             elemento """
  if len(v) == 0: pos = -1
  else:
    centro = len(v) // 2
    if (v[centro] == elem):
     pos = centro
    else:
      if (len(v) == 1):
        pos = -1
      elif (elem < v[centro]):
        pos = busqueda_binaria_rec(v[0:centro], elem)
      else:
        pos = busqueda_binaria_rec(v[centro+1:len(v)], elem)
        # la posicion en el subvector + centro + 1 sera la posicion
        # en el vector original, pero solo si el elemento estaba
        if pos != -1: pos += centro + 1
  return pos
```

Introducción: Ordenación

- Se desea ordenar una colección de elementos que poseen clave.
- Clasificar u ordenar consiste en reorganizar una colección de n elementos, de modo que:
  $clave[a_{p(1)}] < clave[a_{p(2)}] < ... < clave[a_{p(n)}]$

Métodos de ordenación básicos

- Poco eficientes: Adecuados para ejemplares de tamaño pequeño.
- Clasificación:
  - Métodos de *intercambio*: Ordenan el ejemplar intercambiando pares de elementos.
  - Métodos de *inserción*: Ordenan el ejemplar introduciendo de manera ordenada cada elemento en la posición que le corresponde dentro de una subsecuencia previamente ordenada.
  - Métodos de *selección*: Ordenan el ejemplar escogiendo de entre los elementos no ordenados el menor (o mayor), y colocándolo a continuación de los que ya están en orden.

Métodos de ordenación avanzados

- Aplican técnicas de diseño de algoritmos a los esquemas básicos (p.e. Merge-sort, Quick-sort)
- Generalización de algoritmos básicos (p.e. Shell)
- Se sustentan en estructuras de datos avanzadas (p.e. Heap-sort)
- Mejoran la eficiencia de los algoritmos básicos.
- Adecuados para ejemplares de gran tamaño.

Método de la burbuja: Traza (i)

- El elemento de menor clave «flota».
- Se intercambian los elementos para que el menor vaya «ascendiendo».

Método de la burbuja: Traza (i)

- El elemento de menor clave «flota».
- Se intercambian los elementos para que el menor vaya «ascendiendo».

```
22, 10, 12, 10, 1, 5
```

Método de la burbuja: Traza (i)

- El elemento de menor clave «flota».
- Se intercambian los elementos para que el menor vaya «ascendiendo».

```
22, 10, 12, 10, 1, 5
```

```
22, 10, 12, 10, 1, 5
```

Método de la burbuja: Traza (i)

- El elemento de menor clave «flota».
- Se intercambian los elementos para que el menor vaya «ascendiendo».

```
22, 10, 12, 10, 1, 5

22, 10, 12, 10, 1, 5

22, 10, 12, 10, 1, 5
```

Método de la burbuja: Traza (i)

- El elemento de menor clave «flota».
- Se intercambian los elementos para que el menor vaya «ascendiendo».

```
22, 10, 12, 10, 1, 5

22, 10, 12, 10, 1, 5

22, 10, 12, 10, 1, 5

22, 10, 12, 1, 10, 5
```

Método de la burbuja: Traza (i)

- El elemento de menor clave «flota».
- Se intercambian los elementos para que el menor vaya «ascendiendo».

22, 10, 12, 10, 1, 5

22, 10, 12, 10, 1, 5

22, 10, 12, 10, 1, 5

22, 10, 12, 1, 10, 5

22, 10, 12, 1, 10, 5

Método de la burbuja: Traza (i)

- El elemento de menor clave «flota».
- Se intercambian los elementos para que el menor vaya «ascendiendo».

22, 10, 1, 12, 10, 5

22, 10, 12, 10, 1, 5

22, 10, 12, 10, 1, 5

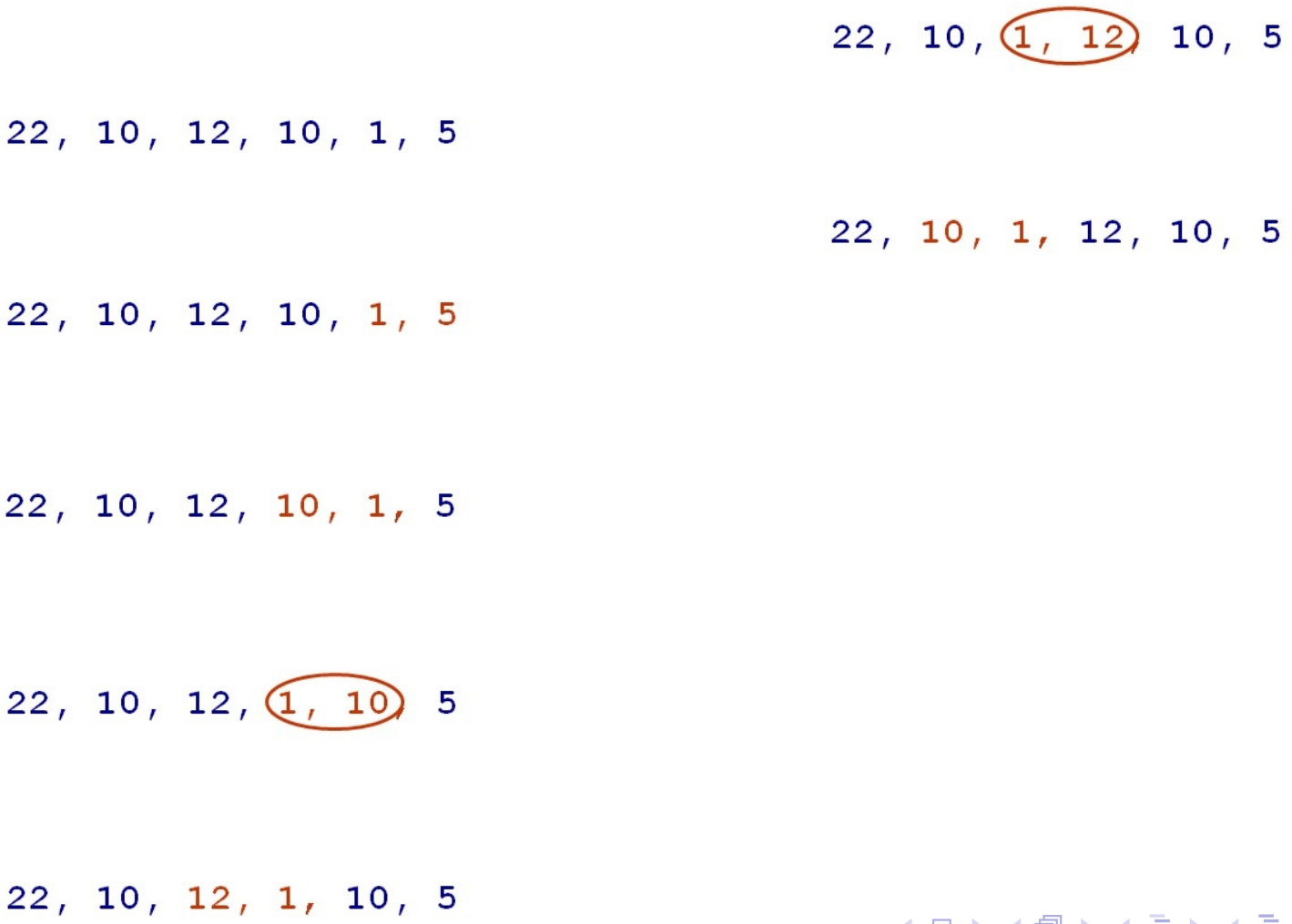
22, 10, 12, 10, 1, 5

22, 10, 12, 1, 10, 5

22, 10, 12, 1, 10, 5

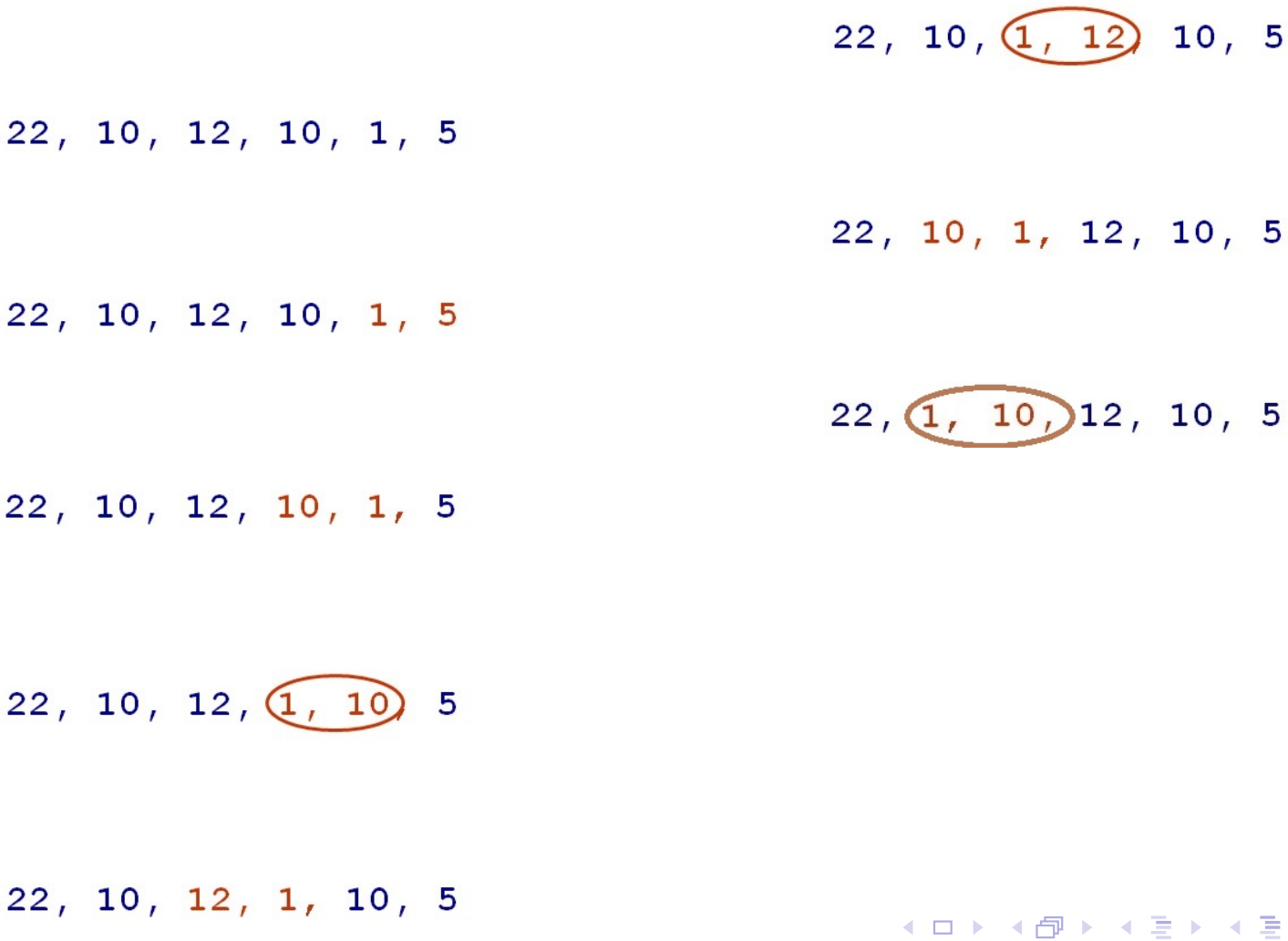
Método de la burbuja: Traza (i)

- El elemento de menor clave «flota».
- Se intercambian los elementos para que el menor vaya «ascendiendo».



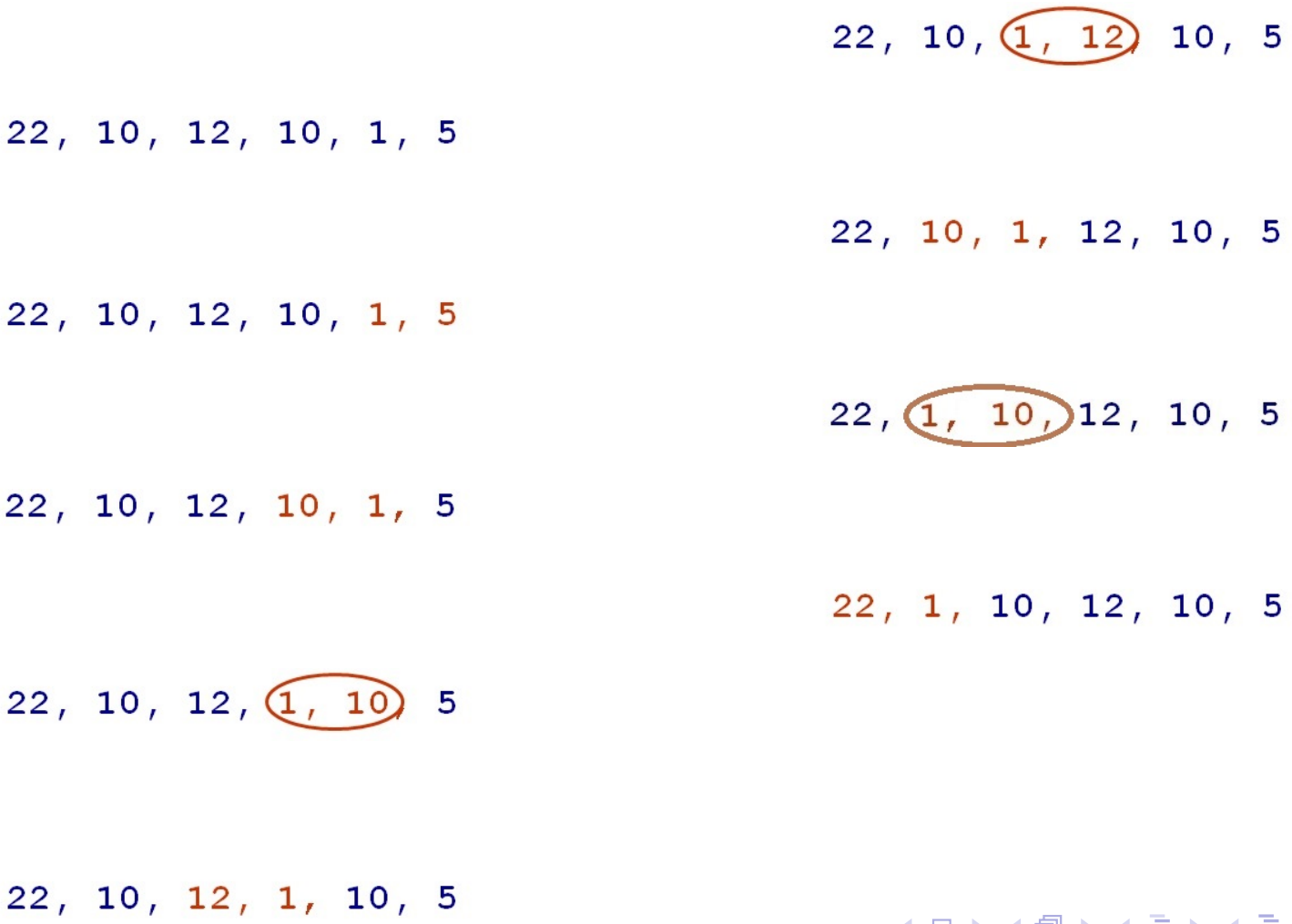
Método de la burbuja: Traza (i)

- El elemento de menor clave «flota».
- Se intercambian los elementos para que el menor vaya «ascendiendo».



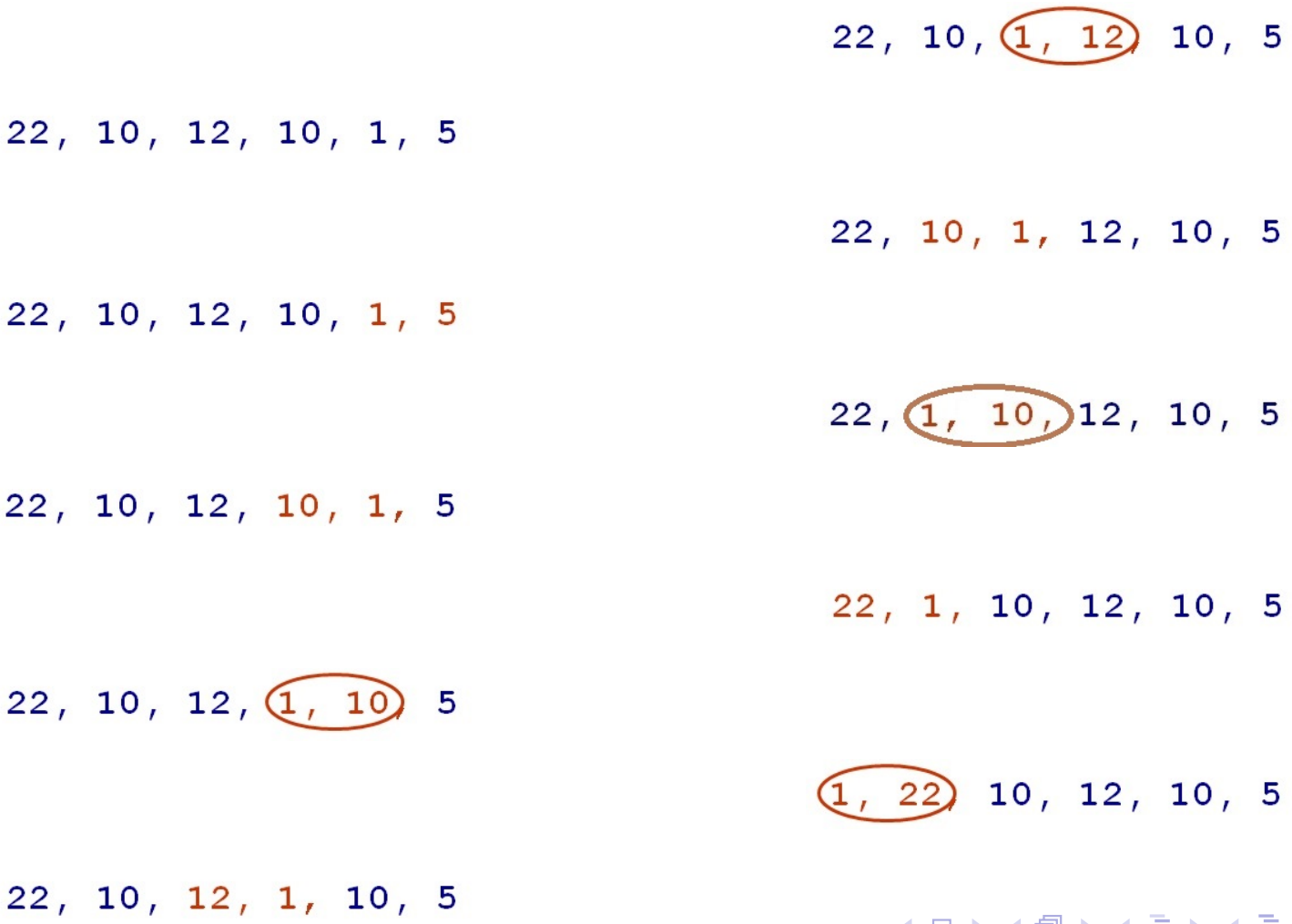
Método de la burbuja: Traza (i)

- El elemento de menor clave «flota».
- Se intercambian los elementos para que el menor vaya «ascendiendo».



Método de la burbuja: Traza (i)

- El elemento de menor clave «flota».
- Se intercambian los elementos para que el menor vaya «ascendiendo».



Método de la burbuja: Traza (i)

- El elemento de menor clave «flota».
- Se intercambian los elementos para que el menor vaya «ascendiendo».

22, 10, 12, 10, 1, 5

22, 10, 12, 10, 1, 5

22, 10, 12, 10, 1, 5

22, 10, 12, 1, 10, 5

22, 10, 12, 1, 10, 5

22, 10, 1, 12, 10, 5

22, 10, 1, 12, 10, 5

22, 1, 10, 12, 10, 5

22, 1, 10, 12, 10, 5

1, 22, 10, 12, 10, 5

Método de la burbuja

Habrá que hacer tantas ejecuciones de esta función como elementos desordenados tengamos:

```
def ascender(v,inicio,fin):
  for i in range (fin,inicio,-1):
    if (v[i] < v[i-1]):
      v[i], v[i-1] = v[i-1], v[i]
```

Método de la burbuja: Traza (ii)

22, 10, 12, 10, 1, 5

Método de la burbuja: Traza (ii)

22, 10, 12, 10, 1, 5

1, 22, 10, 12, 10, 5

Método de la burbuja: Traza (ii)

22, 10, 12, 10, 1, 5

1, 22, 10, 12, 10, 5

1, 5, 22, 10, 12, 10

Método de la burbuja: Traza (ii)

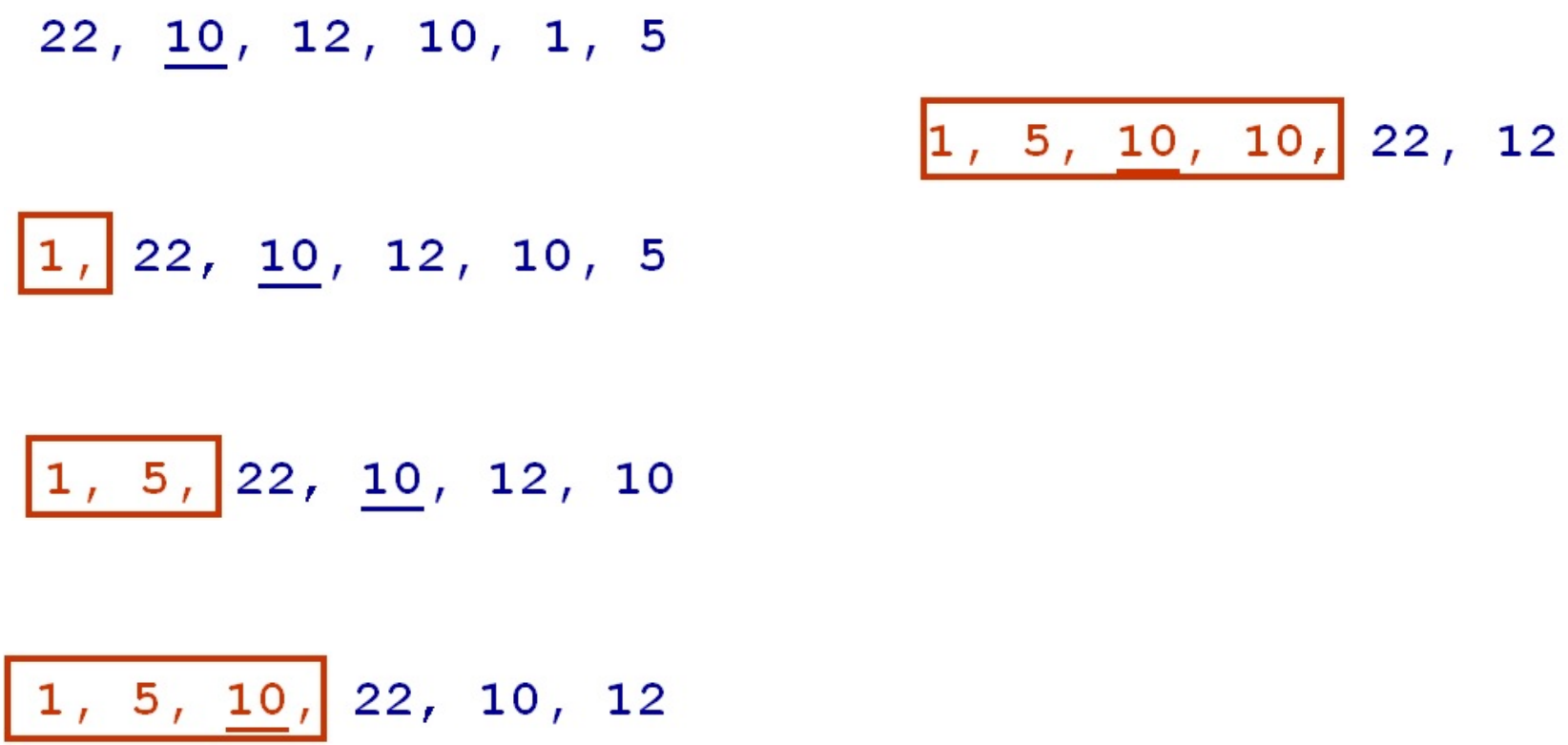
22, 10, 12, 10, 1, 5

1, 22, 10, 12, 10, 5

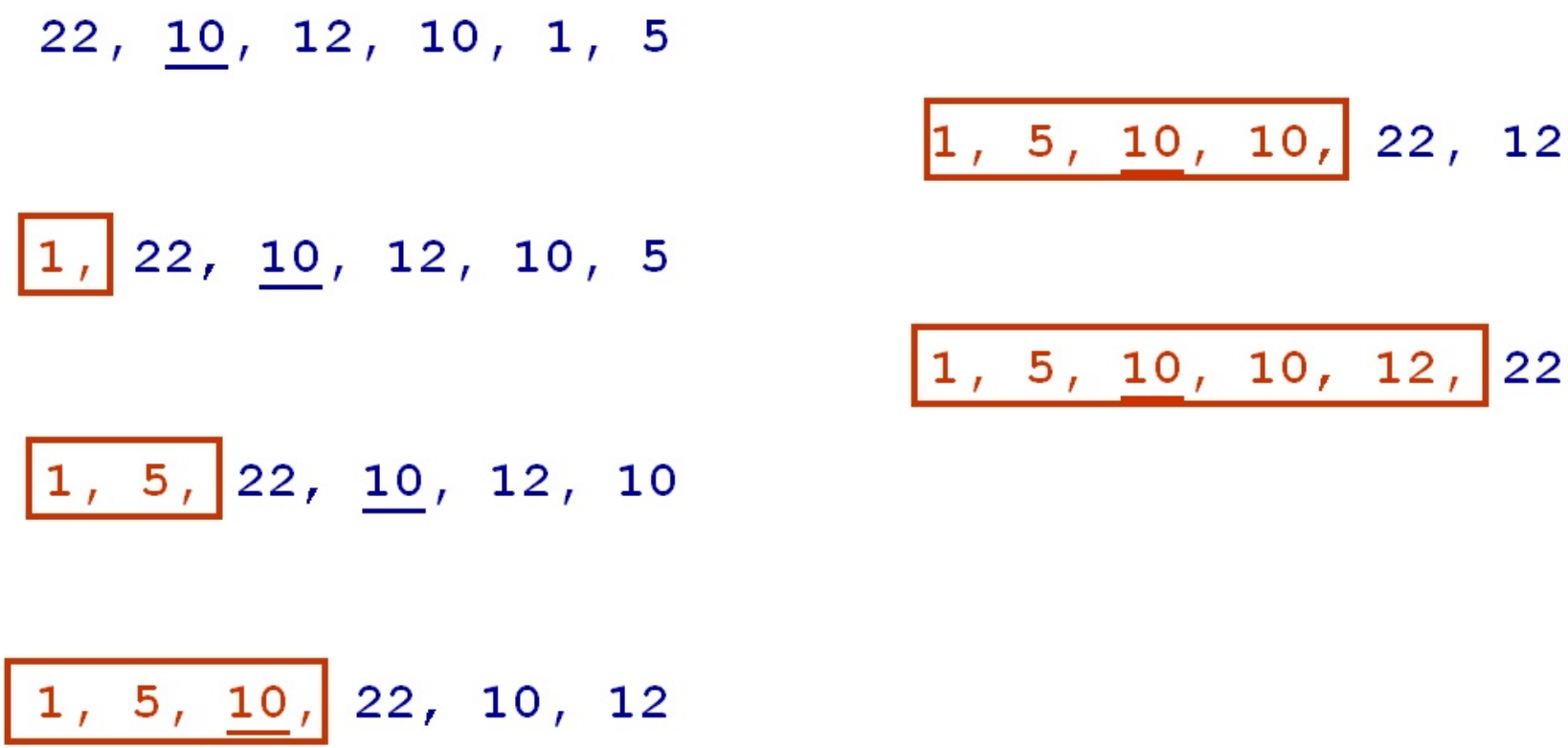
1, 5, 22, 10, 12, 10

1, 5, 10, 22, 10, 12

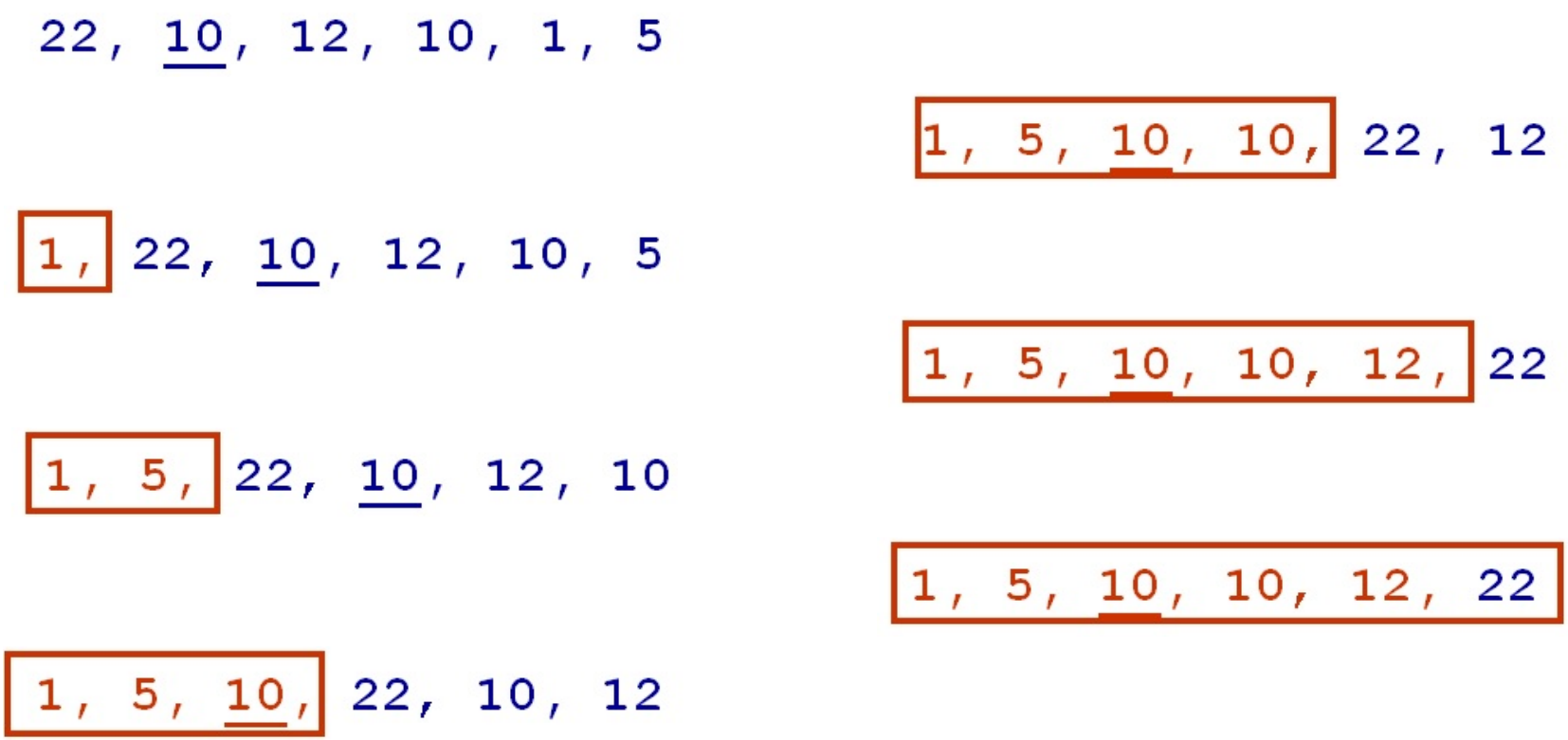
Método de la burbuja: Traza (ii)



Método de la burbuja: Traza (ii)



Método de la burbuja: Traza (ii)



Método de la burbuja

El algoritmo completo realiza varias pasadas invocando la función ascender:

```
def burbuja(v,inicio,fin):
  for pasada in range(inicio,fin-1):
    ascender(v,pasada,fin)
```

Método de la burbuja

El algoritmo completo realiza varias pasadas invocando la función ascender:

```
def burbuja(v,inicio,fin):
  for pasada in range(inicio,fin-1):
    ascender(v,pasada,fin)
```

recordando que...

```
def ascender(v,inicio,fin):
  for i in range (fin,inicio,-1):
    if (v[i] < v[i-1]):
      v[i], v[i-1] = v[i-1], v[i]
```

Método de Selección directa: Traza

- Selecciona el menor elemento de la lista y lo coloca justo a continuación de todos los que ya han sido ordenados.
- Al iniciar el algoritmo, el primer elemento (uno cualquiera de los dos extremos) se considera ordenado.
- El algoritmo de ordenación comienza propiamente con el segundo elemento a ordenar.

Método de Selección directa: Traza

22, 10, 12, 10, 1, 5

Método de Selección directa: Traza

22, 10, 12, 10, 1, 5

22, 10, 12, 10, 1, 5

Método de Selección directa: Traza

22, 10, 12, 10, 1, 5

22, 10, 12, 10, 1, 5

1, 10, 12, 10, 22, 5

Método de Selección directa: Traza

22, 10, 12, 10, 1, 5

22, 10, 12, 10, 1, 5

1, 10, 12, 10, 22, 5

1, 10, 12, 10, 22, 5

Método de Selección directa: Traza

22, 10, 12, 10, 1, 5

22, 10, 12, 10, 1, 5

1, 10, 12, 10, 22, 5

1, 10, 12, 10, 22, 5

1, 5, 12, 10, 22, 10

Método de Selección directa: Traza

22, 10, 12, 10, 1, 5

22, 10, 12, 10, 1, 5

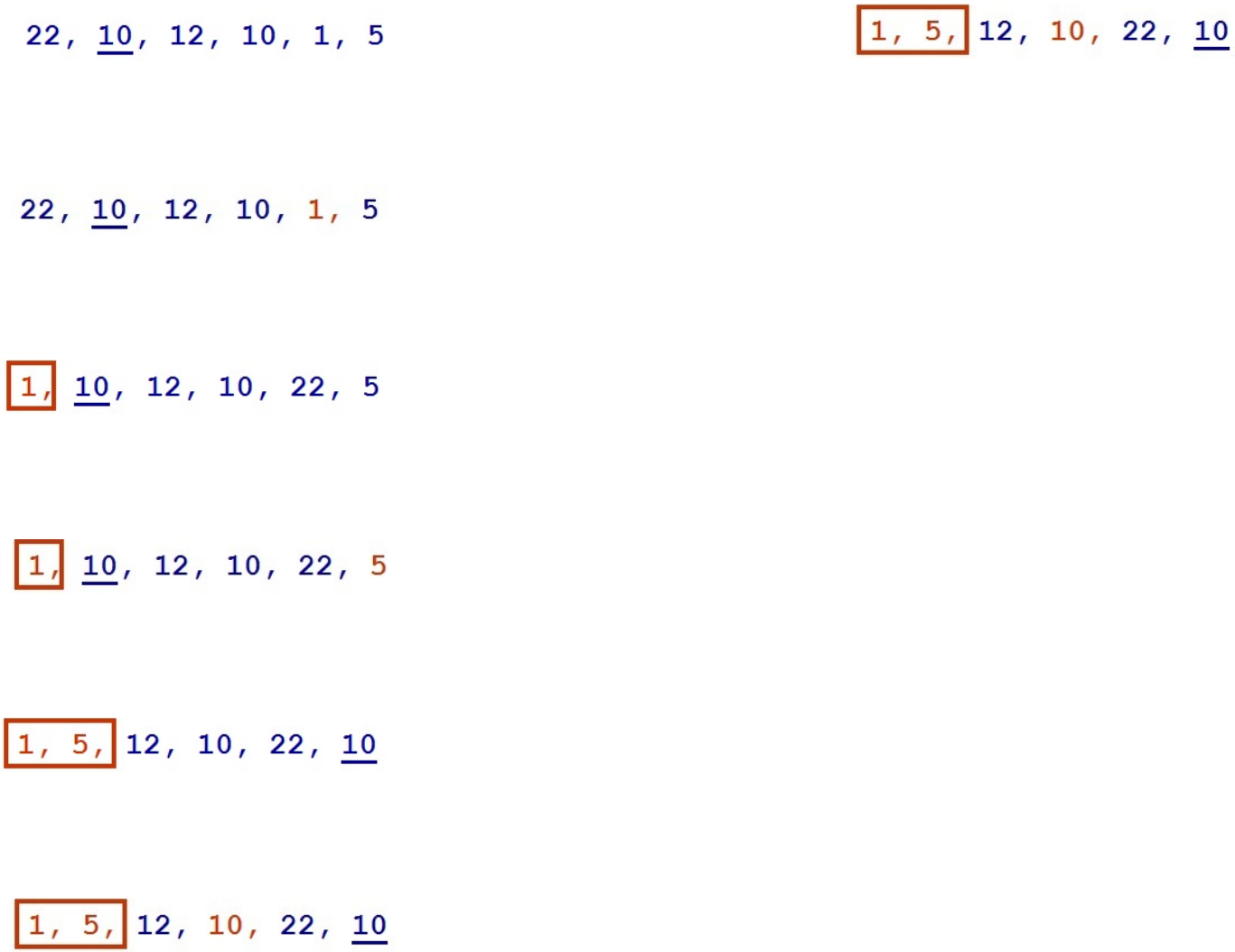
1, 10, 12, 10, 22, 5

1, 10, 12, 10, 22, 5

1, 5, 12, 10, 22, 10

1, 5, 12, 10, 22, 10

Método de Selección directa: Traza



Método de Selección directa: Traza

22, 10, 12, 10, 1, 5

22, 10, 12, 10, 1, 5

1, 10, 12, 10, 22, 5

1, 10, 12, 10, 22, 5

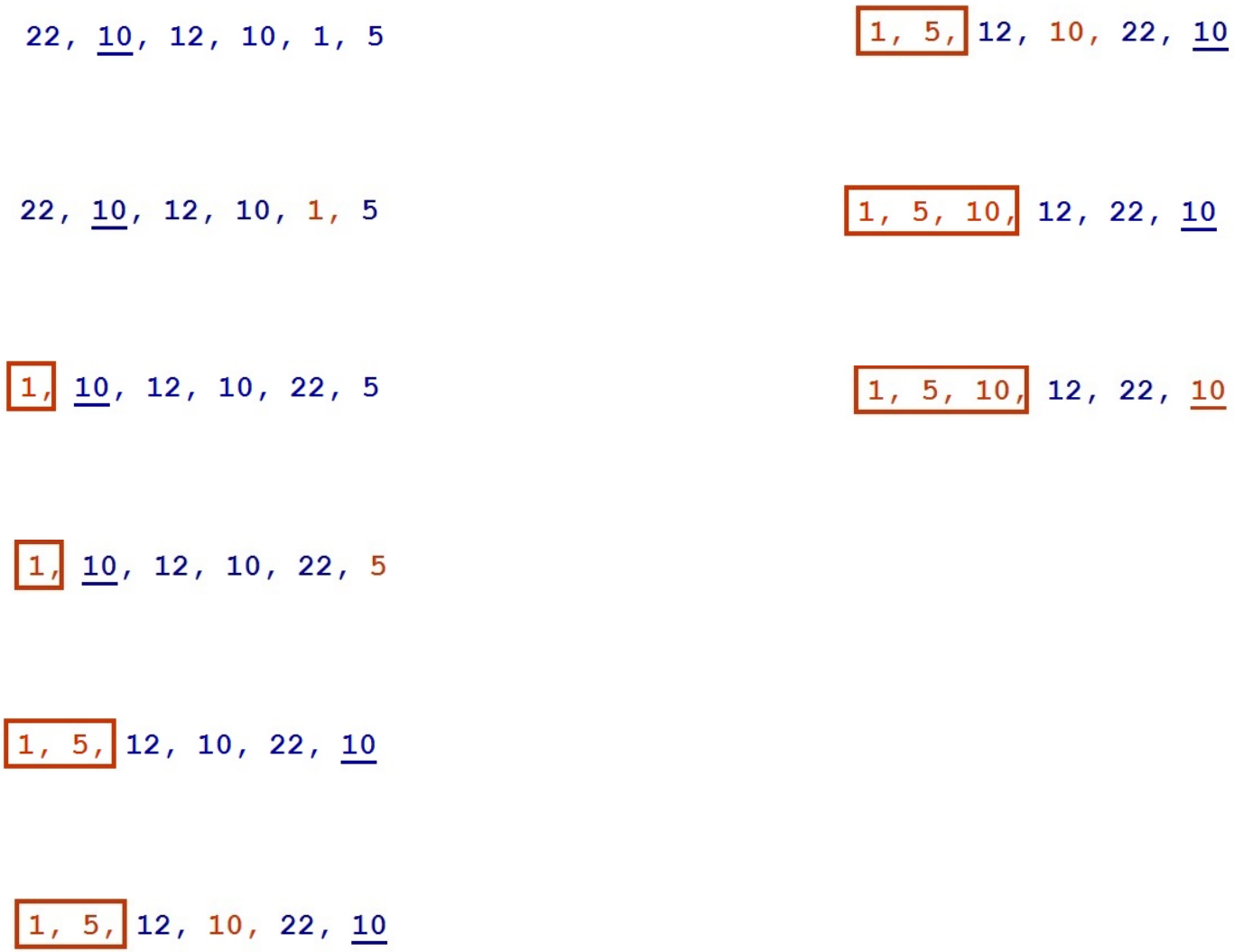
1, 5, 12, 10, 22, 10

1, 5, 12, 10, 22, 10

1, 5, 12, 10, 22, 10

1, 5, 10, 12, 22, 10

Método de Selección directa: Traza



Método de Selección directa: Traza

22, 10, 12, 10, 1, 5

22, 10, 12, 10, 1, 5

1, 10, 12, 10, 22, 5

1, 10, 12, 10, 22, 5

1, 5, 12, 10, 22, 10

1, 5, 12, 10, 22, 10

1, 5, 12, 10, 22, 10

1, 5, 10, 12, 22, 10

1, 5, 10, 12, 22, 10

1, 5, 10, 10, 22, 12

Método de Selección directa: Traza

22, 10, 12, 10, 1, 5

22, 10, 12, 10, 1, 5

1, 10, 12, 10, 22, 5

1, 10, 12, 10, 22, 5

1, 5, 12, 10, 22, 10

1, 5, 12, 10, 22, 10

1, 5, 12, 10, 22, 10

1, 5, 10, 12, 22, 10

1, 5, 10, 12, 22, 10

1, 5, 10, 10, 22, 12

1, 5, 10, 10, 22, 12

Método de Selección directa: Traza

22, 10, 12, 10, 1, 5

22, 10, 12, 10, 1, 5

1, 10, 12, 10, 22, 5

1, 10, 12, 10, 22, 5

1, 5, 12, 10, 22, 10

1, 5, 12, 10, 22, 10

1, 5, 12, 10, 22, 10

1, 5, 10, 12, 22, 10

1, 5, 10, 12, 22, 10

1, 5, 10, 10, 22, 12

1, 5, 10, 10, 22, 12

1, 5, 10, 10, 12, 22

Método de selección directa

```
def posicion_del_menor(v,inicio,fin):
  posicion = inicio
  menor = v[inicio]
  for i in range(inicio+1, fin+1):
    if (v[i] < menor):
      menor = v[i]
      posicion = i
  return posicion
```

Método de selección directa

```
def posicion_del_menor(v,inicio,fin):
  posicion = inicio
  menor = v[inicio]
  for i in range(inicio+1, fin+1):
    if (v[i] < menor):
      menor = v[i]
      posicion = i
  return posicion
```

```
def seleccion_directa (v,inicio,fin):
  for pasada in range(inicio,fin):
    pos = posicion_del_menor(v,pasada,fin);
    v[pos], v[pasada] =  v[pasada], v[pos]
```

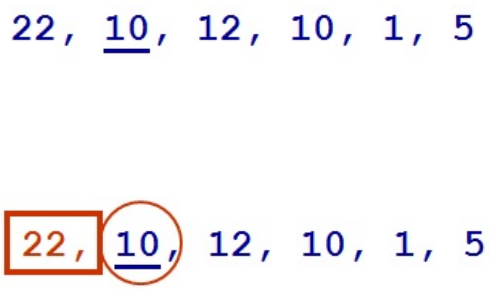
Método de inserción directa

- Se inserta en orden cada uno de los elementos del vector dentro de una subsecuencia ordenada.
- Tras la inserción, la subsecuencia debe quedar ordenada de nuevo.
- En la iteración i se inserta el elemento de la posición i + 1 de manera ordenada entre las posiciones 0 e i, creando una secuencia mayor

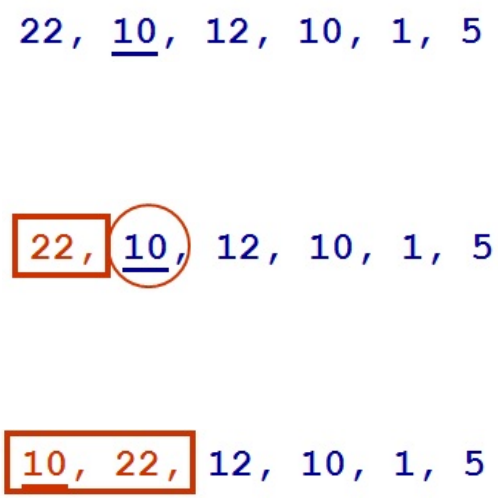
Método de inserción directa: Traza

22, 10, 12, 10, 1, 5

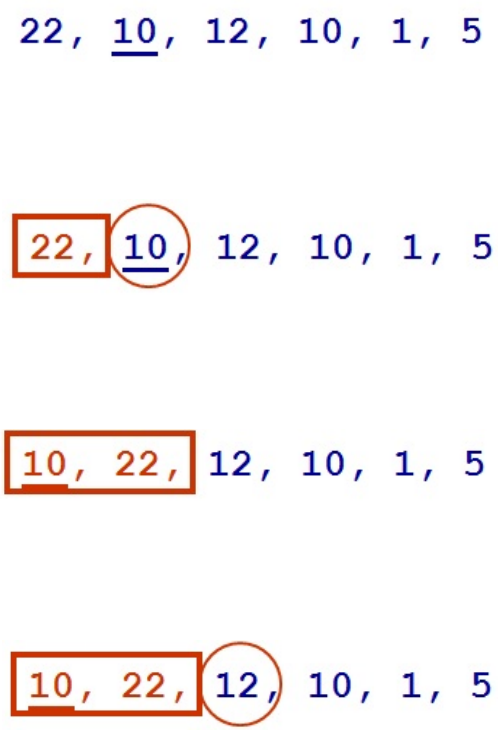
Método de inserción directa: Traza



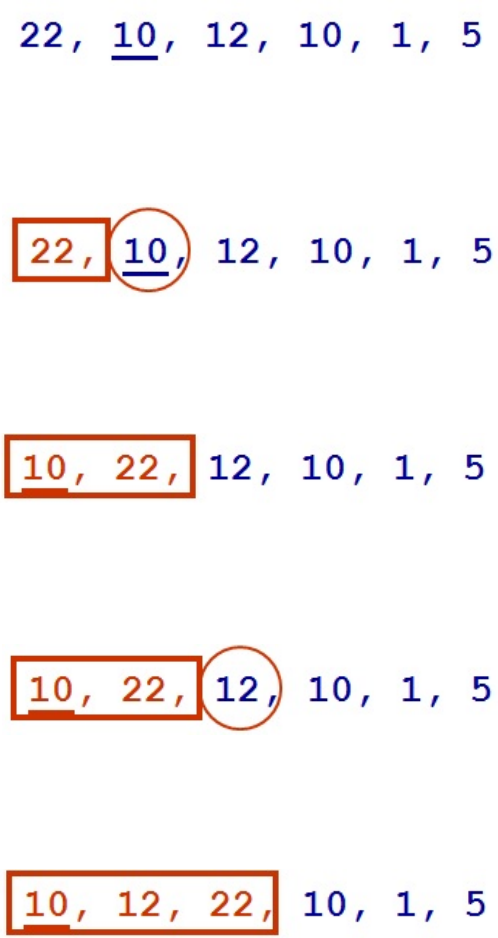
Método de inserción directa: Traza



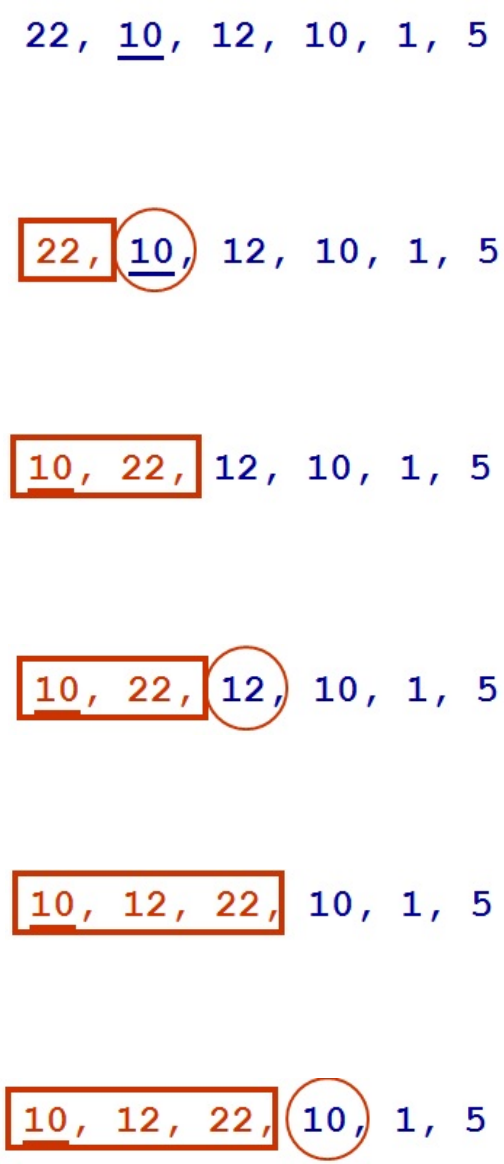
Método de inserción directa: Traza



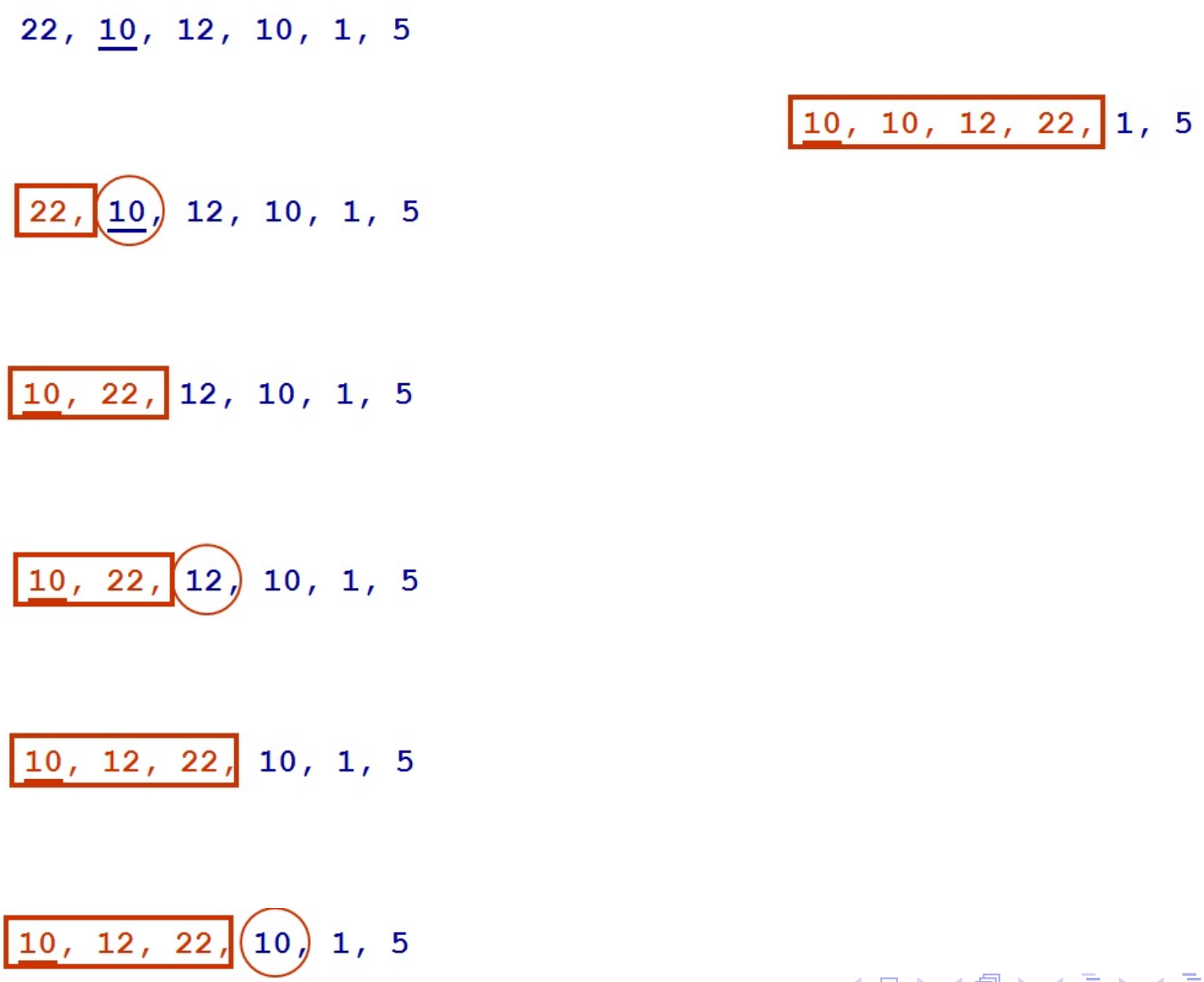
Método de inserción directa: Traza



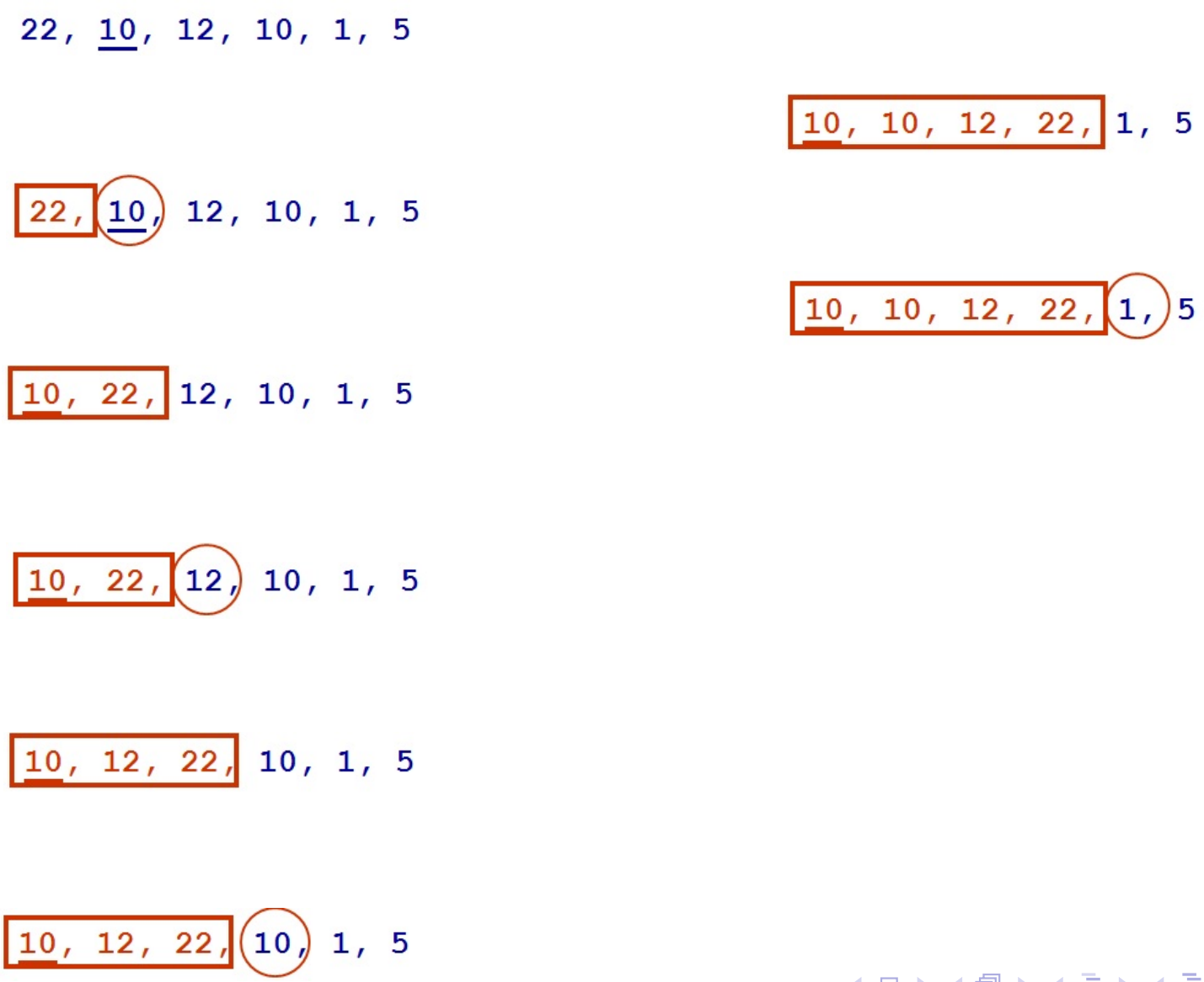
Método de inserción directa: Traza



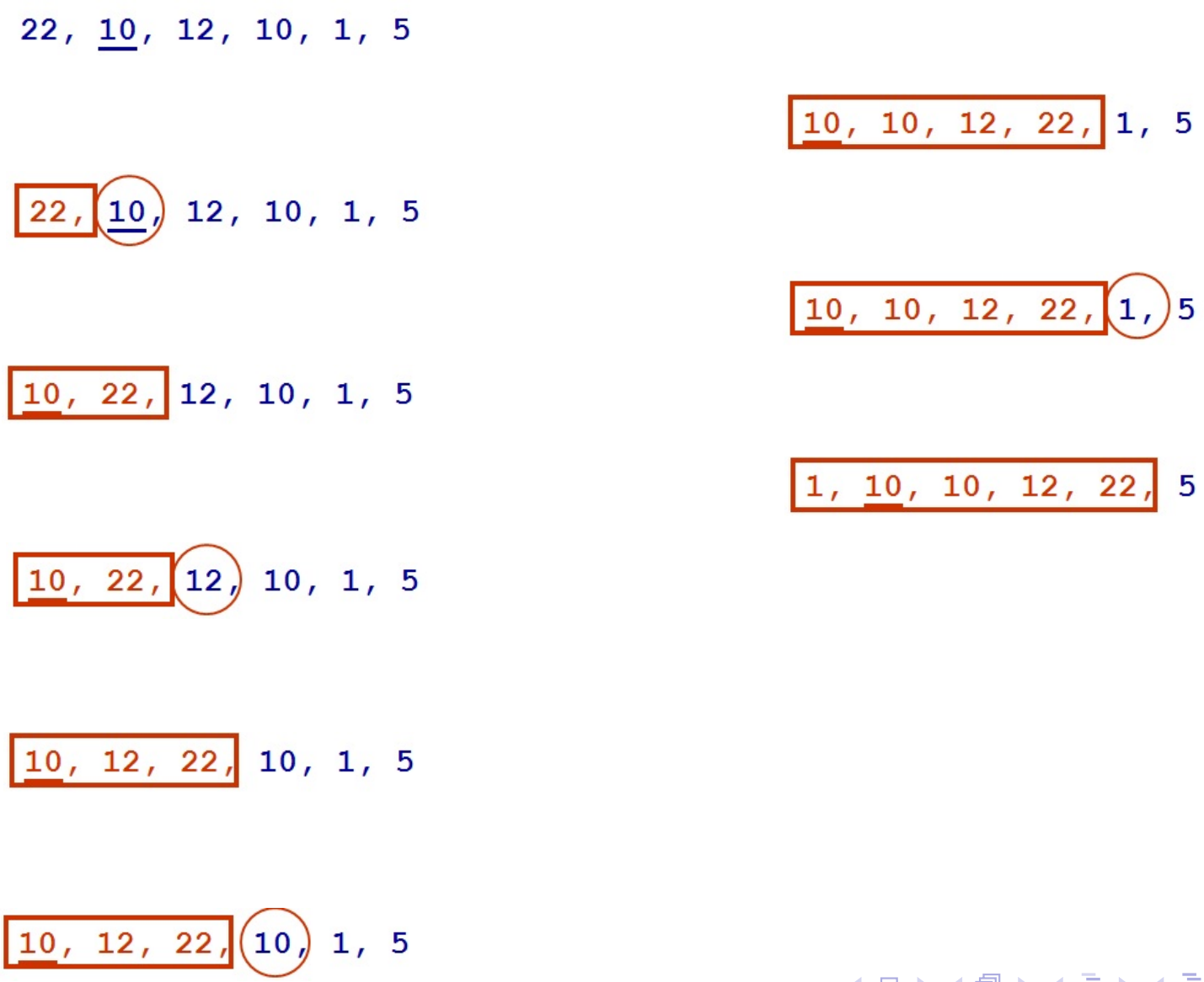
Método de inserción directa: Traza



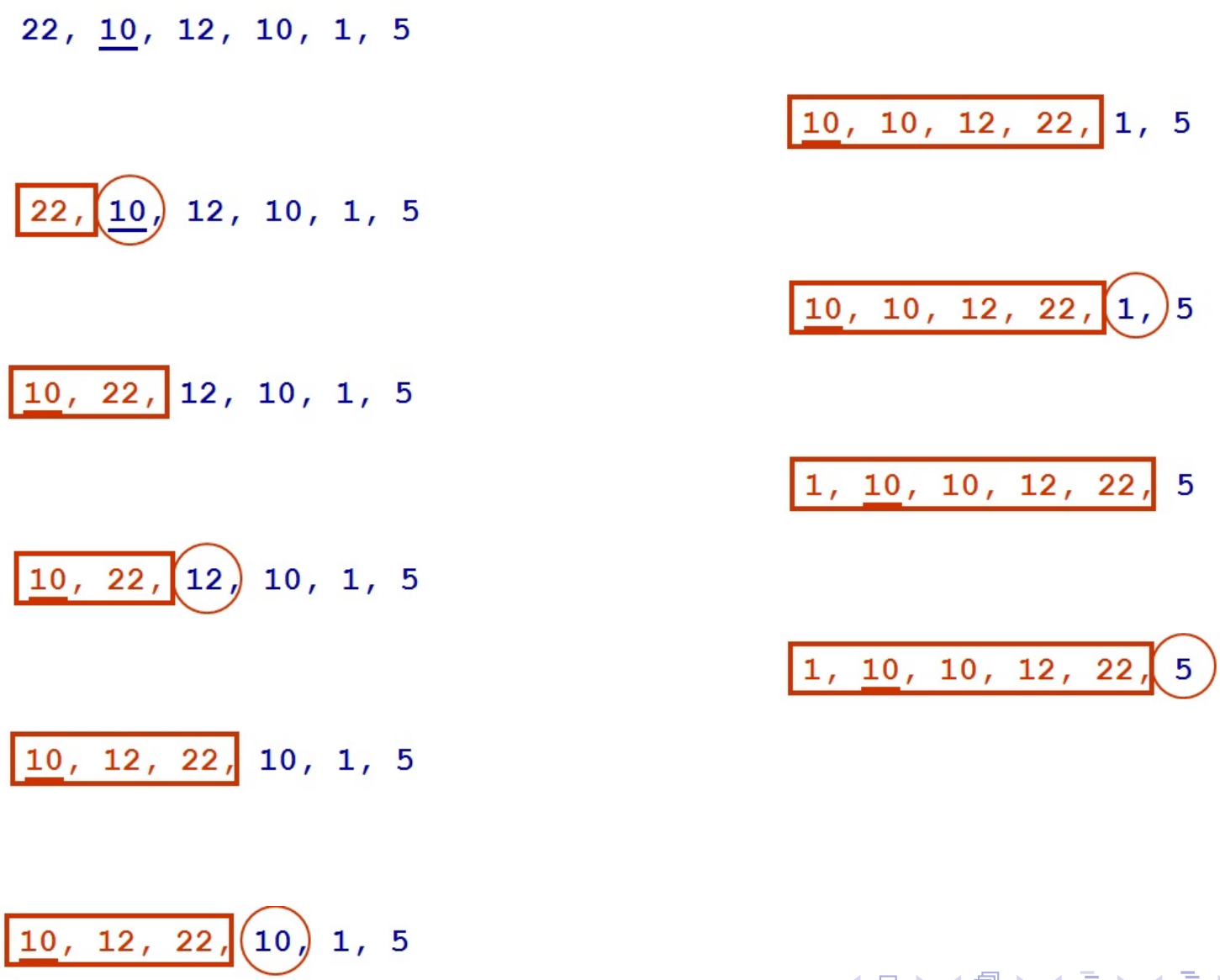
Método de inserción directa: Traza



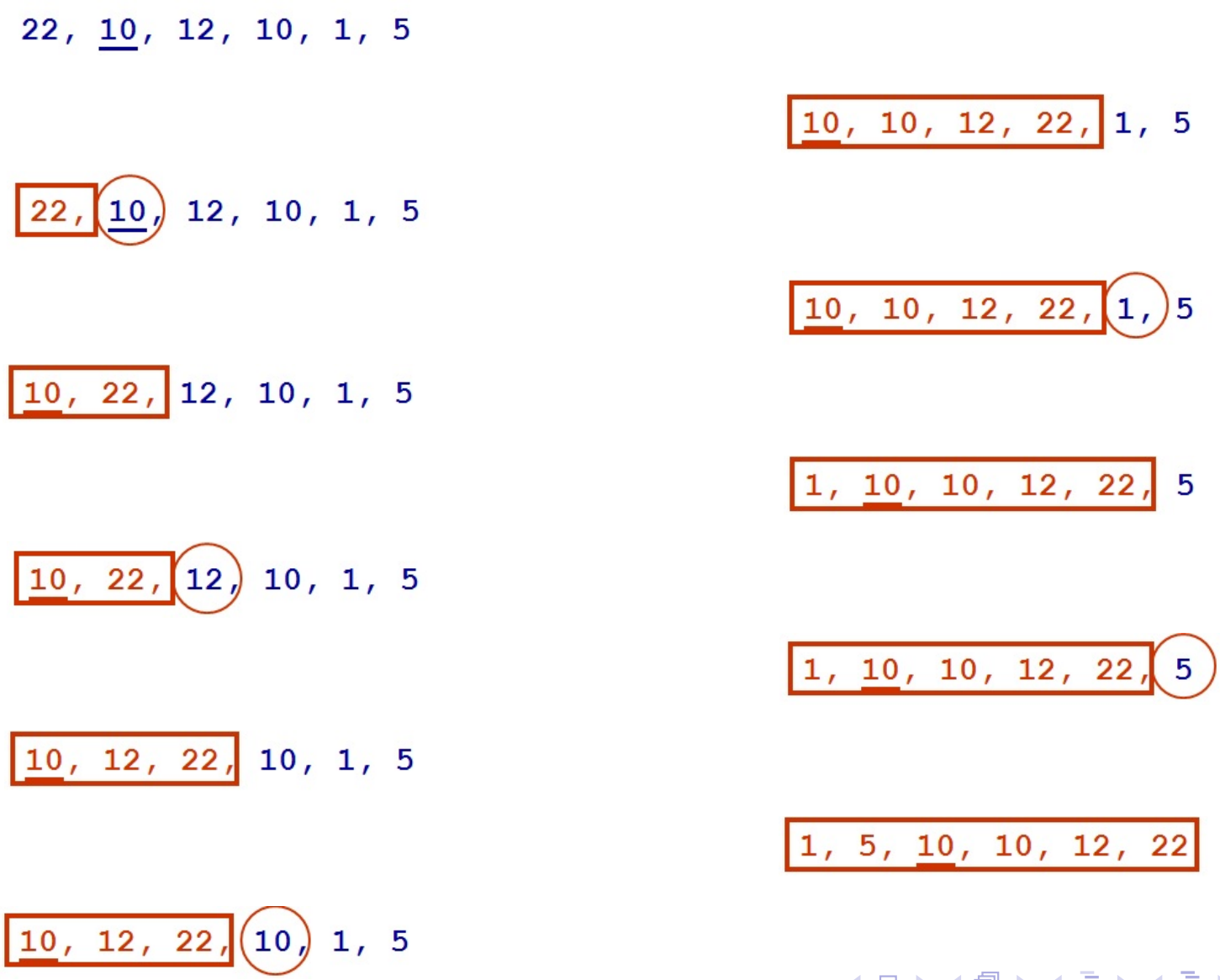
Método de inserción directa: Traza



Método de inserción directa: Traza



Método de inserción directa: Traza



Método de inserción directa

```
def insertar_ordenado(v,ini,fin,elem):
  i = fin
  while (v[i-1]>elem and i>0):
     v[i] = v[i-1]
     i -= 1
  v[i] = elem

def insercion_directa(v,inicio,fin):
  for pasada in range (inicio+1, fin+1):
    insertar_ordenado(v, inicio, pasada, v[pasada])
```

Complejidad asintótica

- La evaluación del tiempo de ejecución se lleva a cabo mediante la observación del comportamiento asintótico (notación O).
- Todos los algoritmos básicos de ordenación son $O(n^2)$.
- Las diferencias entre los métodos básicos radica en la cuenta de algunas operaciones, por ejemplo:
  - Número de comparaciones entre claves: Relevante cuando la comparación es costosa.
  - Número de movimientos de elementos: Relevante cuando el elemento es grande y hay que optimizar el uso de memoria.

Resumen

- Existen dos métodos para buscar elementos en una colección: búsqueda lineal y binaria.
- La búsqueda binaria es más rápida que la lineal.
- Se puede ordenar una colección utilizando métodos básicos o avanzados de ordenación.
- Los métodos básicos son menos eficientes pero muy útiles cuando la colección es pequeña.
- Los métodos básicos son: Burbuja, Inserción Directa y Selección Directa.

Referencias

- Iteresantes animaciones que muestran la evolución de una lista de números a medida que se ordena en función de los distintos métodos `http://interactivepython.org/runestone/static/pythonds/SortSearch/sorting.html`