CSCI 241

Lecture 4
Recursive Sorting:
Mergesort and Quicksort

Announcements

- First programming assignment (A1) is out!
  - Due 1/25.
  - It's largeish, so don't wait to get started.
  - In lab this week you will complete the implementation of *unit tests* to verify that your sorts work.
- Office hours: posted on the course webpage.
- Mentor hours: 4-7pm, CF162/164

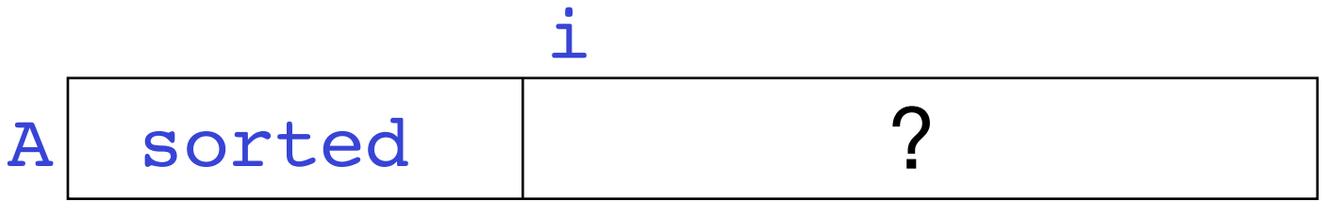
Goals:

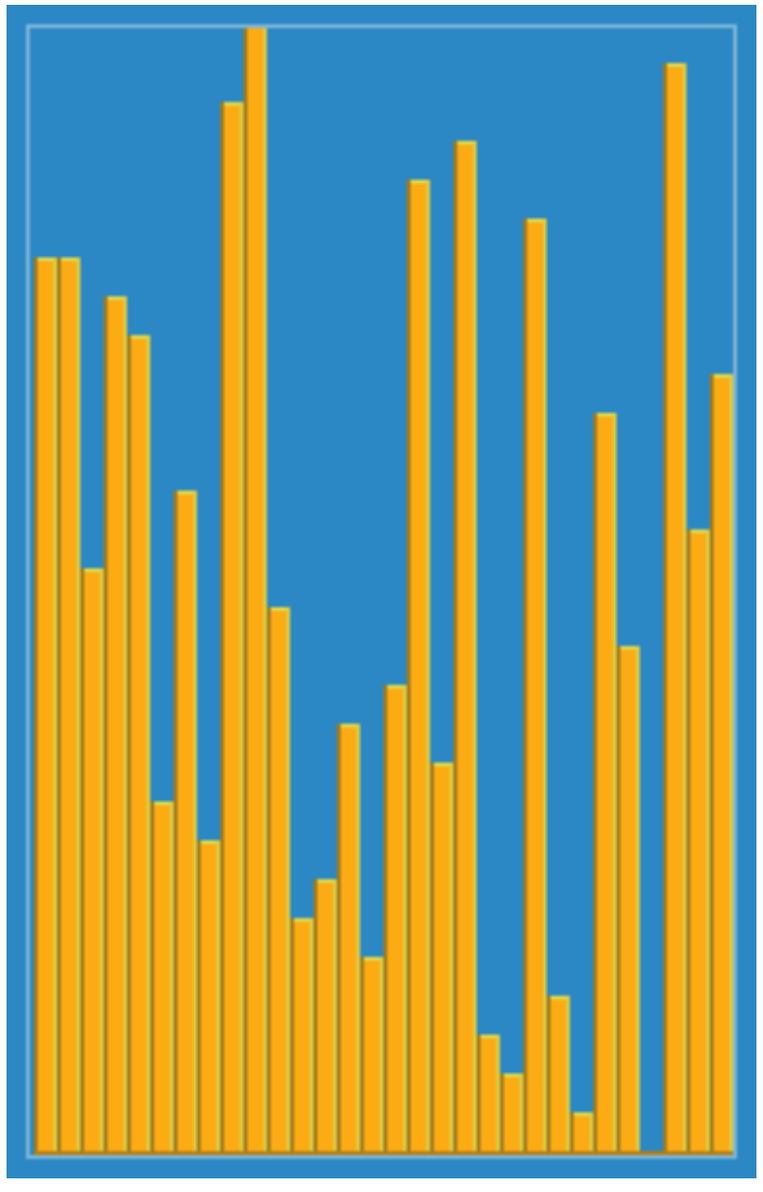
- Know the generic steps of a divide-and-conquer algorithm.
- Thoroughly understand the mechanism of mergesort and quicksort.
- Be prepared to implement **merge** and **partition** helper methods.

Incremental Algorithms

solve a problem a little bit at a time.

Natural programming mechanism: loops





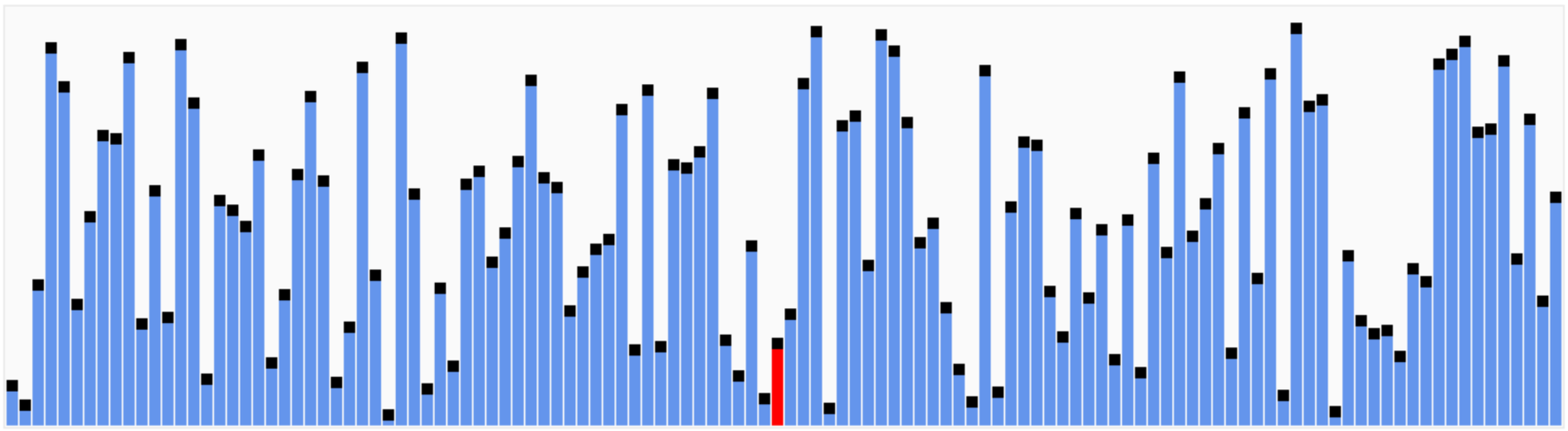
insertion sort

Divide-and-Conquer Algorithms

solve a problem by breaking it into smaller problems.

(easier!)

Natural programming
mechanism: recursion



https://upload.wikimedia.org/wikipedia/commons/f/fe/Quicksort.gif

```
/** sort A[start..end] using mergesort */
mergeSort(A, start, end):
   if (A.length < 2):
      return
   mid = (end-start)/2
```

```
mergeSort(A,start,mid)
```

Conquer (left)

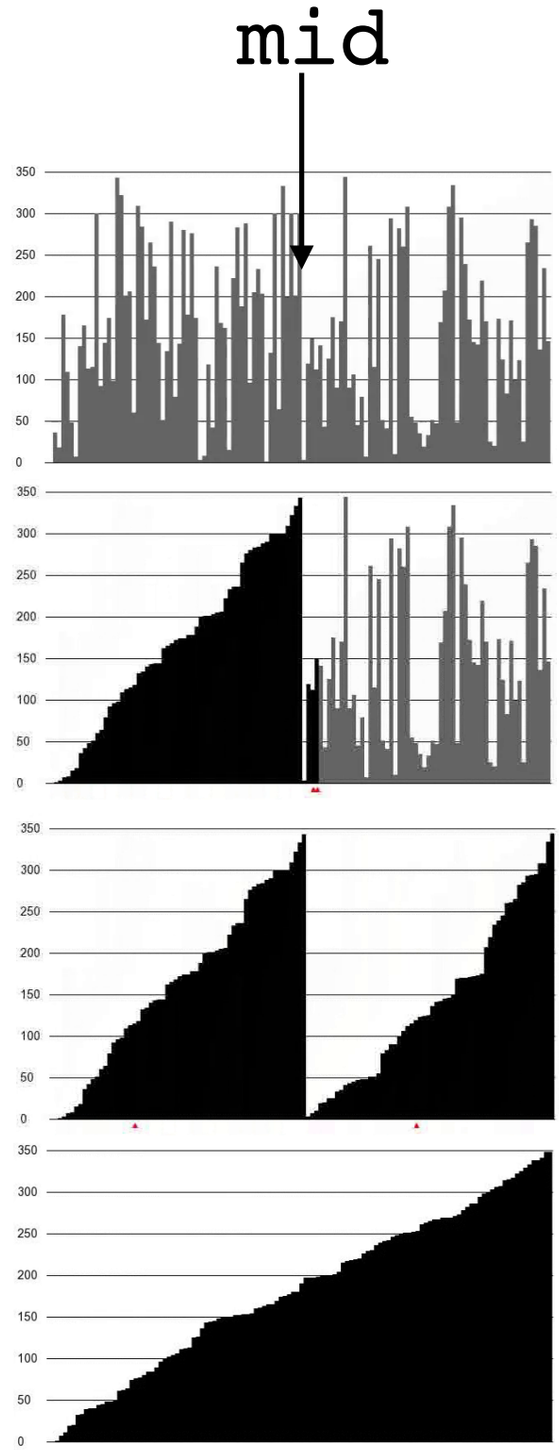
```
mergeSort(A,mid, end)
```

Conquer (right)

```
merge(A, start, mid, end)
```

Combine

Divide

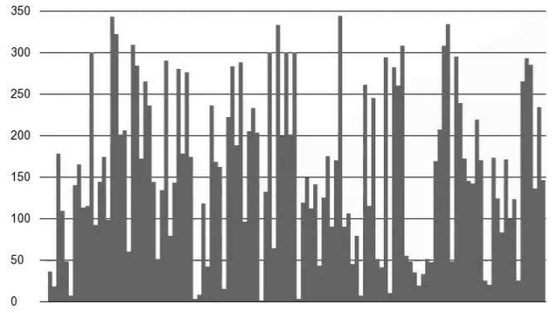


1. Spec

```
/** sort A[start..end] using mergesort */
mergeSort(A, start, end):
   if (A.length < 2):
      return
   mid = (end-start)/2
```

2. Base case

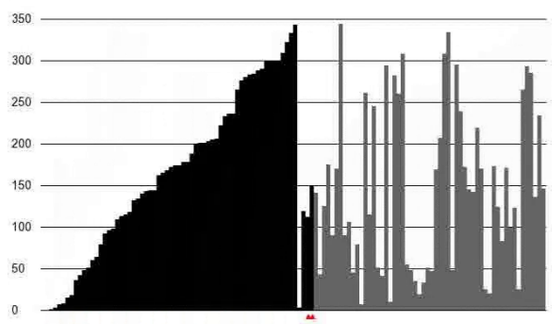
Divide



3. Progress

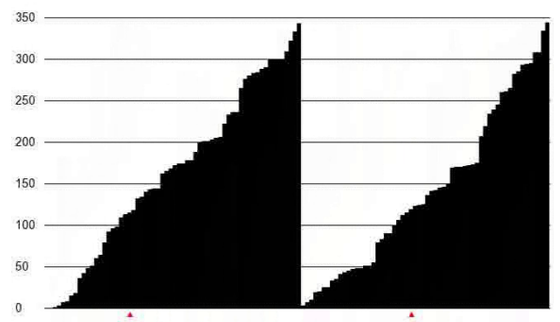
```
mergeSort(A,start,mid)
```

Conquer (left)




```
mergeSort(A,mid, end)
```

Conquer (right)




```
merge(A, start, mid, end)
```

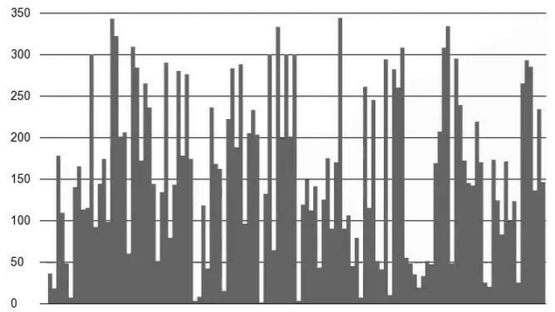
Combine

1. Spec

```
/** sort A[start..end] using mergesort */
mergeSort(A, start, end):
   if (A.length < 2):
      return
   mid = (end-start)/2
```

2. Base case

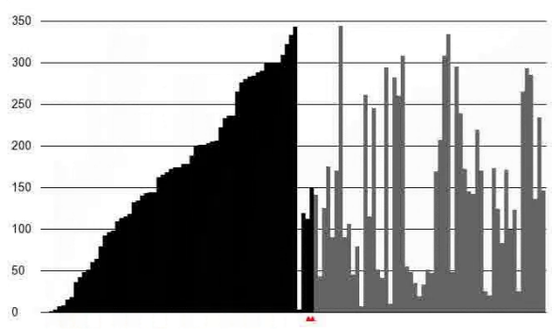
Divide



3. Progress

```
   sort A[start..mid]
```

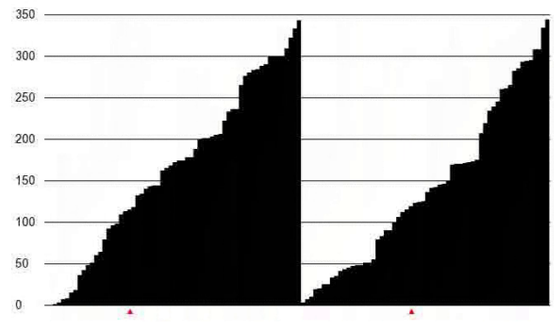
Conquer (left)



4. Replace recursive calls with spec

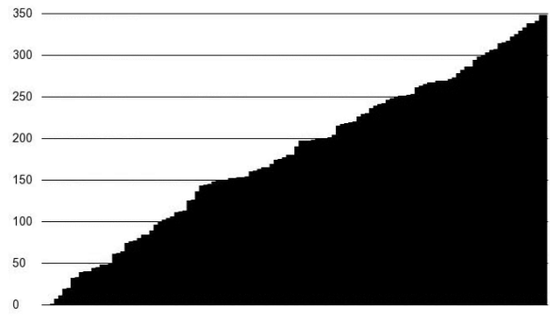
```
   sort A[mid..end]
```

Conquer (right)




```
merge(A, start, mid, end)
```

Combine

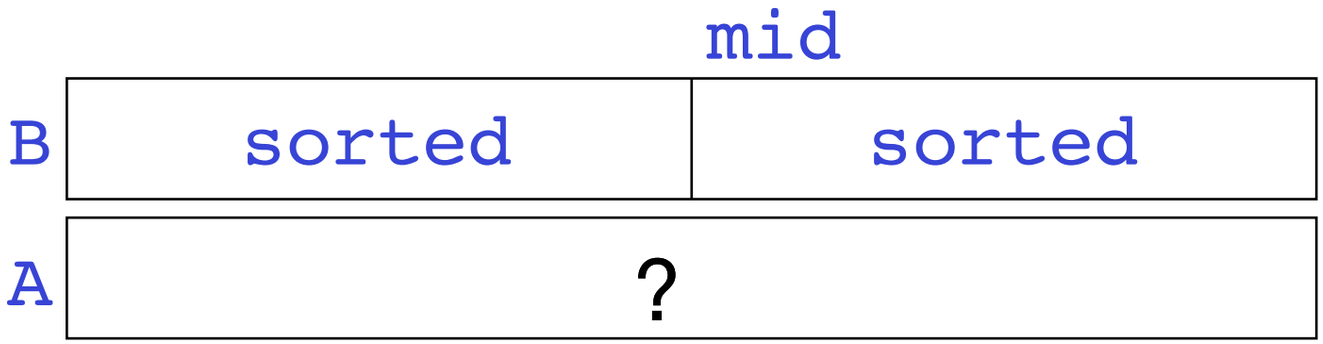


Merge Step

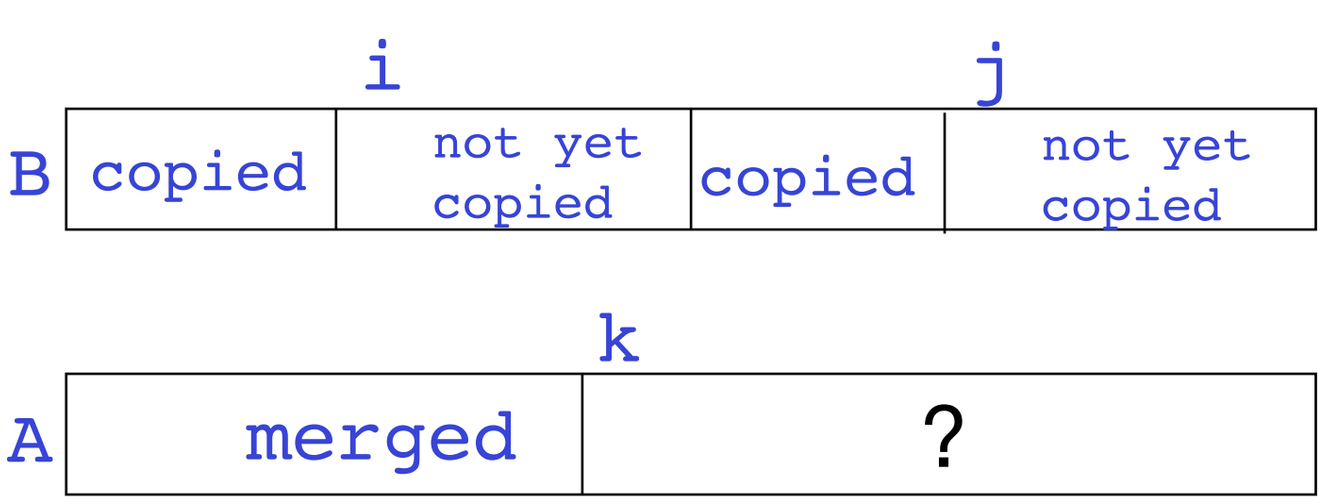
- Merge two halves, each of which is **sorted**.

Merge step: Loop Invariant

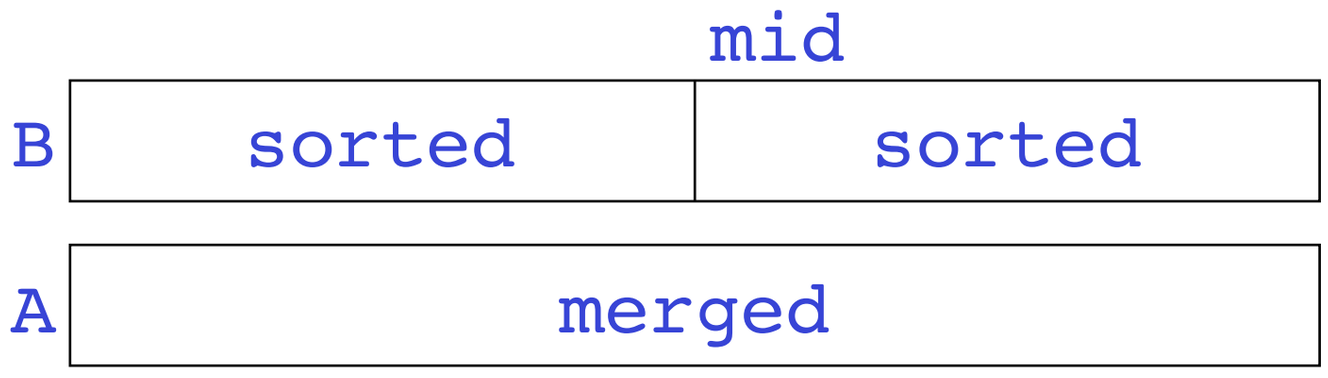
Precondition



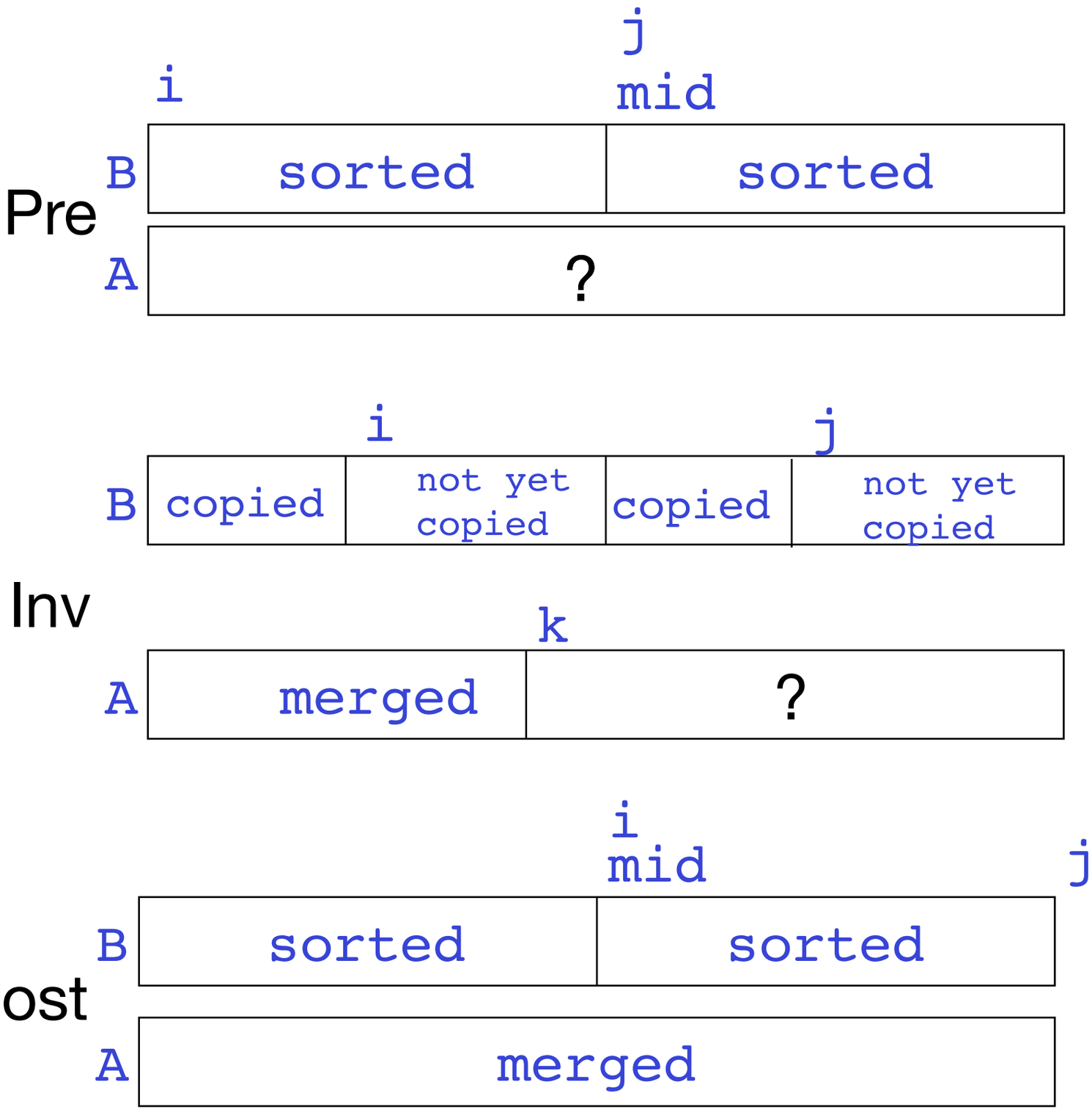
Invariant



Postcondition



Merge step




```
/** Merge sorted halves of A */
merge(A):
  B = a deep copy of A
  mid = A.length / 2
  i = 0
  j = mid
  k = 0
while i < mid and j < end:
  if B[i] < B[j]:
    A[k] = B[i]
    i++
  else:
    A[k] = B[j]
    j++
  k++

while i < mid:
  A[k] = B[i]
  i++, k++

while j < end:
  A[k] = B[j]
  j++, k++
```

Smaller thing goes first

Ran out of things in one list or the other

Copy remaining things from nonempty half

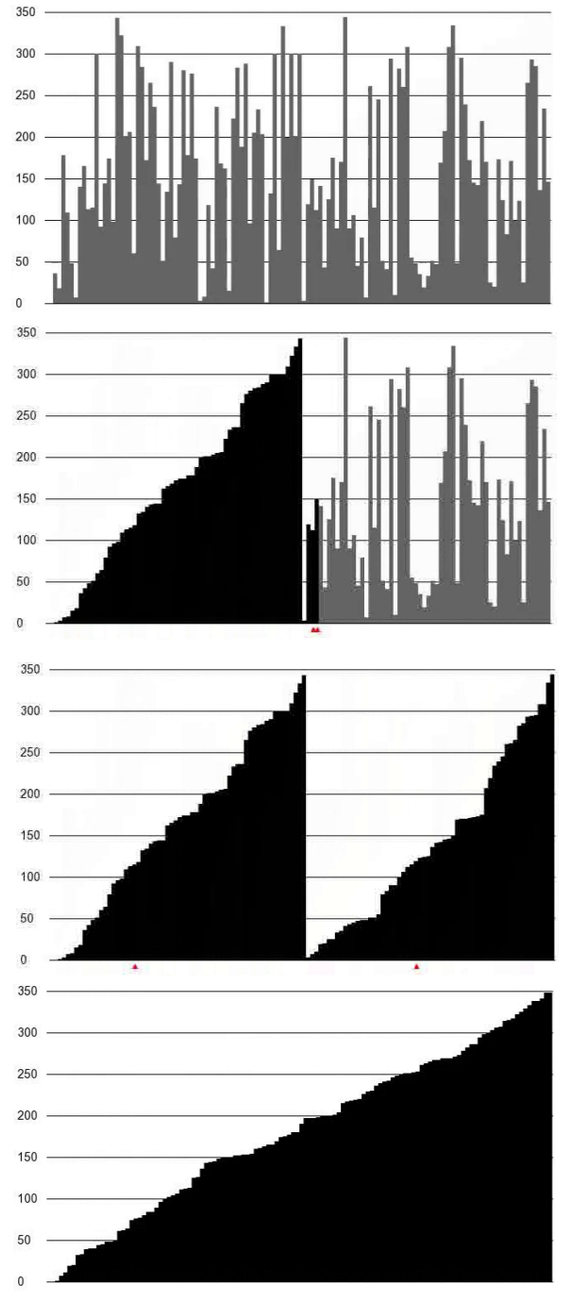
```
/** sort A[start..end] using mergesort */
mergeSort(A, start, end):
   if (A.length < 2):
      return
   mid = (end-start)/2

   mergeSort(A,start,mid)

   mergeSort(A,mid, end)

   merge(A, start, mid, end)
```

Divide

Conquer (left)

Conquer (right)

Combine



https://visualgo.net/bn/sorting

Quicksort

```
/** mergesort A[st..end]*/
mergeSort(A, st, end):
  if (small):
    return

mid = (end-start)/2

mergeSort(A,st,mid)
mergeSort(A,mid, end)

merge(A, st, mid, end)
```

Divide

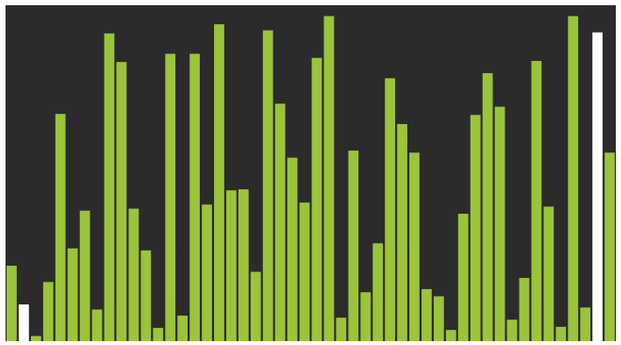
Conquer

Combine

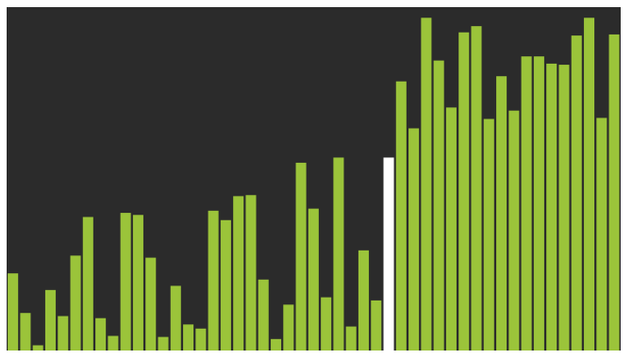
```
/** quicksort A[st..end]*/
quickSort(A, st, end):
  if (small):
    return

mid = partition(A,st,end)

quickSort(A,st,mid)
quickSort(A,mid, end)
```

Quicksort

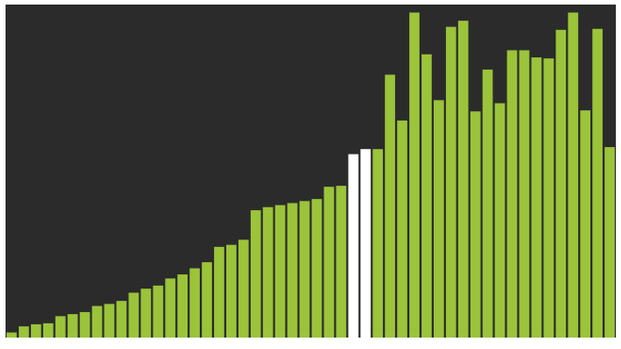
Unsorted:



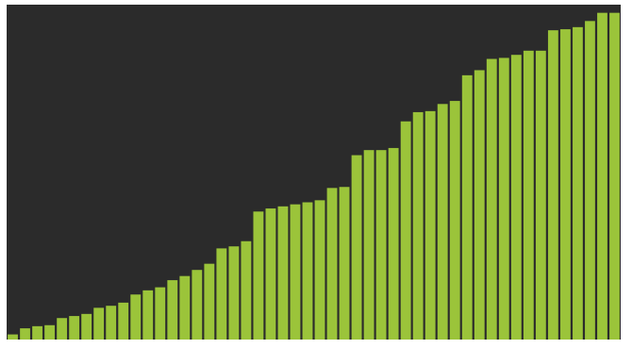
Small things left
big things right:



Sort left things:



Sort right things:



```
/** quicksort A[st..end]*/
quickSort(A, st, end):
  if (small):
    return
```

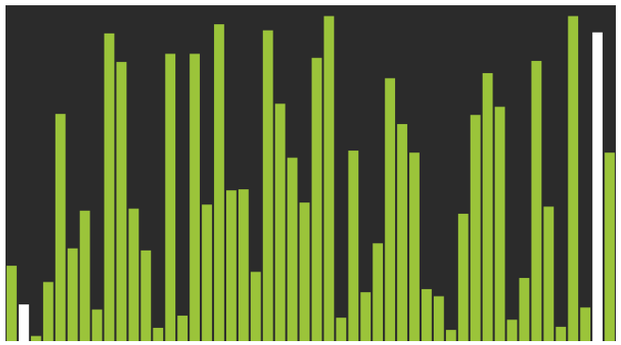
←— `mid = partition(A,st,end)`

←— `quickSort(A,st,mid)`

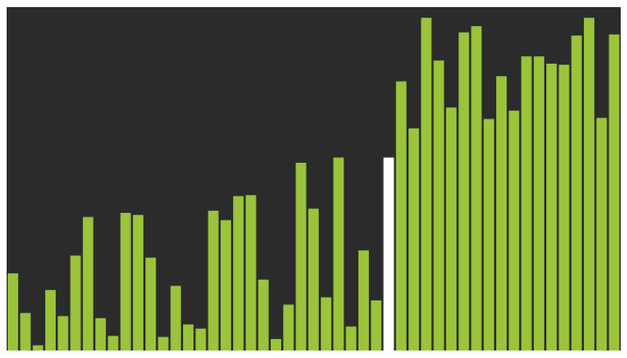
←— `quickSort(A,mid, end)`

Quicksort - Does it work?

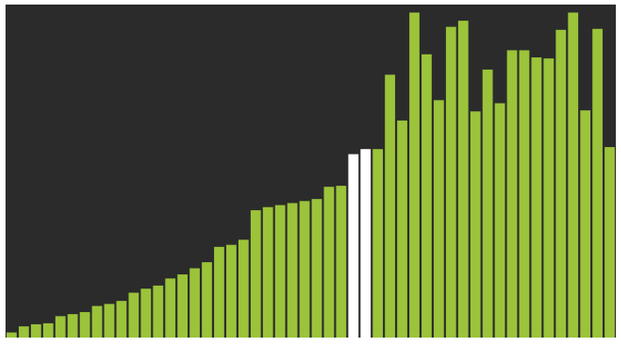
Unsorted:



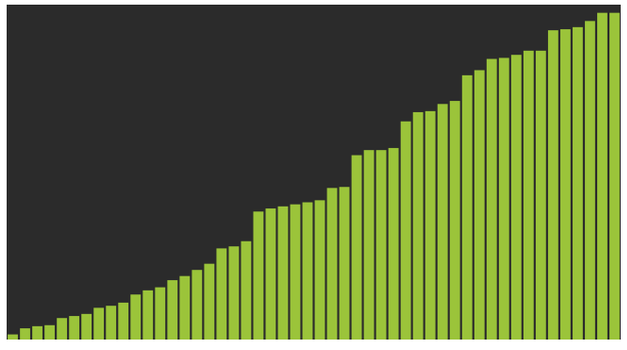
Small things left big things right:



Sort left things:



Sort right things:



```
/** quicksort A[st..end]*/
quickSort(A, st, end):
  if (small):
    return
```

←— `mid = partition(A,st,end)`

←— `quickSort(A,st,mid)`

←— `quickSort(A,mid, end)`

Quicksort

Key issues:

1. Picking the pivot
   - First, middle, or last
   - Median of first, middle, and last
2. Implementing `partition`

```
/** quicksort A[st..end]*/
quickSort(A, st, end):
  if (small):
    return

mid = partition(A,st,end)

quickSort(A,st,mid)

quickSort(A,mid, end)
```

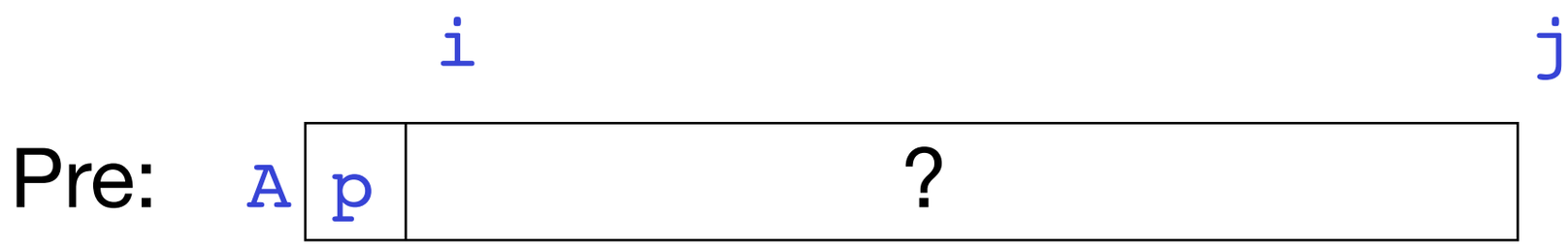
Quicksort

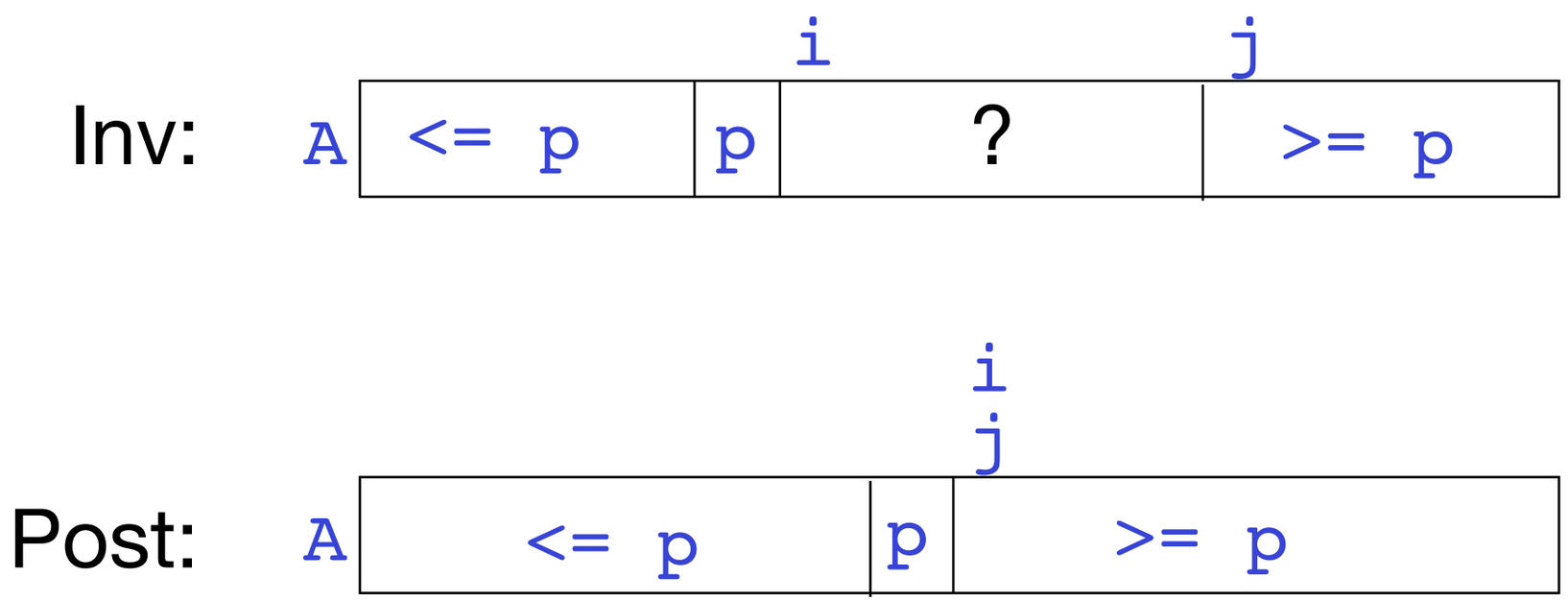
Key issues:

1. Picking the pivot
   - First, middle, or last
   - Median of first, middle, and last
2. Implementing `partition`

```
/** quicksort A[st..end]*/
quickSort(A, st, end):
  if (small):
    return

mid = partition(A,st,end)

quickSort(A,st,mid)

quickSort(A,mid, end)
```

```
/** partition A around the pivot A[pivIndex].
  * return the pivot's new index.
  *  precondition: start <= pivIndex < end
  *  postcondition: A[start..i] <= A[i] <= A[i+1..end]
  *     where i is the return value */
   public int partition(int[] A, int pivIndex) {
```





Caution

- There are multiple ways to implement partition. I just showed you my favorite.

- If you look at other resources, you may find different approaches that accomplish the same result.