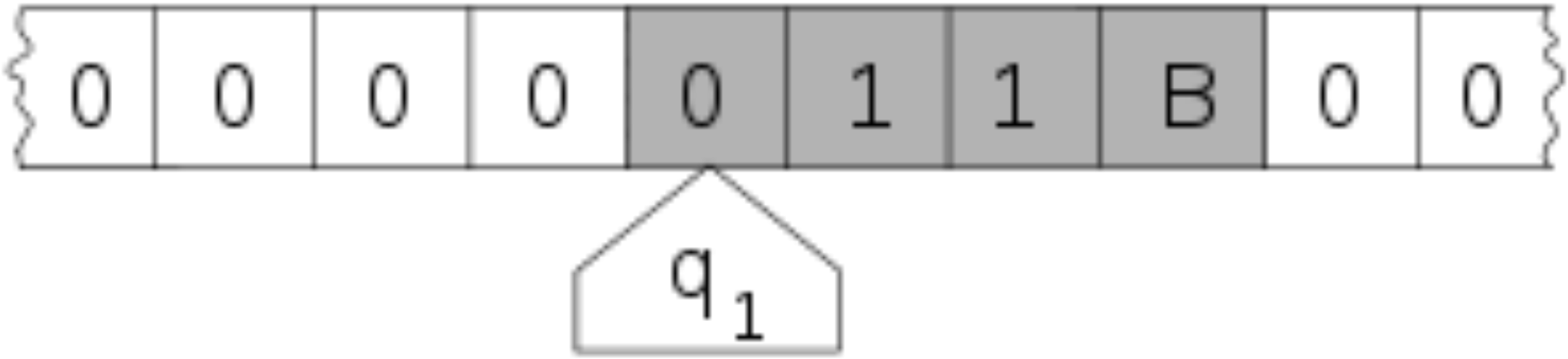
CSCI 1102 Computer Science 2

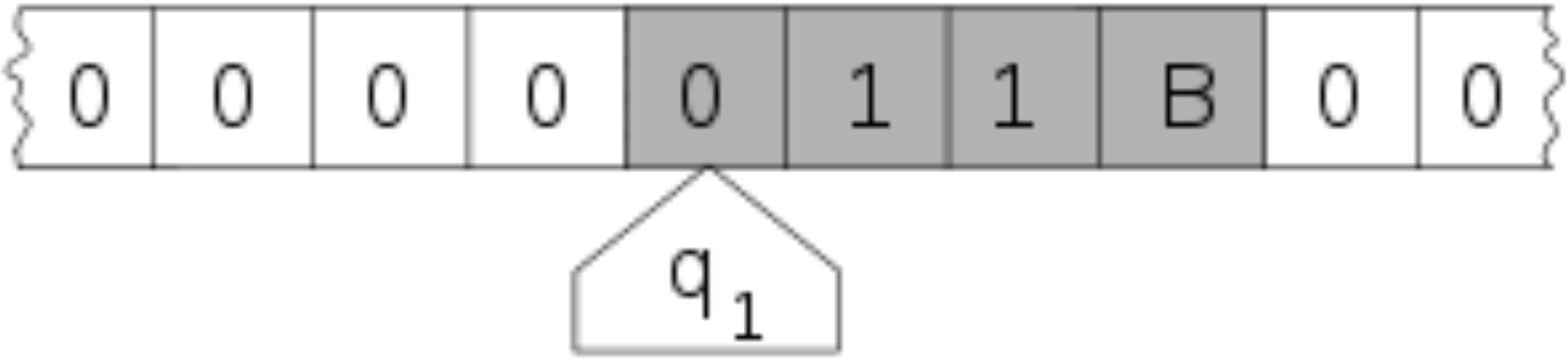
Meeting 14: Thursday 3/18/2021

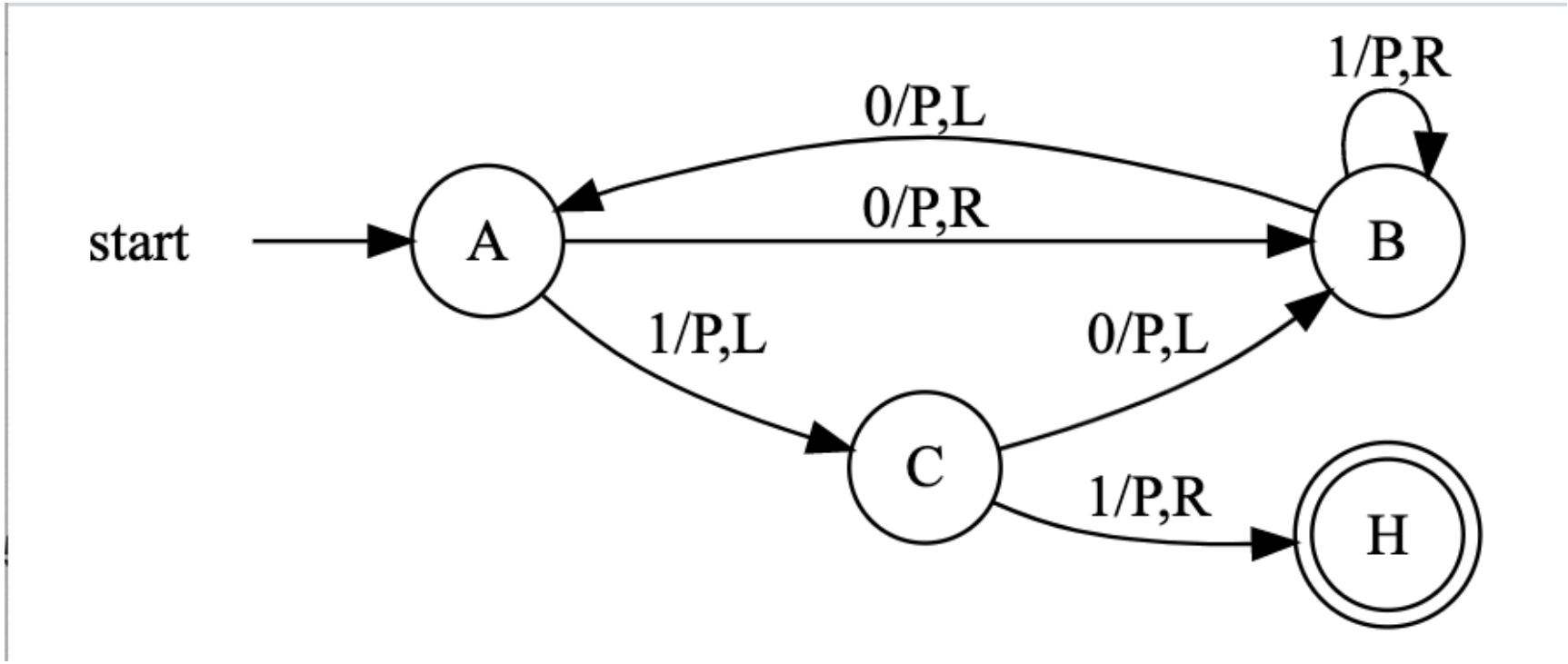
More Sorting; Order & Equality

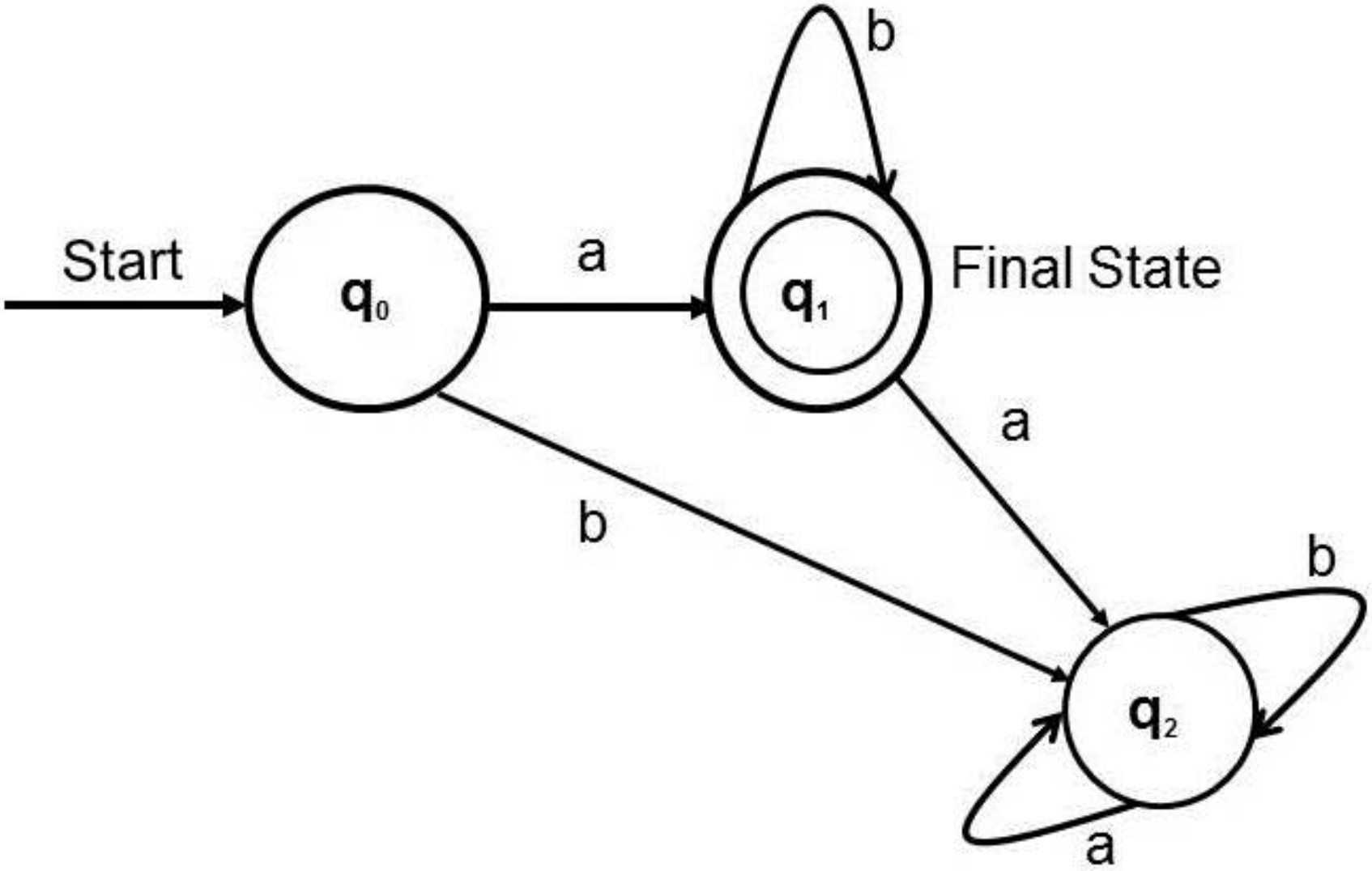

Alan Turing

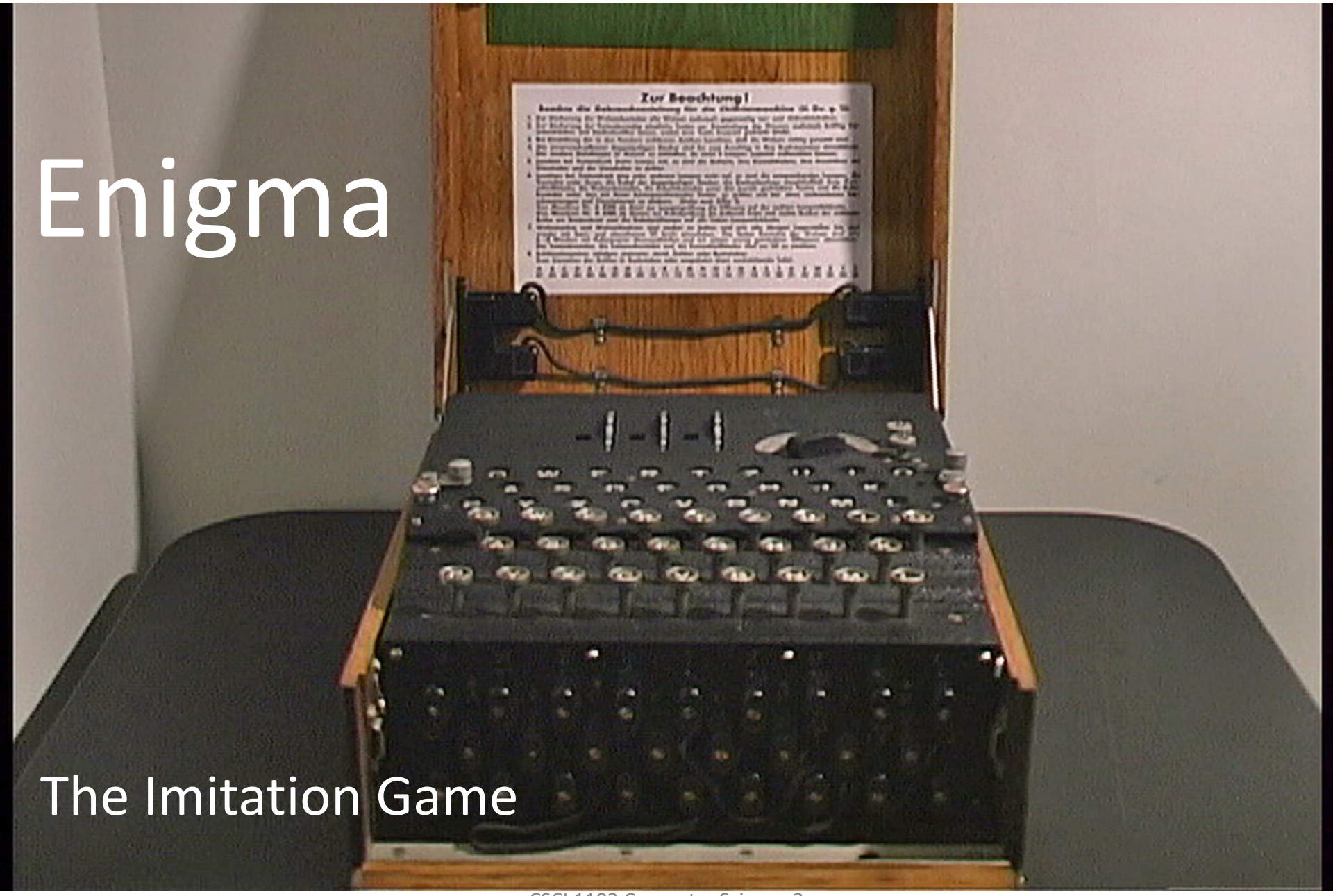


A Machine Model for Effective Computability


0
0
0
0
0
1
1
B
0
0
q 1


start
A
B
C
H
0/P,L
0/P,R
1/P,R
1/P,L
0/P,L
1/P,R


Start
q_0
a
q_1
Final State
b
a
b
b
q_2
a


Enigma
Zur Beachtung!
The Imitation Game

Alan Turing

- Turing Machine
- Universal Machine
- Enigma Cypher Machine
- Turing Test in AI
- Turing Award

Today

Quicksort & Insertion Sort

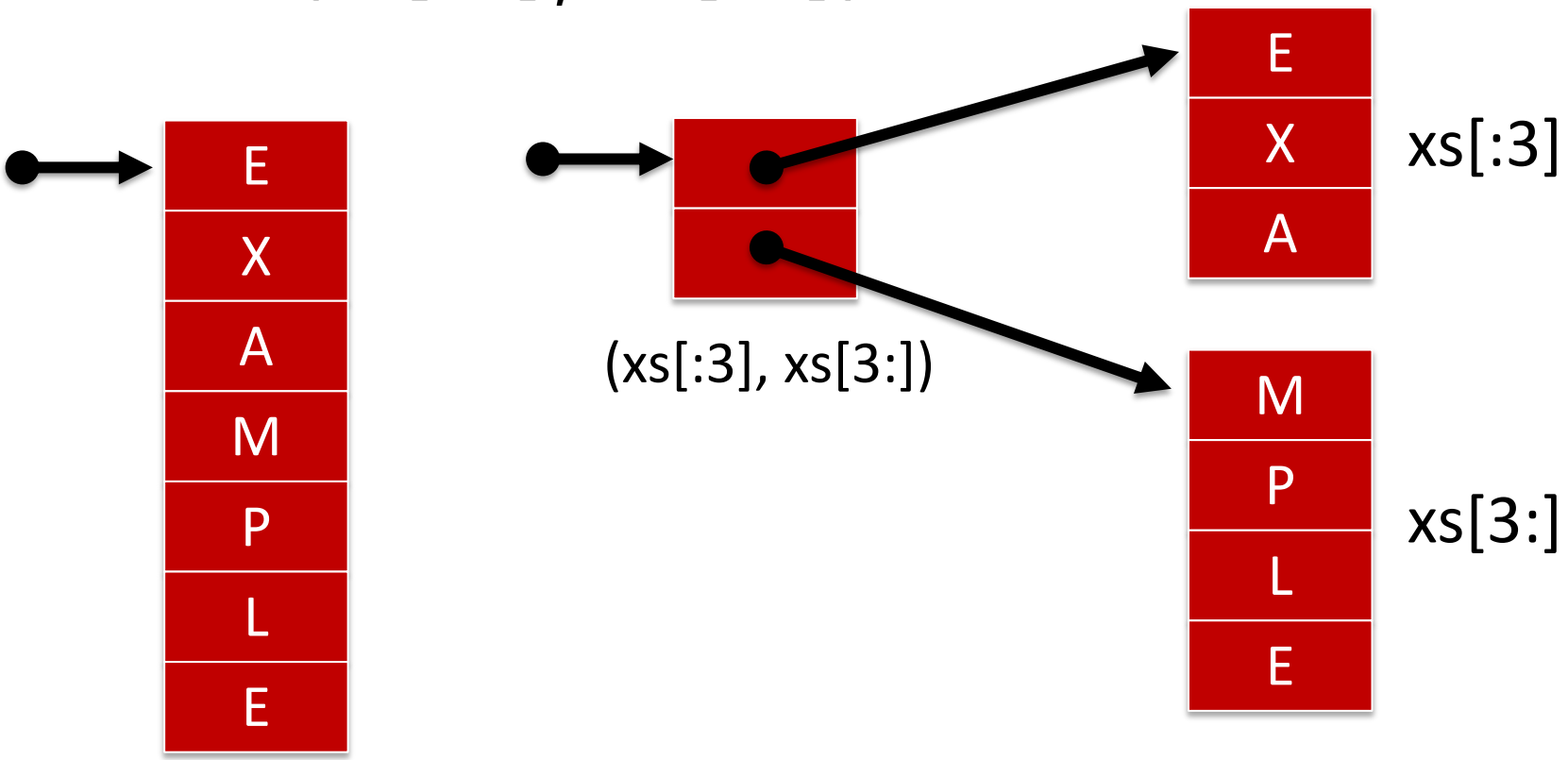
Order & Equality

Mergesort

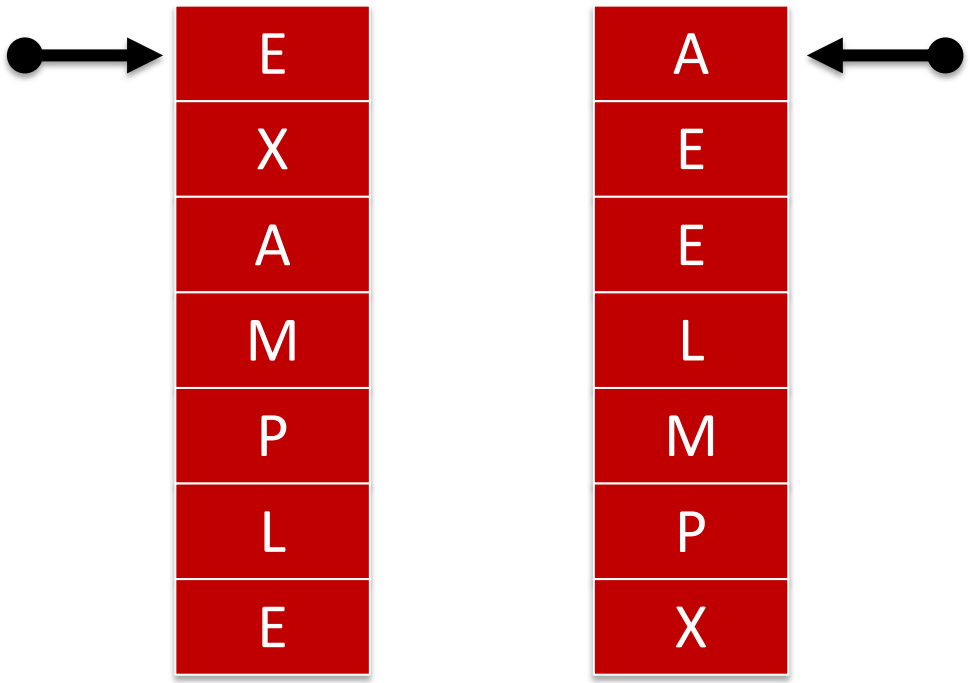
John von Neumann, 1945

Memory Usage in Python

```
17
18 # split the list xs in half.
19 def split(xs, n):
20   return (xs[:n], xs[n:])
21
```



Memory Usage



Use another array as a work area.

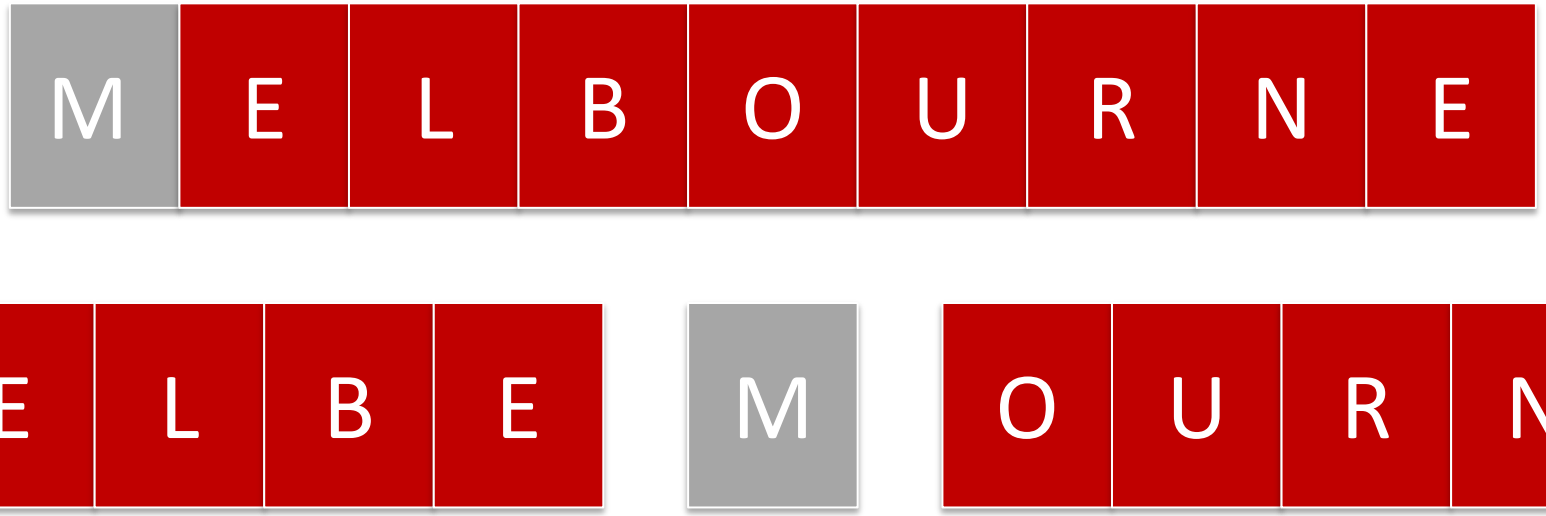
```java
// mergesort a[lo..hi] using auxiliary array aux[lo..hi]
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
    if (hi <= lo) return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid + 1, hi);
    merge(a, aux, lo, mid, hi);
}

/**
 * Rearranges the array in ascending order, using the natural order.
 * @param a the array to be sorted
 */
public static void sort(Comparable[] a) {
    Comparable[] aux = new Comparable[a.length];
    sort(a, aux, 0, a.length-1);
    assert isSorted(a);
}
```

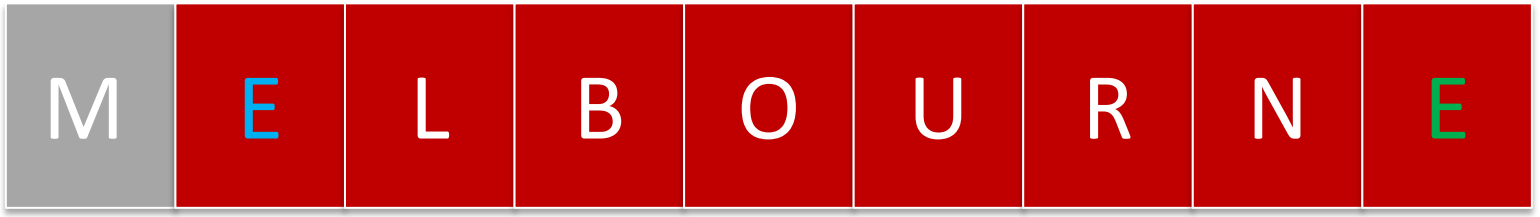
```java
// stably merge a[lo .. mid] with a[mid+1 ..hi] using aux[lo .. hi]
private static void merge(Comparable[] a, Comparable[] aux, int lo, int mid, int hi) {

  // copy to aux[]
  for (int k = lo; k <= hi; k++)
    aux[k] = a[k];

  // merge back to a[]
  int i = lo, j = mid + 1;
  for (int k = lo; k <= hi; k++) {
    if (i > mid) a[k] = aux[j++];
    else if (j > hi) a[k] = aux[i++];
    else if (less(aux[j], aux[i])) a[k] = aux[j++];
    else a[k] = aux[i++];
  }
}
```

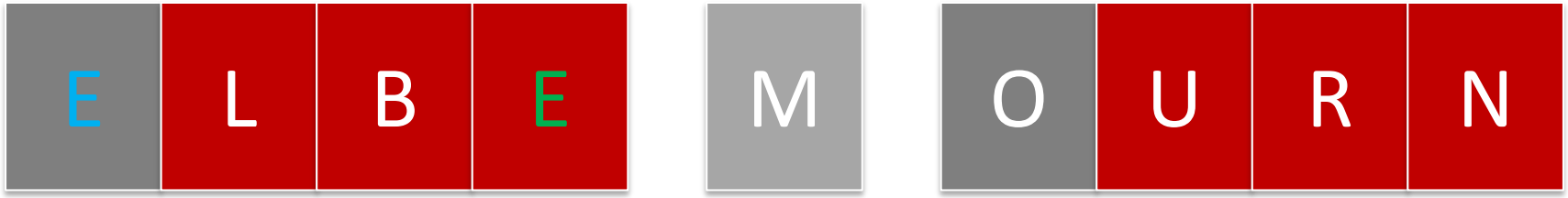
Quicksort

Tony Hoare, 1959

Divide-and-Conquer




M E L B O U R N E


E L B E
M
O U R N

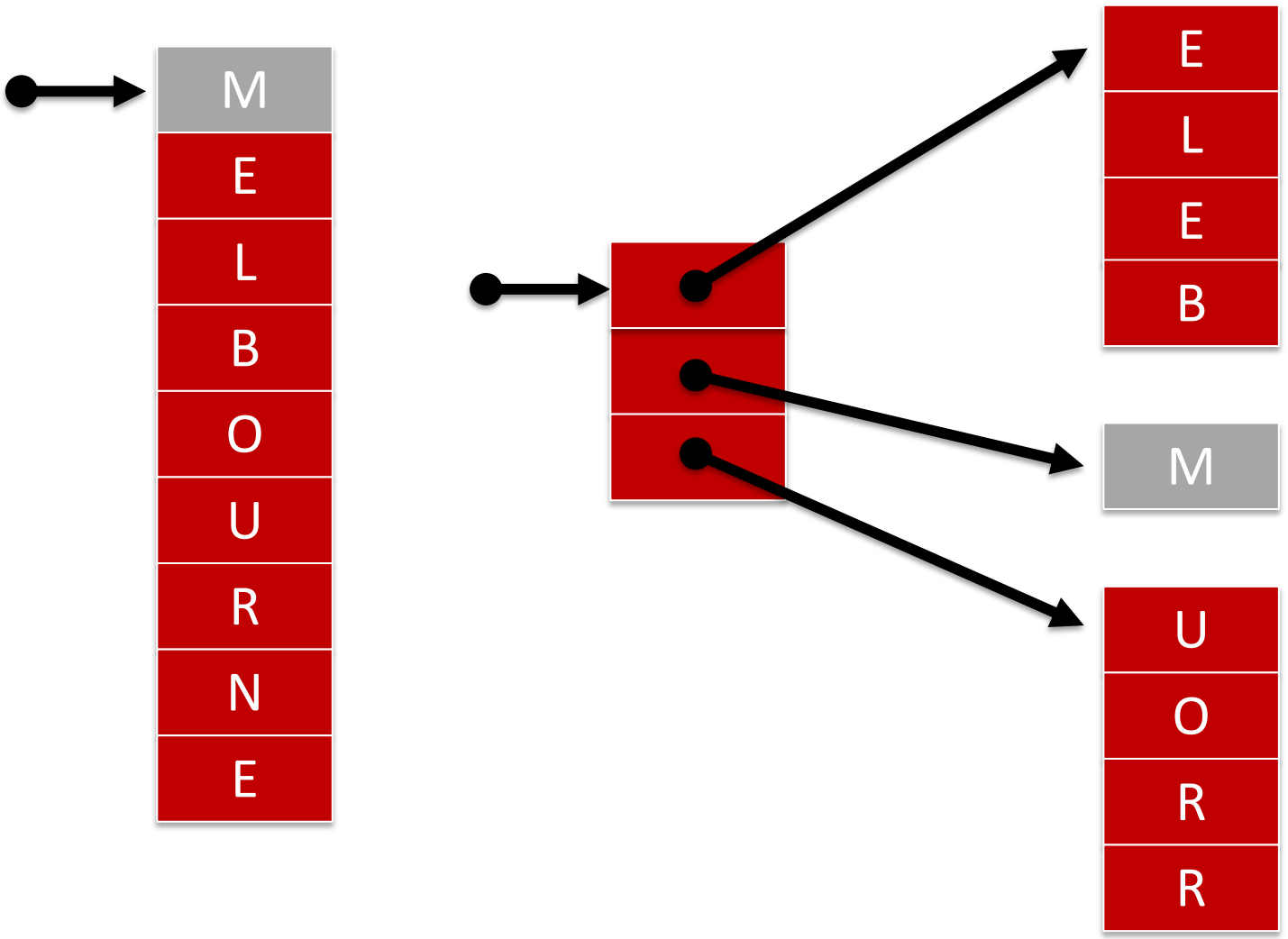

B
E E
L
N
O
U R

```
45
46 # Quicksort - Tony Hoare, 1959
47 #
48 def qsort(xs):
49   n = len(xs)
50   if n < 2:
51     return xs
52   else:
53     pivot = xs[0]
54     (smaller, equal, greater) = partition(xs, pivot)
55     return qsort(smaller) + equal + qsort(greater)
56
```

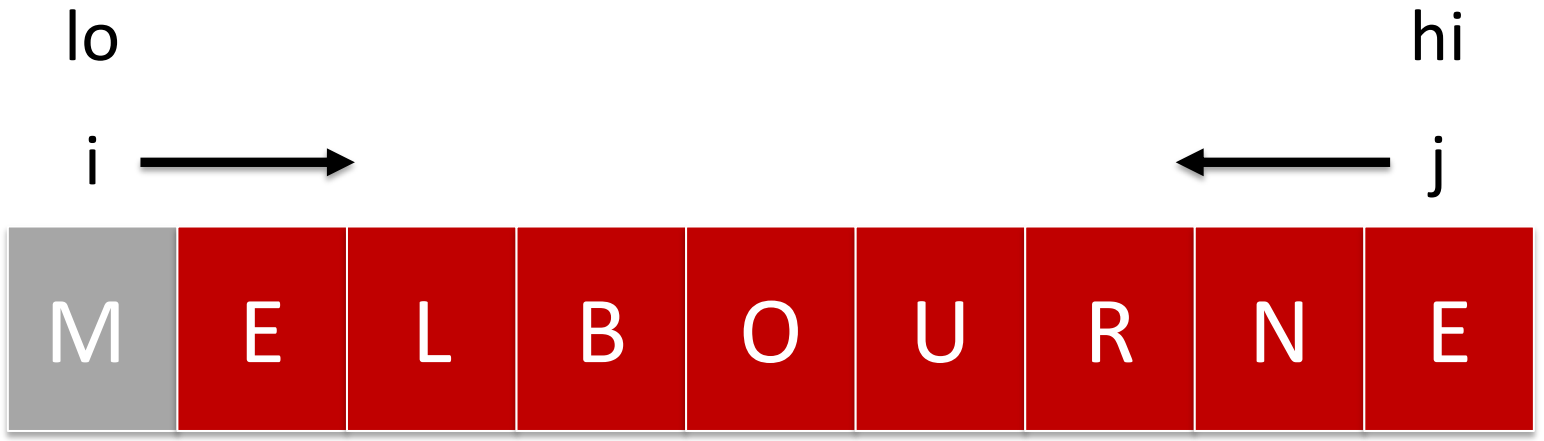
Append

```
38 # Quicksort - Tony Hoare, 1959
39 #
40 def partition(xs, pivot):
41   smaller = [ x for x in xs if x <  pivot]
42   equal   = [ x for x in xs if x == pivot]
43   greater = [ x for x in xs if x >  pivot]
44   return (smaller, equal, greater)
45
46 def qsort(xs):
47   n = len(xs)
48   if n < 2:
49     return xs
50   else:
51     pivot = xs[0]
52     (smaller, equal, greater) = partition(xs, pivot)
53     return qsort(smaller) + equal + qsort(greater)
54
```

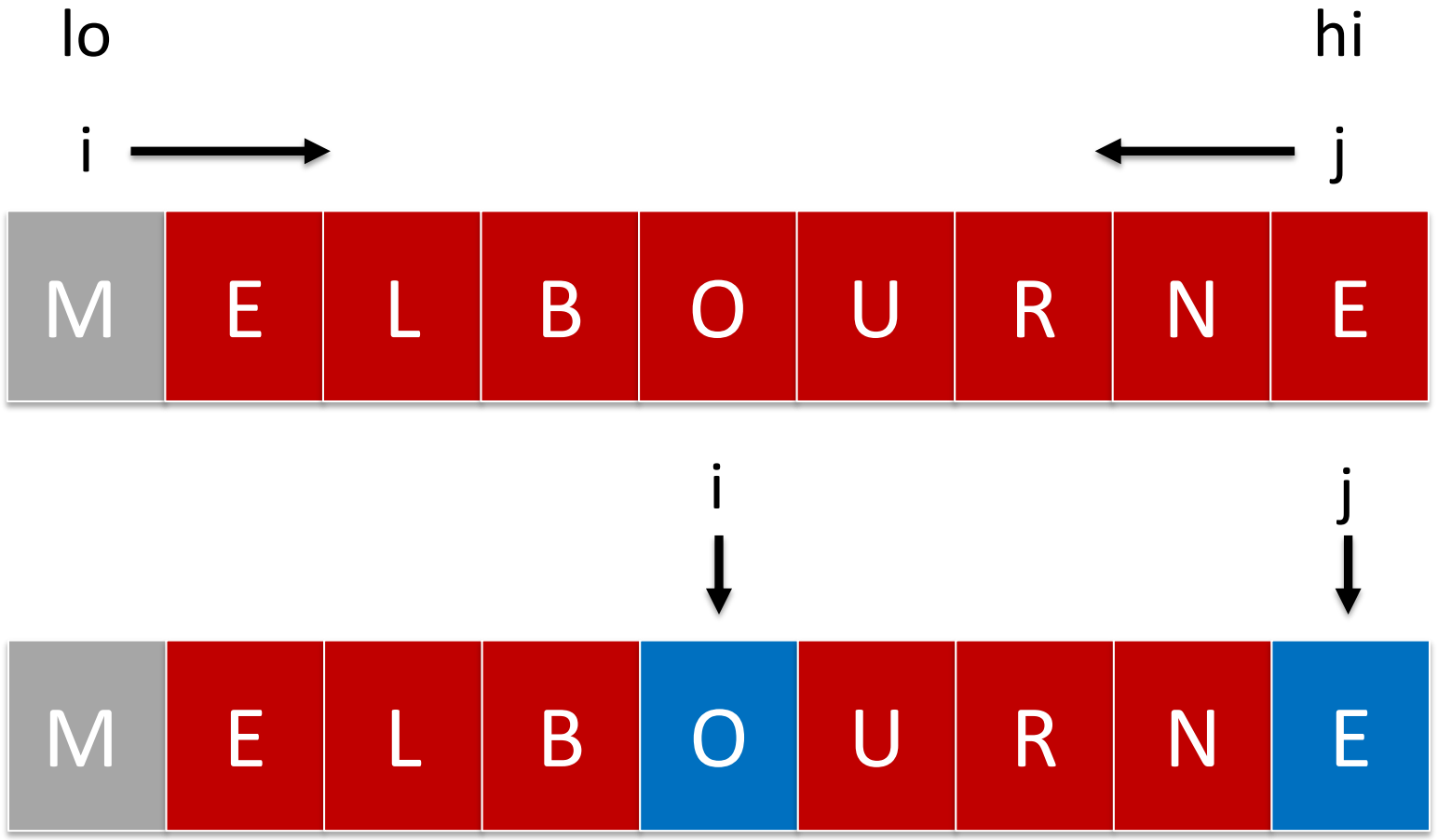
Memory Usage in Python



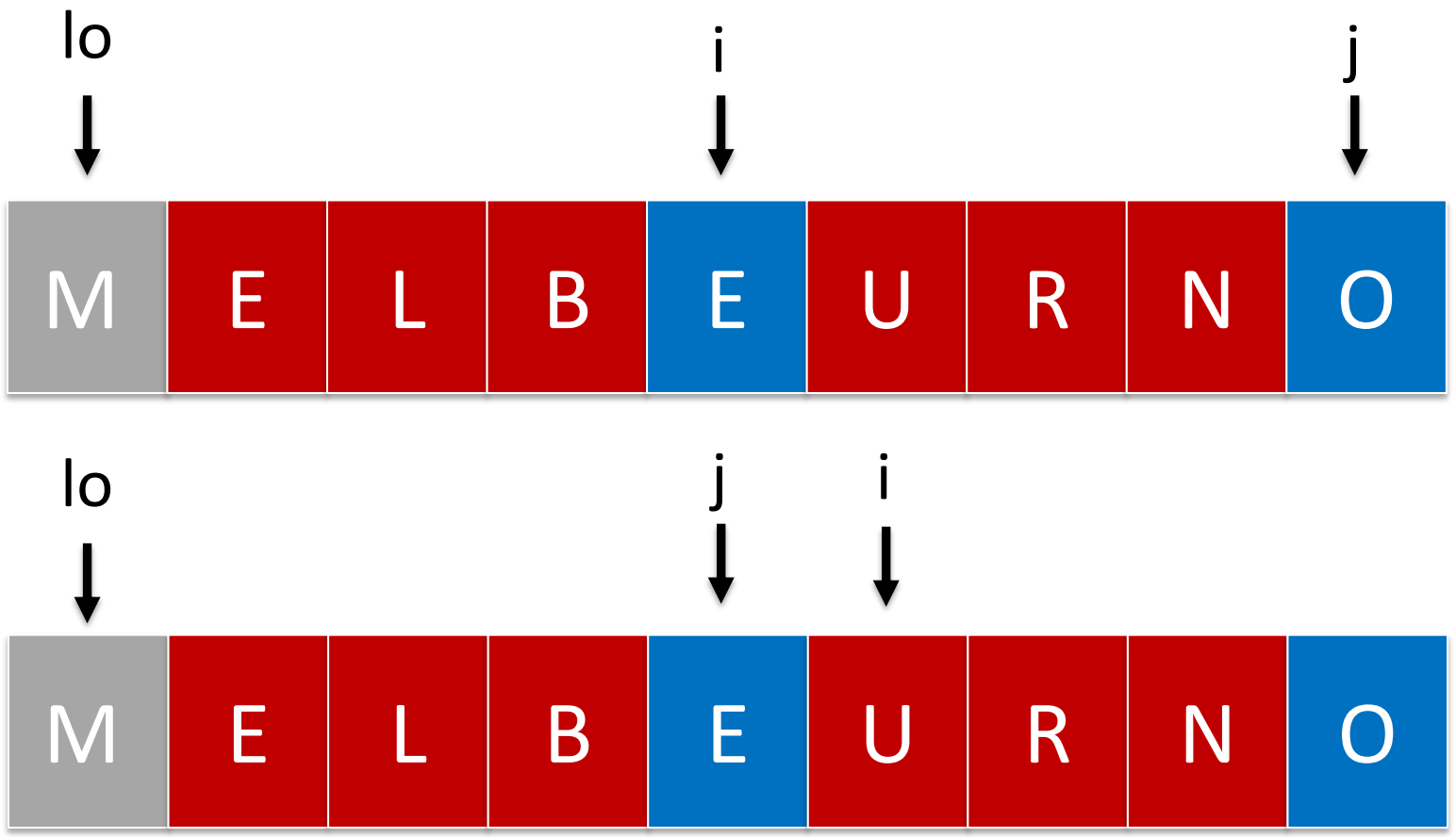
Memory Efficiency – Array in place



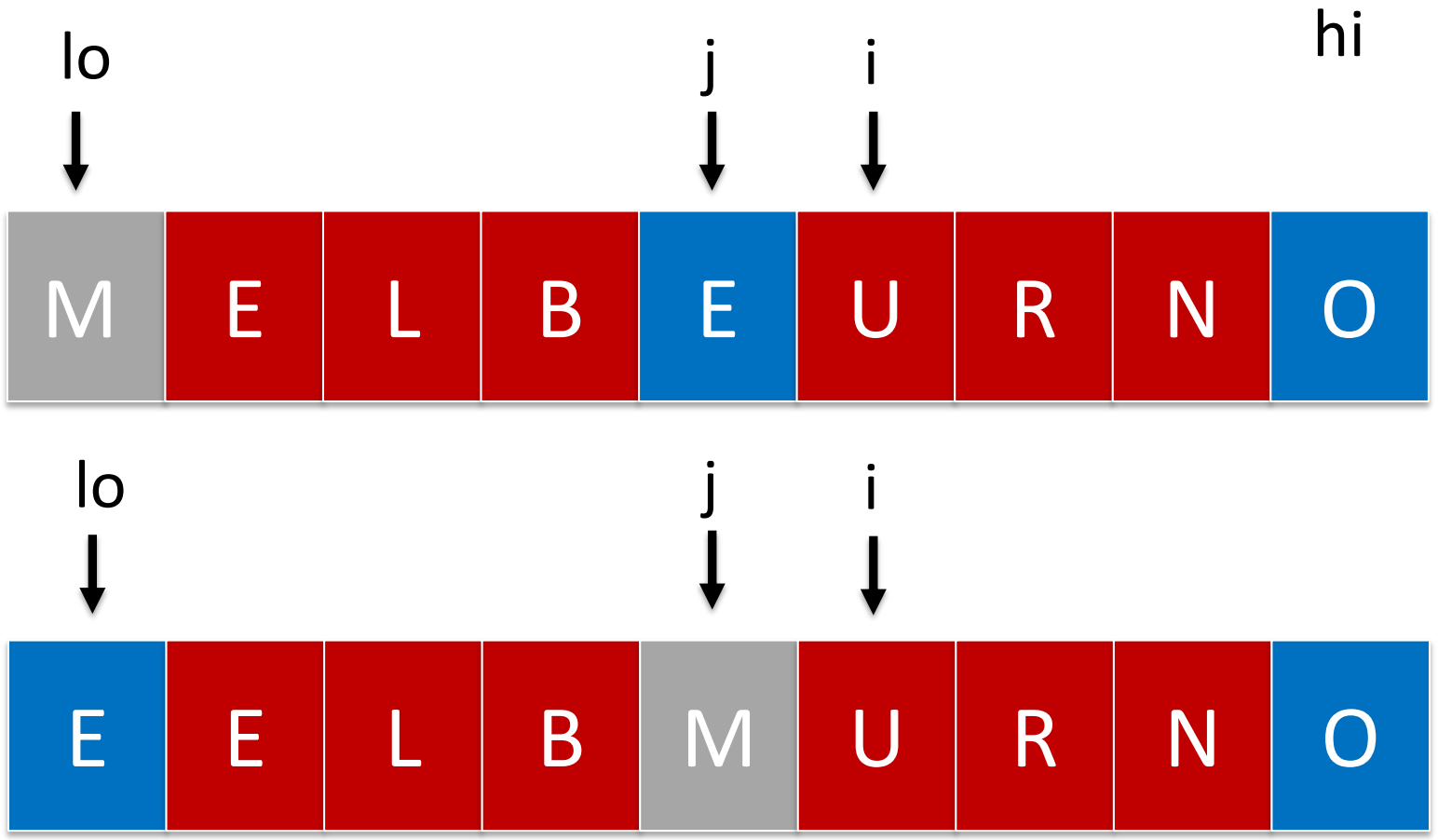
Memory Efficiency – Array in place



Memory Efficiency – Array in place



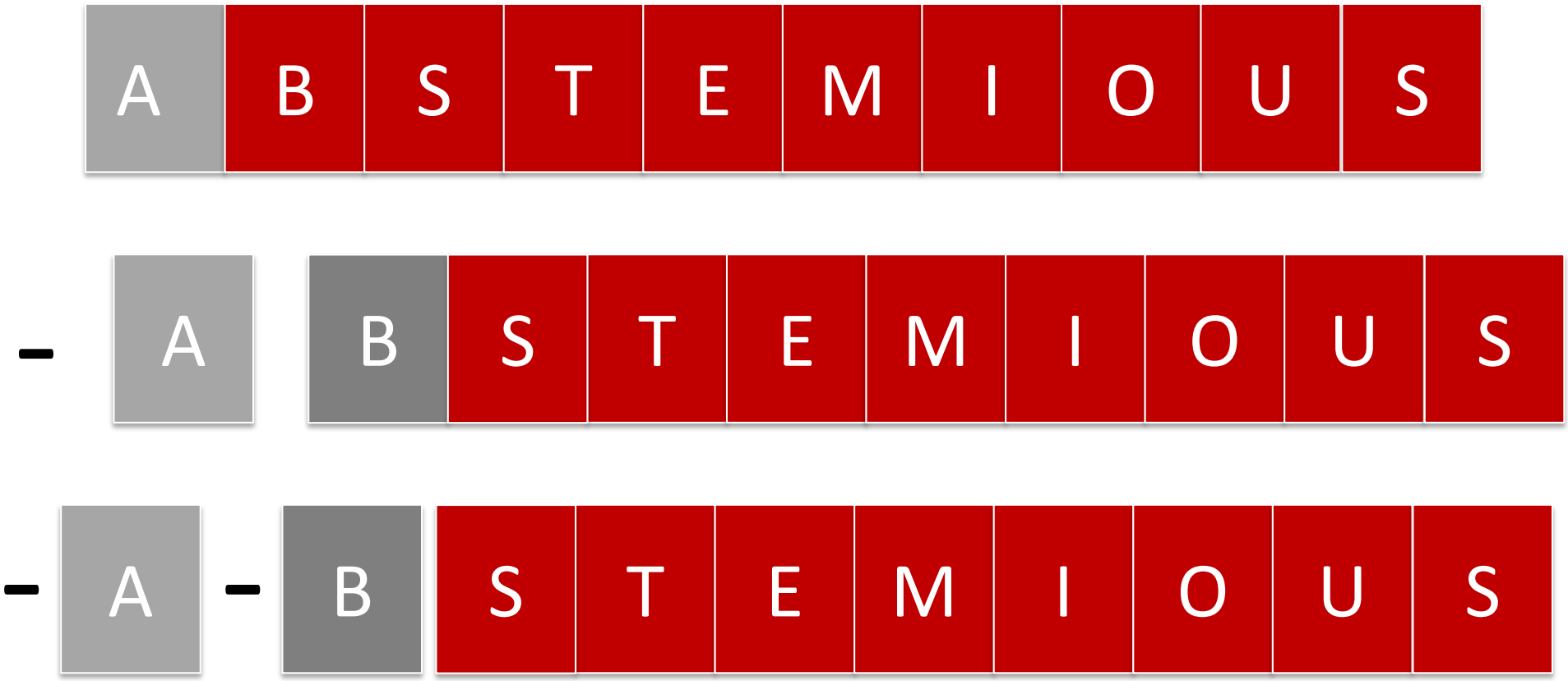
Memory Efficiency – Array in place



```java
// quicksort the subarray from a[lo] to a[hi]
private static void sort(Comparable[] a, int lo, int hi) {
  if (hi <= lo) return;
  int j = partition(a, lo, hi);
  sort(a, lo, j - 1);
  sort(a, j + 1, hi);
  assert isSorted(a, lo, hi);
}

public static void sort(Comparable[] a) {
  edu.princeton.cs.algs4.StdRandom.shuffle(a);
  sort(a, 0, a.length - 1);
  assert isSorted(a);
}
```

```
private static int partition (Comparable[]a,int lo, int hi){
  int i = lo;
  int j = hi + 1;
  Comparable v = a[lo];
  while (true) {

    // find item on lo to swap
    while (less(a[++i], v)) if (i == hi) break;

    // find item on hi to swap
    while (less(v, a[--j])) if (j == lo) break;      // redundant since

    // check if pointers cross
    if (i >= j) break;

    exch(a, i, j);
  }
  // put partitioning item v at a[j]
  exch(a, lo, j);

  // now, a[lo .. j-1] <= a[j] <= a[j+1 .. hi]
  return j;
}
```

Bad Pivots => Worst Case $O(N^2)$

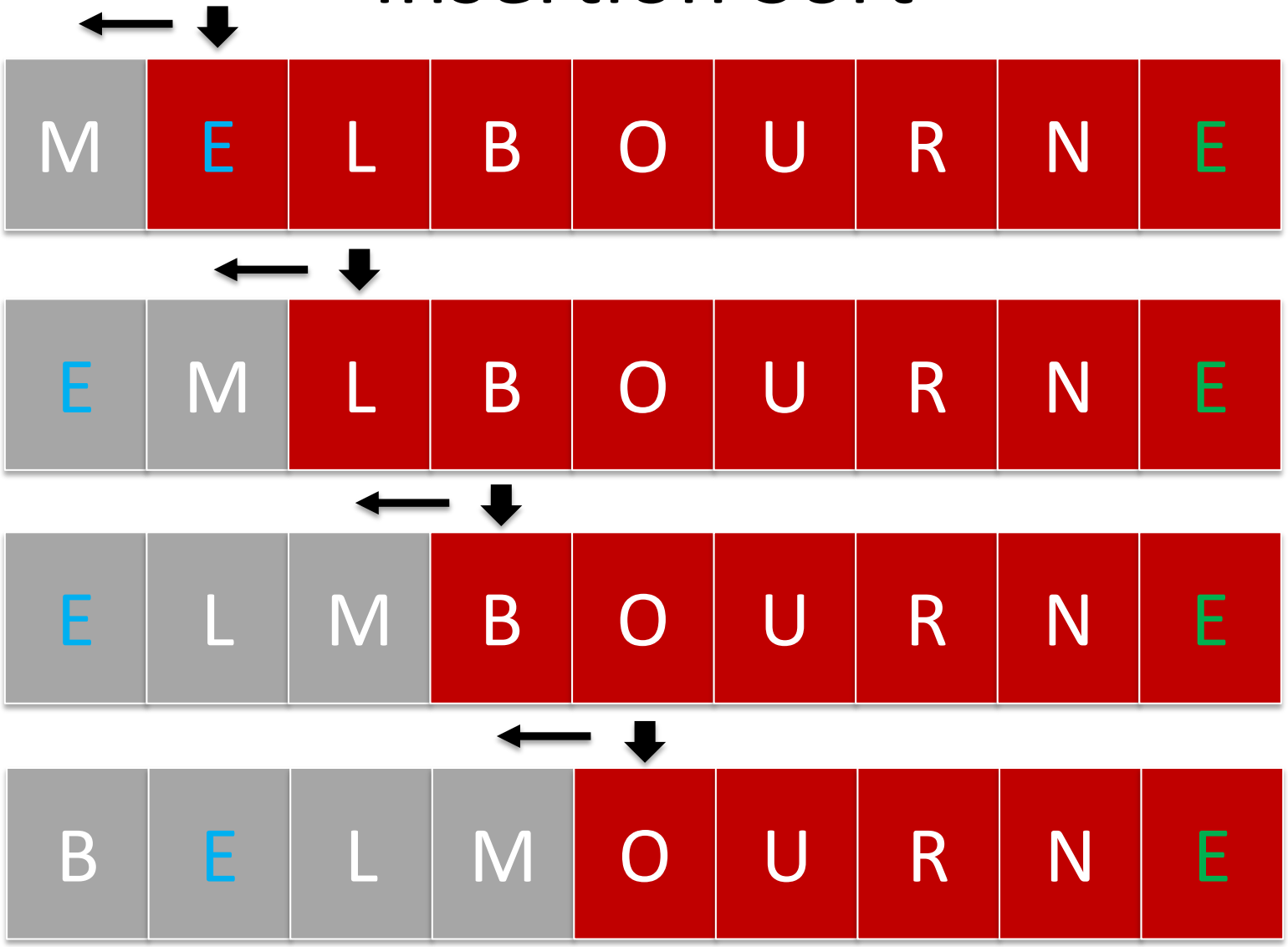


Quicksort

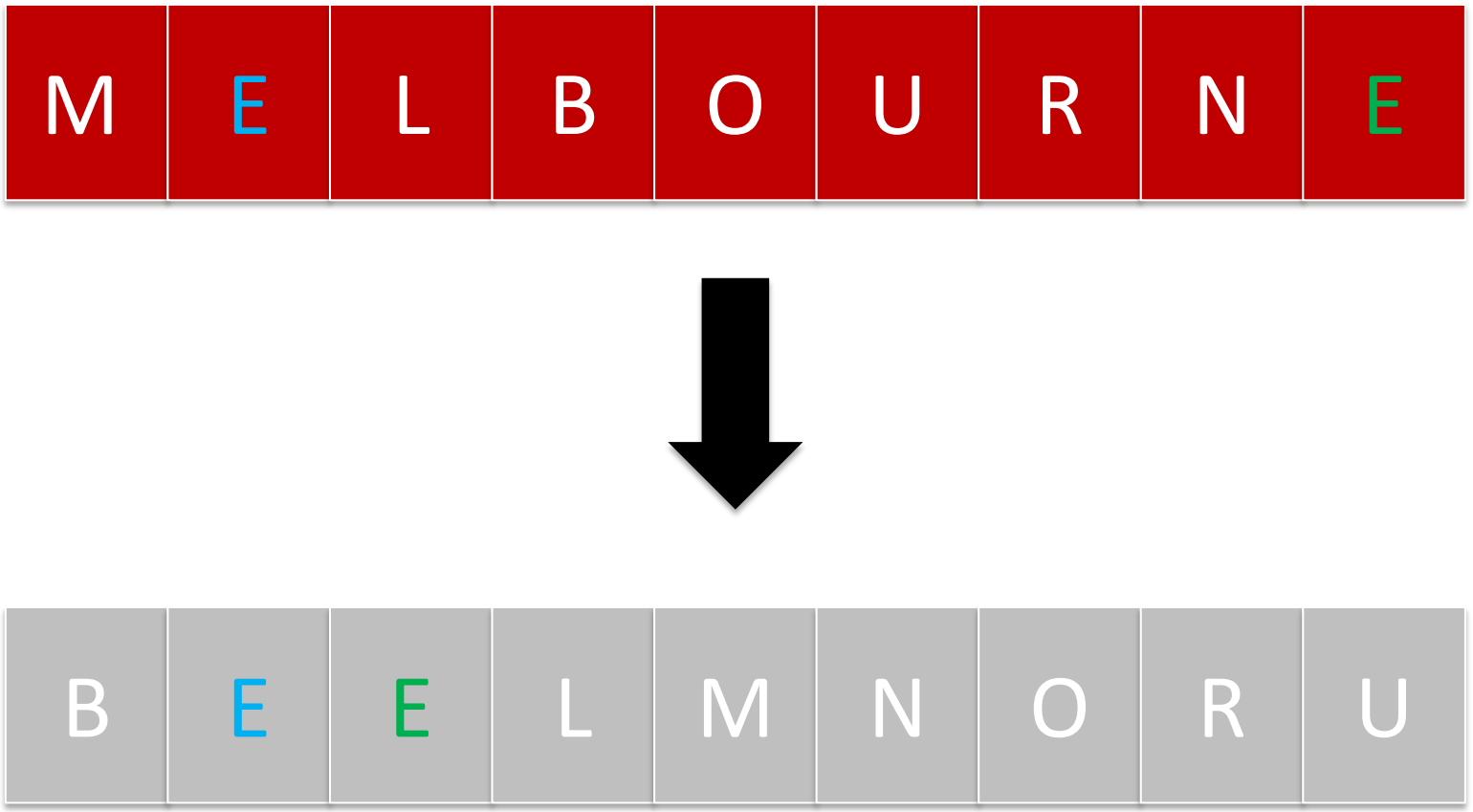
- O(N log N) on average, but $O(N^2)$ in the worst case.
- Quicksort is unstable.
- Quicksort is efficient with memory.

Insertion Sort

Insertion Sort



Insertion Sort is Stable



```
 */
public static void sort(Comparable[] a) {
    int n = a.length;
    for (int i = 1; i < n; i++) {
        for (int j = i; j > 0 && less(a[j], a[j-1]); j--) {
            exch(a, j, j-1);
        }
        assert isSorted(a, 0, i);
    }
    assert isSorted(a);
}
```

Raw use of paramete

Insertion Sort

- Is $O(N^2)$ in the worst case and the average case.
- Is O(N) when the keys are nearly sorted. (!)
- Sorts in place.
- Is stable.