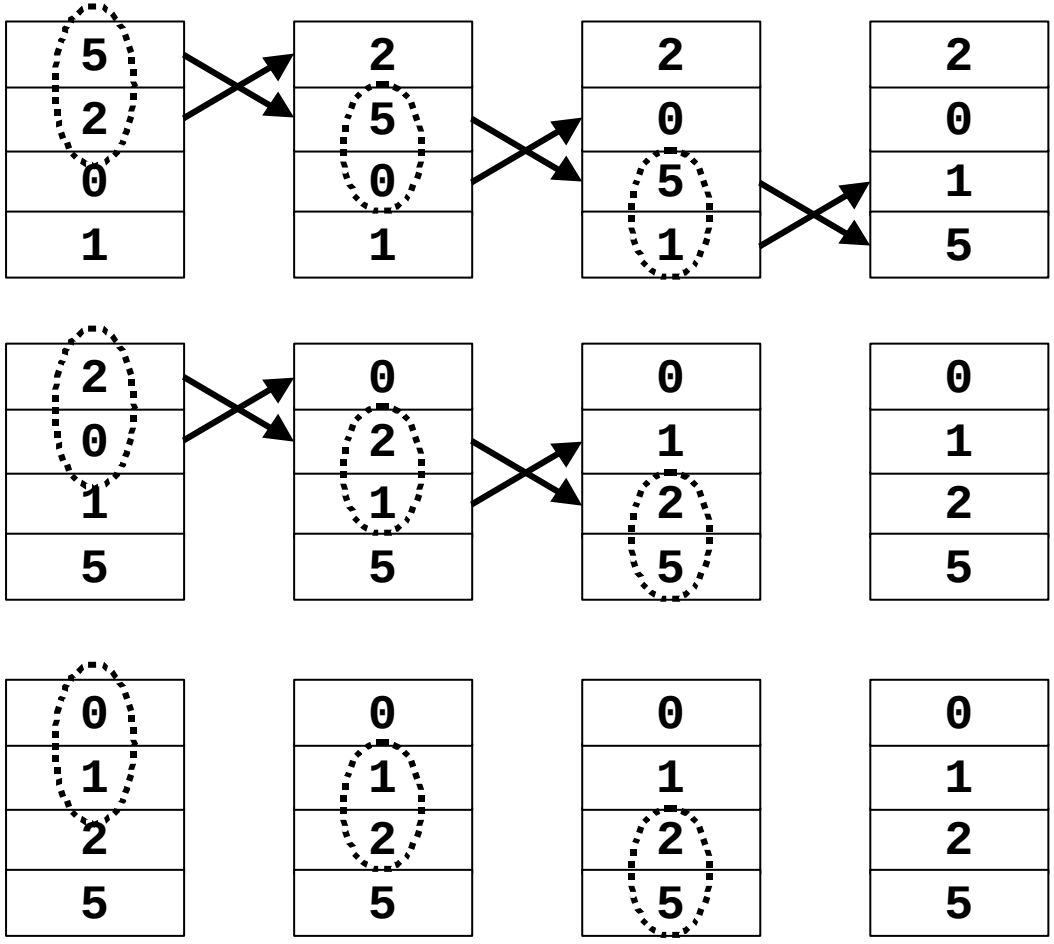
Bubble Sort

Se l'i-esimo elemento è maggiore dell'elemento i+1-esimo, essi vengono scambiati.
Si termina quando non vi sono più scambi da fare.



Nessuno scambio effettuato:
→ Il vettore è ordinato

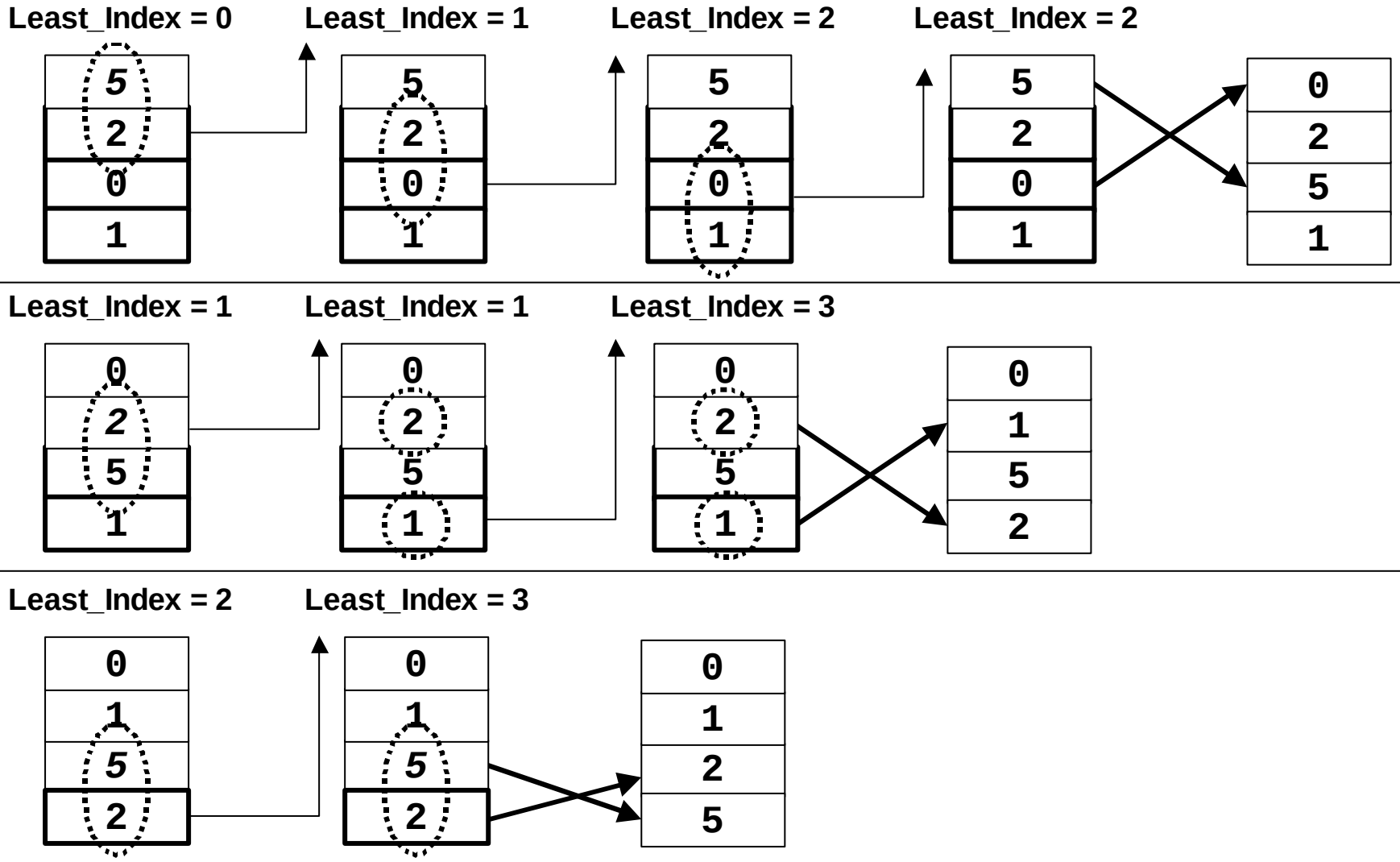
Bubble Sort

```
void bubble_sort (t_vector v)
{
  int i, aux;
  char swap_done;

  do {
    swap_done = 0;
    for (i = 0; i < v.size - 1; i++)
      if (v.data [i] > v.data [i + 1]) {
        // scambiamo gli elementi i e i+1
        aux = v.data [i];
        v.data [i] = v.data [i + 1];
        v.data [i + 1] = aux;
        // segnaliamo che c'e' stato uno scambio
        swap_done = 1;
      }
  }
  while (swap_done);
}
```

Select Sort

Per ogni elemento i-esimo, si seleziona (select), nel sotto-vettore (i, n), la posizione k dell'elemento più piccolo.
Si scambiano quindi gli elementi i-esimo e k-esimo fra loro.



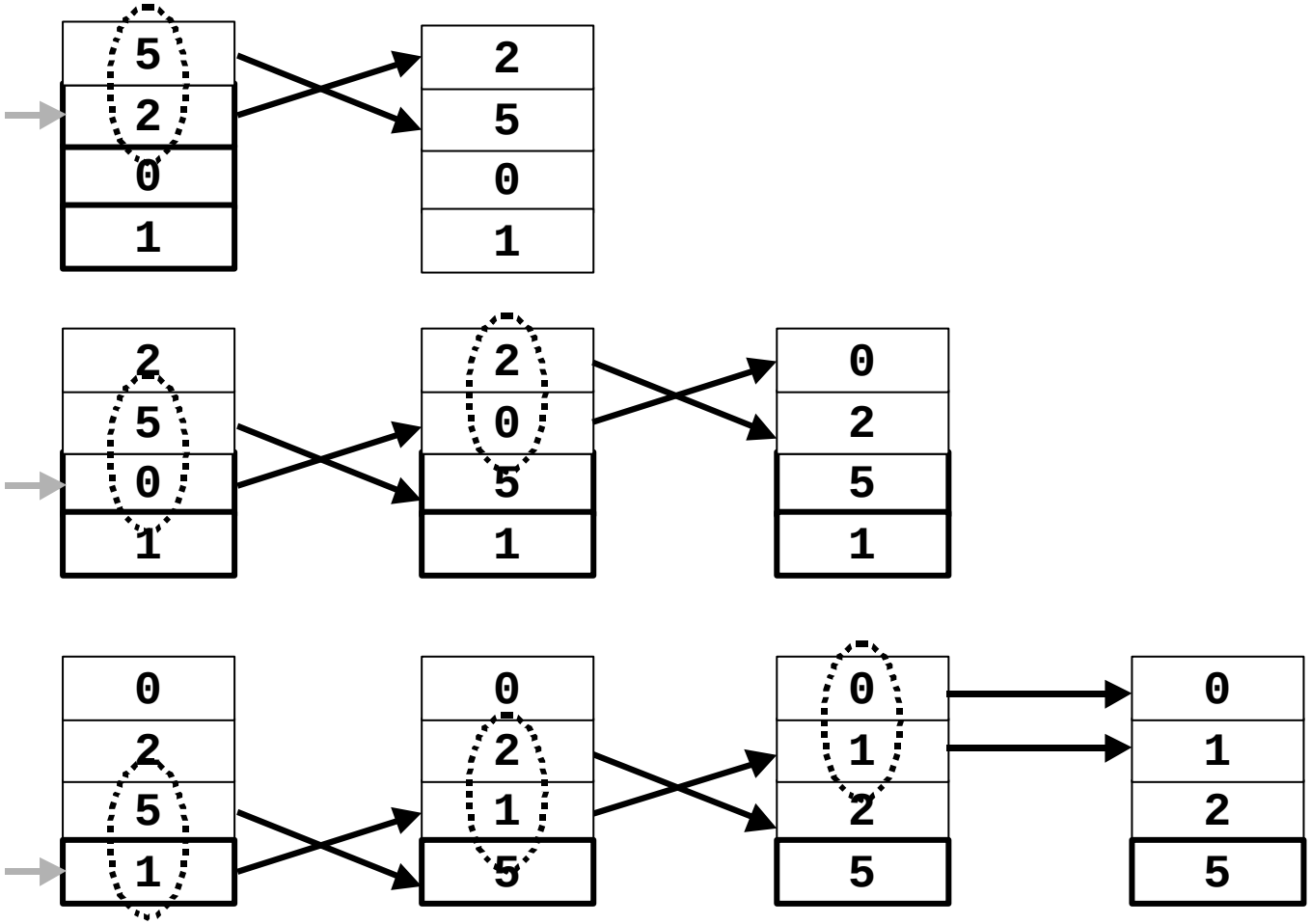
Select Sort

```
void select_sort (t_vector v)
{
  int i, j, least_index, aux;

  for (i = 0;i < v.size - 1;i++) {
    least_index = i;
    for (j = i + 1;j < v.size;j++) {
      if (v.data [least_index] > v.data [j])
        least_index = j;
    }
    aux = v.data [i];
    v.data [i] = v.data [least_index];
    v.data [least_index] = aux;
  }
}
```

Insert Sort

Ogni elemento i-esimo (a partire dal secondo) lo si posiziona (inserisce) al posto giusto nel sotto-vettore [1, i -1]

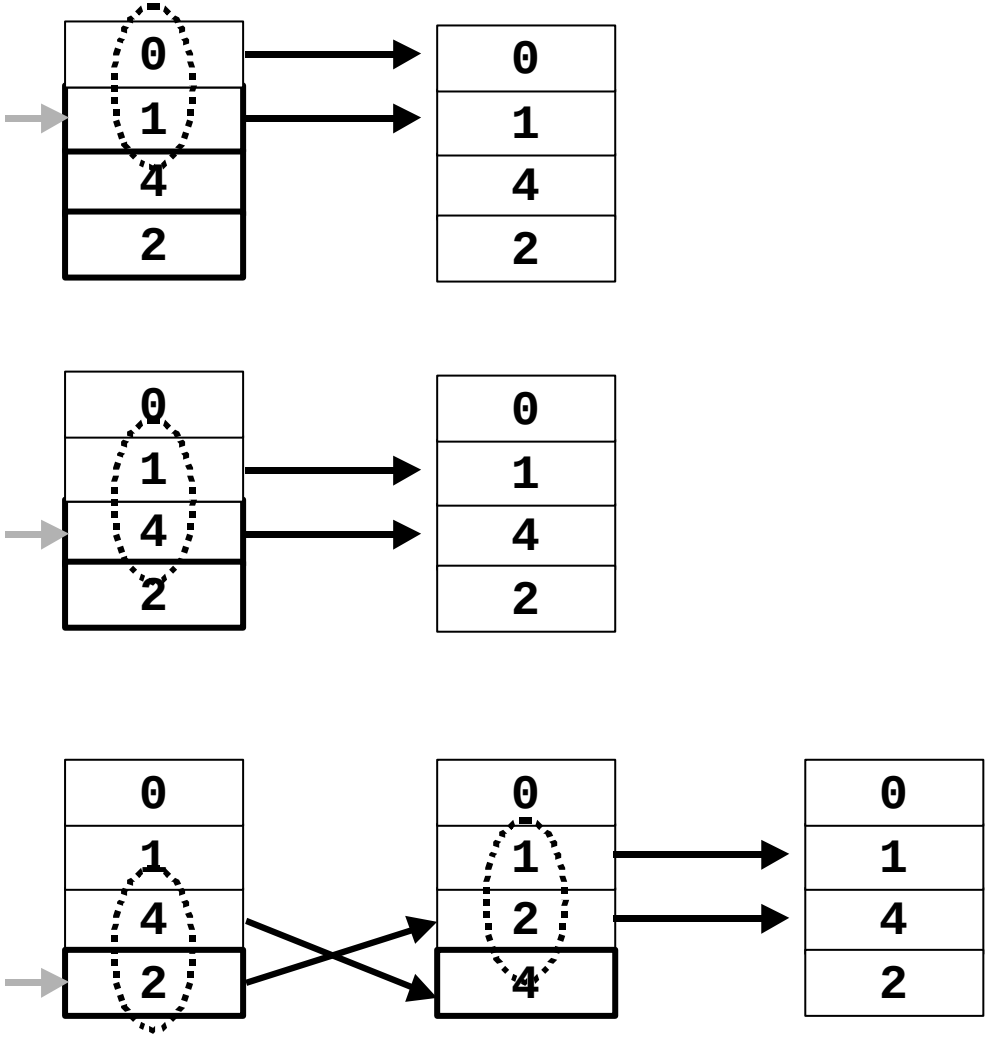


Insert Sort

```
void insert_sort (t_vector v)
{
  int i, j, aux;

  for (i = 1; i < v.size;i++) {
    for (j = i; j >= 1; j--) {
      if (v.data [j] < v.data [j - 1]) {
        aux = v.data [j];
        v.data [j] = v.data [j - 1];
        v.data [j - 1] = aux;
      }
      else
        break;
    }
  }
}
```

Insert Sort (su un vettore “quasi” ordinato)



Sono stati necessari solo 4 passi
non n*(n-1) = 4*3 = 12 passi

Quick Sort

E’ un algoritmo **efficiente** che si basa sulla suddivisione dell’array in **sotto-array**, riordinando questi ultimi.

Gli step sono i seguenti:

1. A partire da un array, si individua un **elemento di separazione**
2. A partire dall’array originale, si costruiscono due sotto-array: il primo contiene tutti gli elementi *minori* dell’elemento di separazione, il secondo contiene tutti gli elementi *maggiori* dell’elemento di separazione
3. Si riapplicano i passi 1 e 2 ai due sotto-array.
4. Se un sotto-array e’ costituito da 3 o meno elementi, si ordinano gli elementi tra loro.

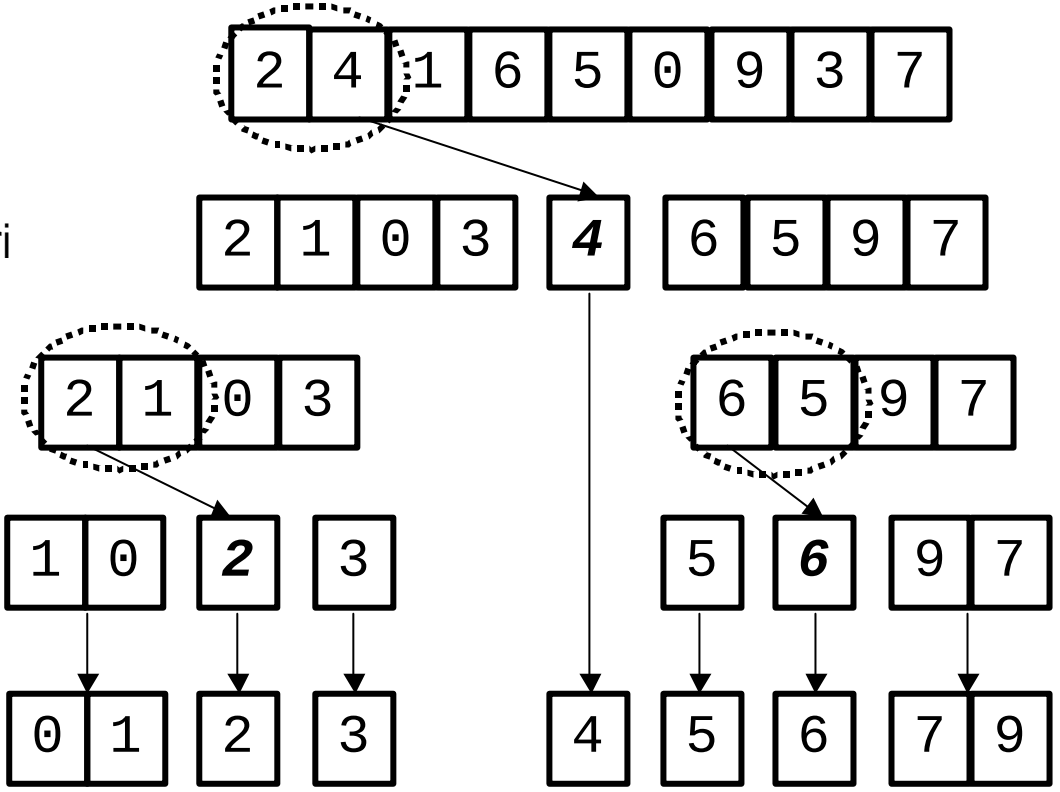
Quick Sort/1

Si determina il **pivot**
(l'elemento piu' grande)
PIVOT = 4

Si creano i due sotto-vettori

Si riapplica il
procedimento

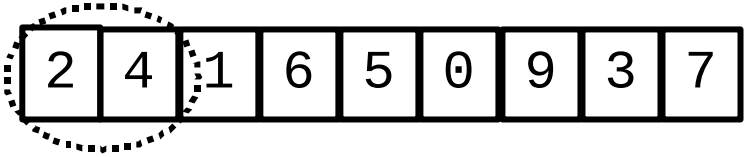
Alla fine abbiamo il vettore ordinato



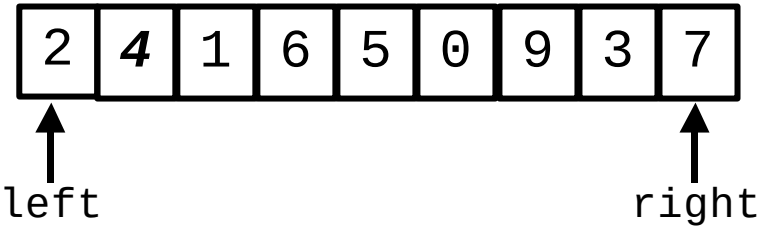
La procedura di **separazione** e' il fulcro dell'algoritmo

Quick Sort/2

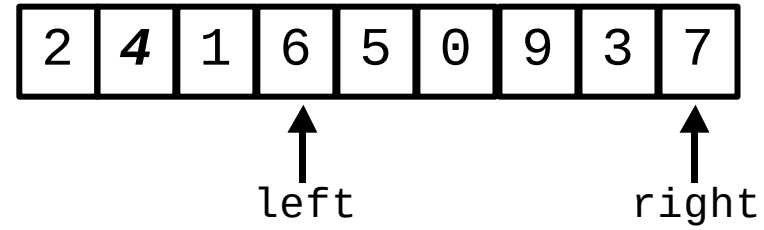
Si determina il **pivot** e il **pivotindex:**
PIVOT = 4,
PIVOTINDEX = 1



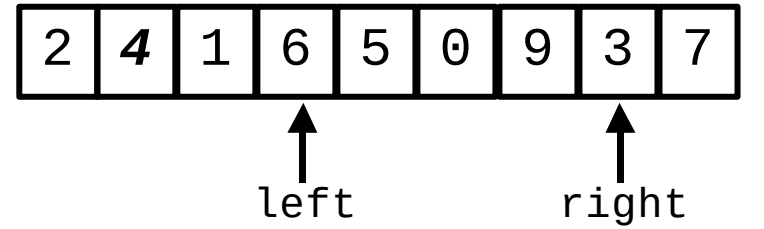
Si incrementa **left** fino a che non si trova un elemento maggiore del pivot

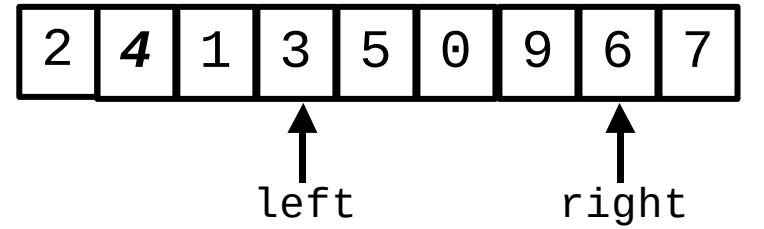


Si decrementa **right** fino a che non si trova un elemento minore o uguale al pivot



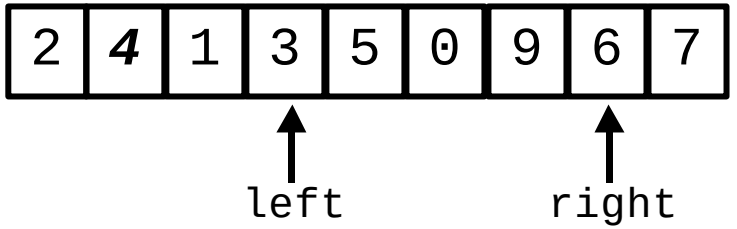
Si scambiano di posto gli elementi indicizzati dal **left** e **right**



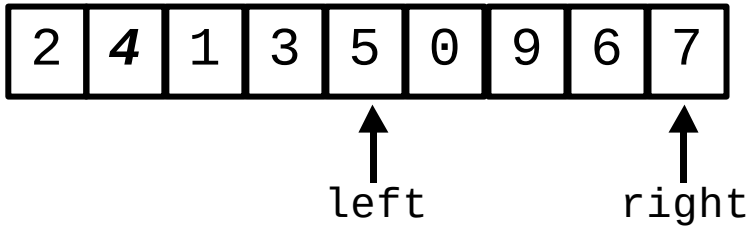


Quick Sort/3

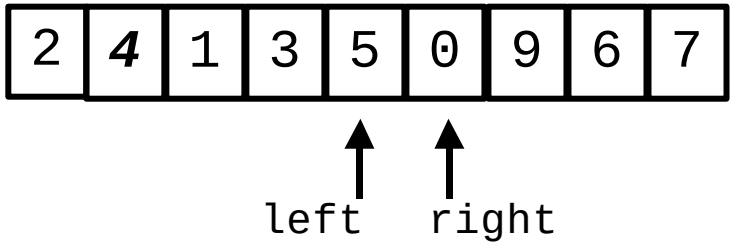
Si incrementa **left** fino a che non si trova un elemento maggiore del pivot

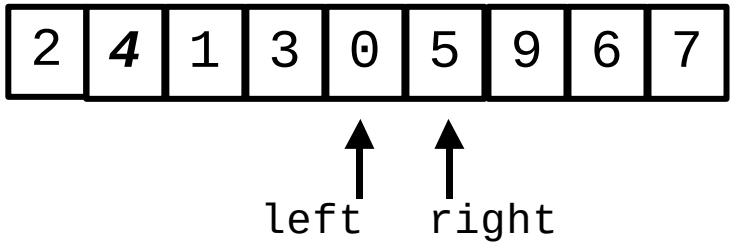


Si decrementa **right** fino a che non si trova un elemento minore o uguale al pivot



Si scambiano di posto gli elementi indicizzati dal **left** e **right**





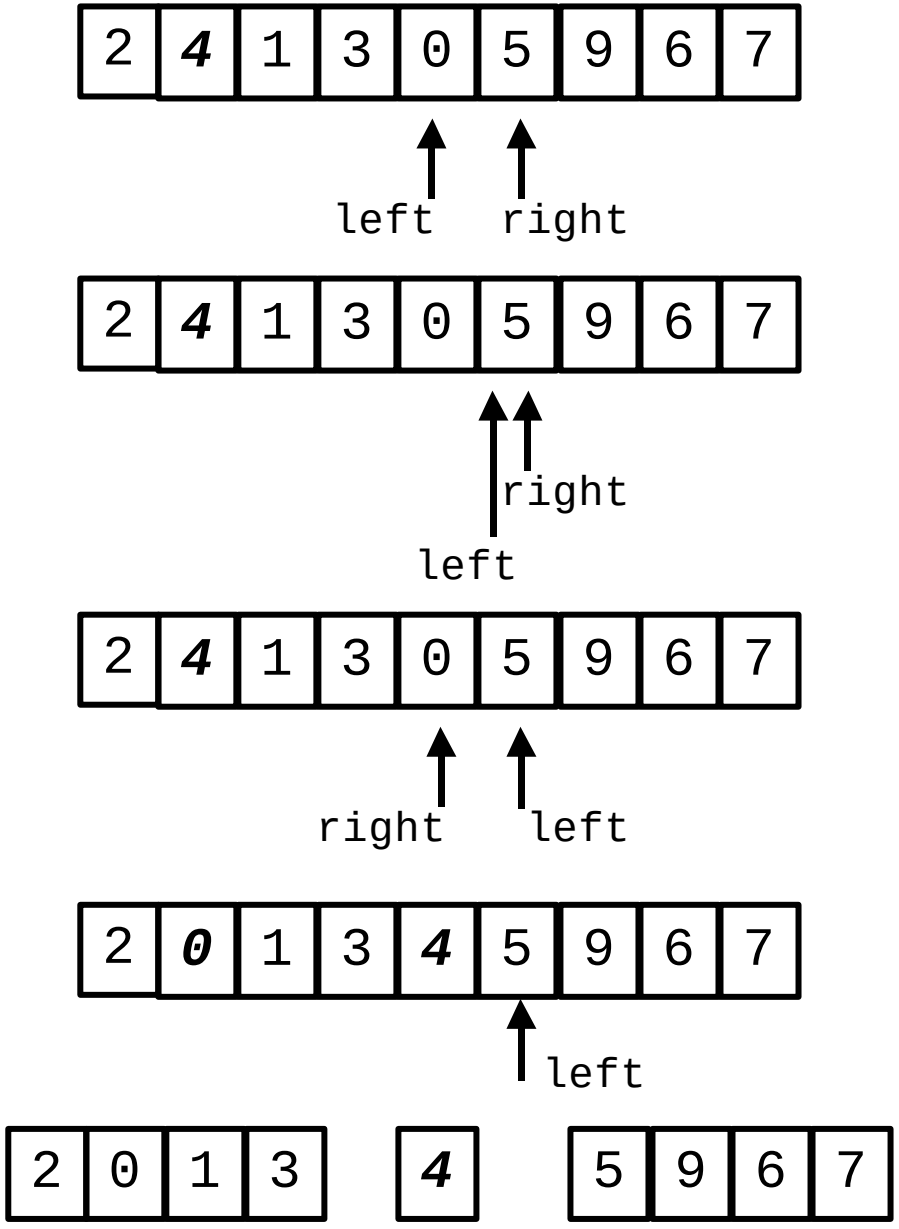
Quick Sort/4

Si incrementa **left** fino a che non si trova un elemento maggiore del pivot

Si decrementa **right** fino a che non si trova un elemento minore o uguale al pivot

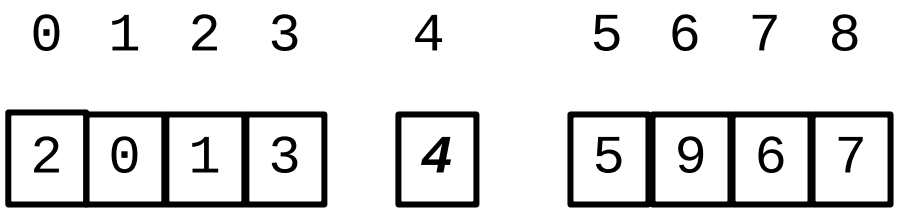
left e **right** si sono “scambiati di posto” (il primo adesso e’ piu’ avanti del secondo)

Si scambiano di posto il **pivot** e l’elemento indicato da **left – 1**, e si separano i vettori all’indice **left – 1**



Quick Sort/5

Si ordinano i due sotto-vettori di indici:
0 – 3
5 – 8
eseguendo nuovamente gli step dell'algoritmo




```
void quicksort (int * vector, int low, int high)
{
  int split;
  if (low < high)
  {
    split = partition (vector, low, high);
    quicksort (vector, low, split - 1);
    quicksort (vector, split + 1, high);
  }
}

int main(int argc, char * argv[])
{
#define SIZE 10
  int v [SIZE];
...
  quicksort (v, 0, SIZE – 1);
...
}
```

Quick Sort/6

```
int partition (int * vector, int low, int high)
{
  int left, right, pivot, pivotindex;

  if (low == high)
    return low;
  if (vector [low] > vector [low + 1])
    pivotindex = low;
  else
    pivotindex = low + 1;

  pivot = vector [pivotindex];

...
```

Quick Sort/7

```
...

  left = low;
  right = high;

  while (left <= right) {
    while ((vector [left] <= pivot) && (left <= right))
      ++left;

    while ((vector [right] > pivot) && (left <= right))
      --right;

    if (left < right)
    {
      swap (vector, left, right);
    }
  }

  swap (vector, left - 1, pivotindex);

  return left - 1;
}
```

Quick Sort/8

```
void swap (int * vector, int p1, int p2)
{
  int aux;
  aux = vector [p1];
  vector [p1] = vector [p2];
  vector [p2] = aux;
}
```