# CS062

# DATA STRUCTURES AND ADVANCED PROGRAMMING

## 15: Quicksort

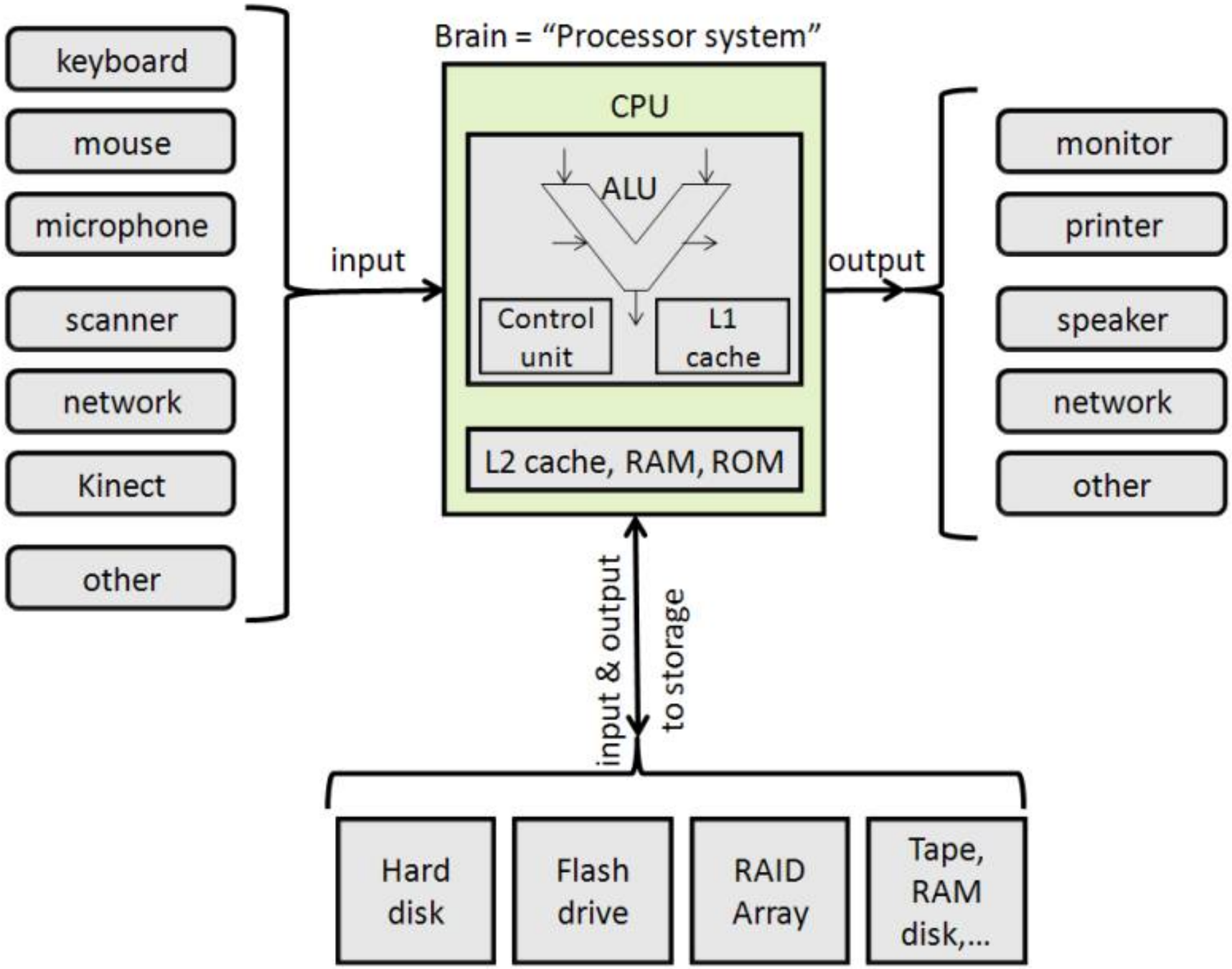
Alexandra Papoutsaki
she/her/hers

# Lecture 13: Quicksort

- This week's assignment
- Quicksort

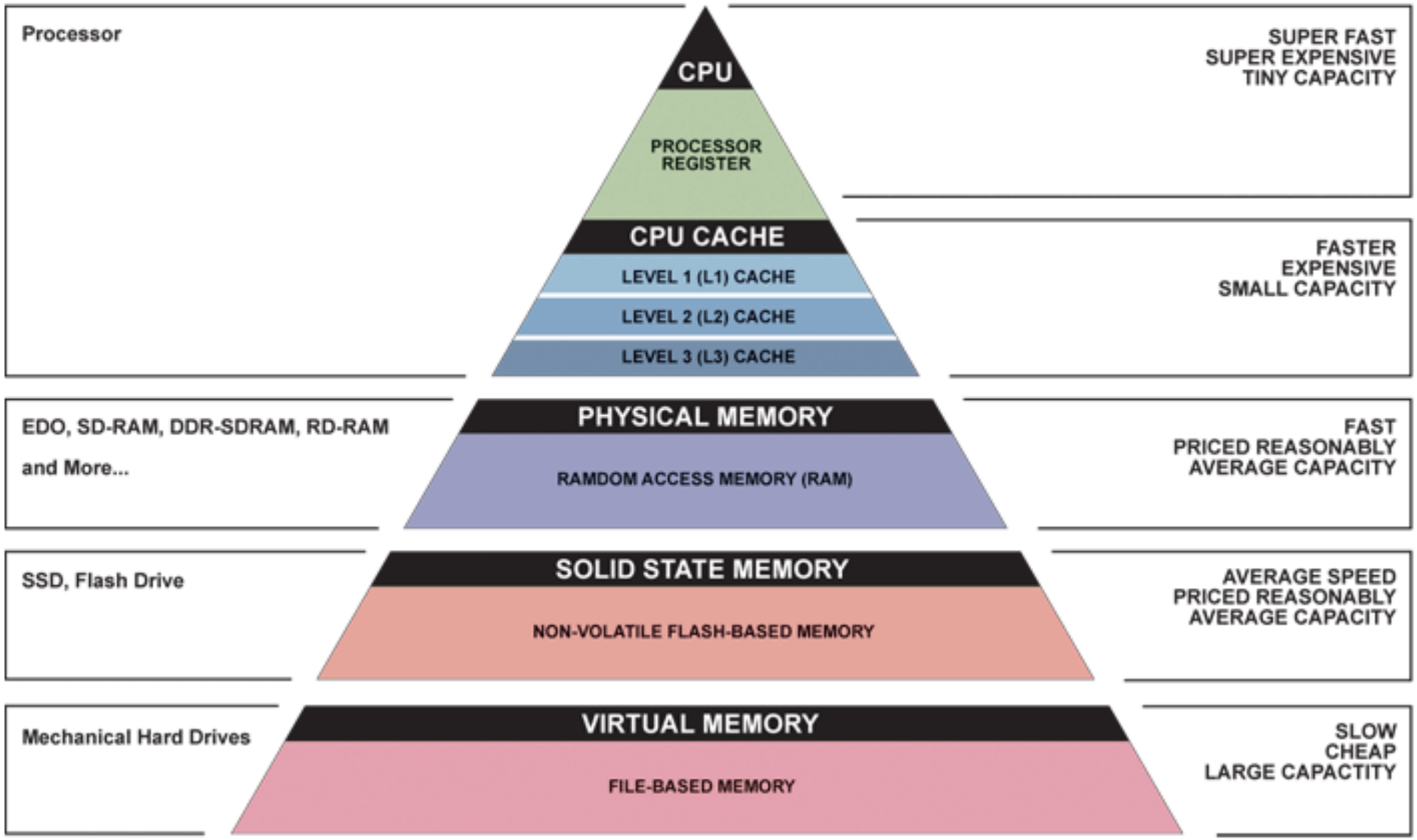
Some slides adopted from Algorithms 4th Edition or COS226

# Basic Von Neumann computer architecture




http://cs.sru.edu/~mullins/cpsc100book/module03_internalHardware/module03-05_internalHardware.html

# Memory Hierarchy (a simplified summarized story)




▲ Simplified Computer Memory Hierarchy
Illustration: Ryan J. Leng

# Registers

- Fastest, most expensive, tiny capacity.
- Run at same speed with CPU's clock ~3GHz today (3 billion operations/sec).
- Typically, 16-32 of them per core.
- Can hold 32 or 64 bits based on architecture that correspond to data or memory locations.

# Cache

- Faster, expensive, small capacity.
- A bit slower than CPU's speed but faster than main memory.
- Typically, 2-3 levels (L1, L2, L3, etc.).
- 32-64 KB for L1, 128-512 KB for L2.

## Main (physical) memory (RAM)

- Fast, reasonably priced, average capacity.
- Much slower than CPU but significantly faster than disk.
- 8-32 GB.
- All programs and data must fit in memory.
  - If not, virtual memory to the rescue while accessing the disk.

External (secondary or auxiliary) memory (disk)

- Slower speed, reasonably priced, large capacity.
- Most computers have now solid state drives (SSDs) which are faster (but typically more expensive) than mechanical hard disk drives (HDDs).
- Hundreds of GB to a few TB.

# Assignment 5: On-disk merge sort

- All sorting algorithms we have seen assume that the data to be sorted can fit in main memory (RAM). In the era of big data, this is not always the case.
- For assignment 5, you will work on an external (on-disk) mergesort.
- Data are read from the disk in chunks of maximum size and are individually sorted using regular merge sort and stored back on disk in temporary "chunk" files.
- Temporary files are merged into an increasingly larger temporary file till the entire original data have been sorted and saved back on disk.
  - This is accomplished by iteratively merging the temporary file with a sorted temporary "chunk" file. Two buffered readers allow you to read a line/file at a time, compare them, and choose the "smallest" to be saved into the temporary file that contains all of merged data so far. Essentially, merge method of mergesort!

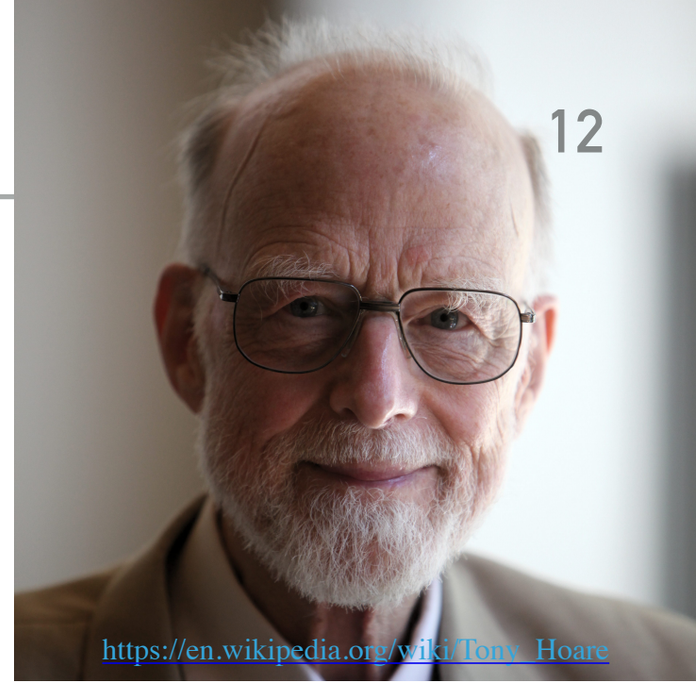
# BufferedReader

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class ReadFile {


 public static void main(String[] args) {
  String fileName = "somePath/File.txt";

  try (BufferedReader br = new BufferedReader(new FileReader(fileName))) {

   String strCurrentLine = br.readLine();

   while (strCurrentLine != null) {
        //do something
        strCurrentLine = br.readLine();
   }

  } catch (IOException e) {
   e.printStackTrace();
  }
 }
}
```

# Lecture 13: Quicksort

- This week's assignment
- Quicksort

# Basics


https://en.wikipedia.org/wiki/Tony_Hoare

- Divide-and-conquer method
- Invented by Sir Tony Hoare in 1959.
  - Wanted to sort Russian words before looking them up in dictionary.
  - Came up with quicksort but did not know how to implement it.
  - Learned Algol 60 and recursion and implemented it.
  - Won the 1980 Turing Award (also invented the concept of null —and regretted it).

# Algorithm sketch

- Partition so that, for some pivot `pivot`:
  - Entry `a[pivot]` is in place.
  - There is no larger entry to the left of `pivot`.
  - No smaller entry to the right of `pivot`.
- Sort each subarray recursively.

# Quicksort Code

```
// quicksort the subarray from a[lo] to a[hi]
private static <E extends Comparable<E>> void quickSort(E[] a, int lo, int hi) {
    if (lo < hi){
       int pivot = partition(a, lo, hi);
       quickSort(a, lo, pivot-1);
       quickSort(a, pivot+1, hi);
    }
}
```

```
/**
* Rearranges the array in ascending order, using the natural order.
* @param a the array to be sorted
*/
public static <E extends Comparable<E>> void quickSort(E[] a) {
   quickSort(a, 0, a.length-1);
}
```

# Lomuto's* partition scheme

- Partition the subarray `a[lo...hi]` so that `a[lo...pivot-1]<=a[pivot]<=a[pivot+1...hi]`
- Start with `pivot` at `hi`, partitioning index `i` at `lo-1`, and pointer `j` at `lo`.
- Repeat the following until pointers `j` moves from `lo` to `hi-1`:
  - If element at `j` is smaller or equal to `pivot` increment `i` by 1 and exchange `a[i]` with `a[j]`.
- Increase `i` by 1 and exchange `a[i]` and `a[hi]`. Return `i` as the new pivot.
- *The recommended textbook follows Hoare's partition scheme which is faster but harder to implement

# Partition Code

```
private static <E extends Comparable<E>> int partition(E[] a, int lo, int hi) {
        int i = lo - 1;
        E pivot = a[hi]

        for (int j = lo; j < hi; j++) {
            if (a[j].compareTo(pivot) <= 0) {
                i++;
                E temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
        }

        i++;
        E temp = a[i];
        a[i] = a[hi];
        a[hi] = temp;

        return i;
    }
```

# Quicksort Example - Sort [S,O,R,T,E,X,A,M,P,L,E]

```
// quicksort the subarray from a[lo] to a[hi]
private static <E extends Comparable<E>> void quickSort(E[] a, int lo, int hi) {
    if (lo < hi){
       int pivot = partition(a, lo, hi);
       quickSort(a, lo, pivot-1);
       quickSort(a, pivot+1, hi);
    }
}
```

Call partition with lo=0, hi=10

```
/**
* Rearranges the array in ascending order, using the natural order.
* @param a the array to be sorted
*/
public static <E extends Comparable<E>> void quickSort(E[] a) {
   quickSort(a, 0, a.length-1);
}
```

| | S | O | R | T | E | X | A | M | P | L | E |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | lo | | | | | | | | | | hi |

# Partition Example, lo=0, hi=10

| | S | O | R | T | E | X | A | M | P | L | E |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

i lo hi

```
    // partition the subarray a[lo..hi] so that a[lo..pivot-1] <= a[pivot] <= a[pivot+1..hi] and return
the partitioning index i.
    private static <E extends Comparable<E>> int partition(E[] a, int lo, int hi) {
        int i = lo - 1;
        E pivot = a[hi];
```

# Partition Example, lo=0, hi=10

| | S | O | R | T | E | X | A | M | P | L | E |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

i lo, j hi

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=0, hi=10

| | S | O | R | T | E | X | A | M | P | L | E |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

i lo j hi

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=0, hi=10

| | S | O | R | T | E | X | A | M | P | L | E |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

i lo j hi

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=0, hi=10

| | S | O | R | T | E | X | A | M | P | L | E |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

i lo j hi

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=0, hi=10

| | S | O | R | T | E | X | A | M | P | L | E |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

i lo j hi

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=0, hi=10

| | S | O | R | T | E | X | A | M | P | L | E |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

lo, i j hi

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=0, hi=10

| | E | O | R | T | S | X | A | M | P | L | E |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

lo, i j hi

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=0, hi=10

| | E | O | R | T | S | X | A | M | P | L | E |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

lo, i j hi

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=0, hi=10

| | E | O | R | T | S | X | A | M | P | L | E |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

lo, i j hi

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=0, hi=10

| | E | O | R | T | S | X | A | M | P | L | E |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

lo i j hi

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=0, hi=10

| | E | A | R | T | S | X | O | M | P | L | E |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

lo i j hi

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=0, hi=10

| | E | A | R | T | S | X | O | M | P | L | E |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

lo i j hi

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=0, hi=10

| | E | A | R | T | S | X | O | M | P | L | E |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

lo i j hi

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=0, hi=10

| | E | A | R | T | S | X | O | M | P | L | E |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

lo i j hi

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=0, hi=10

| | E | A | R | T | S | X | O | M | P | L | E |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

lo i j, hi

```
i++;
E temp = a[i];
a[i] = a[hi];
a[hi] = temp;

return i;
```

# Partition Example, lo=0, hi=10

| | E | A | E | T | S | X | O | M | P | L | R |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

lo i j, hi

```
i++;
E temp = a[i];
a[i] = a[hi];
a[hi] = temp;

return i;
```

# Partition Example, lo=0, hi=10

| | E | A | E | T | S | X | O | M | P | L | R |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

lo i j, hi

```
i++;
E temp = a[i];
a[i] = a[hi];
a[hi] = temp;

return i;
```

Partition for lo=0, hi=10 is complete. i=2

Call quickSort recursively on left subarray with lo=0, hi =1

# Quicksort Example - Sort [E,A]

```
// quicksort the subarray from a[lo] to a[hi]
private static <E extends Comparable<E>> void quickSort(E[] a, int lo, int hi) {
    if (lo < hi){
       int pivot = partition(a, lo, hi);
       quickSort(a, lo, pivot-1);
       quickSort(a, pivot+1, hi);
    }
}
```

Call partition with lo=0, hi=1

| | E | A | E | T | S | X | O | M | P | L | R |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | lo | hi | | | | | | | | | |

# Partition Example, lo=0, hi=1

| | E | A | E | T | S | X | O | M | P | L | R |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| i | lo | hi | | | | | | | | | |

```
    // partition the subarray a[lo..hi] so that a[lo..pivot-1] <= a[pivot] <= a[pvoti+1..hi] and return
the partitioning index i.
    private static <E extends Comparable<E>> int partition(E[] a, int lo, int hi) {
        int i = lo - 1;
        E pivot = a[hi];
```

# Partition Example, lo=0, hi=1

| | E | A | E | T | S | X | O | M | P | L | R |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| i | lo, j | hi | | | | | | | | | |

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=0, hi=1

| | E | A | E | T | S | X | O | M | P | L | R |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | lo, i | j, hi | | | | | | | | | |

```
i++;
E temp = a[i];
a[i] = a[hi];
a[hi] = temp;

return i;
```

# Partition Example, lo=0, hi=1

| | A | E | E | T | S | X | O | M | P | L | R |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | lo, i | j, hi | | | | | | | | | |

```
i++;
E temp = a[i];
a[i] = a[hi];
a[hi] = temp;
return i;
```

# Partition Example, lo=0, hi=1

| | A | E | E | T | S | X | O | M | P | L | R |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | lo, i | j, hi | | | | | | | | | |

```
i++;
E temp = a[i];
a[i] = a[hi];
a[hi] = temp;

return i;
```

Partition for lo=0, hi=1 is complete. i=0

No partition for subarrays of size 1.

Call quickSort recursively on right subarray with lo=3, hi = 10.

# Quicksort Example - Sort [T, S, X, O, M, P, L, R]

```
// quicksort the subarray from a[lo] to a[hi]
private static <E extends Comparable<E>> void quickSort(E[] a, int lo, int hi) {
    if (lo < hi){
       int pivot = partition(a, lo, hi);
       quickSort(a, lo, pivot-1);
       quickSort(a, pivot+1, hi);
    }
}
```

Call partition with lo=3, hi=10

| | A | E | E | T | S | X | O | M | P | L | R |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | lo | | | | | | | hi |

# Partition Example, lo=3, hi=10

| | A | E | E | T | S | X | O | M | P | L | R |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | i | lo | | | | | | | hi |

```
    // partition the subarray a[lo..hi] so that a[lo..pivot-1] <= a[pivot] <= a[pivot+1..hi] and return
the partitioning index pi.
    private static <E extends Comparable<E>> int partition(E[] a, int lo, int hi) {
        int i = lo - 1;
        E pivot = a[hi];
```

# Partition Example, lo=3, hi=10

| | A | E | E | T | S | X | O | M | P | L | R |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | i | lo, j | | | | | | | hi |

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=3, hi=10

| | A | E | E | T | S | X | O | M | P | L | R |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | i | lo | j | | | | | | hi |

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=3, hi=10

| | A | E | E | T | S | X | O | M | P | L | R |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | i | lo | | j | | | | | hi |

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=3, hi=10

| | A | E | E | T | S | X | O | M | P | L | R |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | i | lo | | | j | | | | hi |

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=3, hi=10

| | A | E | E | T | S | X | O | M | P | L | R |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | lo, i | | | j | | | | hi |

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=3, hi=10

| | A | E | E | O | S | X | T | M | P | L | R |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | lo, i | | | j | | | | hi |

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=3, hi=10

| | A | E | E | O | S | X | T | M | P | L | R |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | lo, i | | | | j | | | hi |

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=3, hi=10

| | A | E | E | O | S | X | T | M | P | L | R |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | lo | i | | | j | | | hi |

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=3, hi=10

| | A | E | E | O | M | X | T | S | P | L | R |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | lo | i | | | j | | | hi |

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=3, hi=10

| | A | E | E | O | M | X | T | S | P | L | R |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | lo | i | | | | j | | hi |

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=3, hi=10

| | A | E | E | O | M | X | T | S | P | L | R |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | lo | | i | | | j | | hi |

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=3, hi=10

| | A | E | E | O | M | P | T | S | X | L | R |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | lo | | i | | | j | | hi |

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=3, hi=10

| | A | E | E | O | M | P | T | S | X | L | R |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | lo | | i | | | | j | hi |

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=3, hi=10

| | A | E | E | O | M | P | T | S | X | L | R |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | lo | | | i | | | j | hi |

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=3, hi=10

| | A | E | E | O | M | P | L | S | X | T | R |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | lo | | | i | | | j | hi |

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=3, hi=10

| | A | E | E | O | M | P | L | S | X | T | R |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | lo | | | | i | | | j, hi |

```
i++;
E temp = a[i];
a[i] = a[hi];
a[hi] = temp;

return i;
```

# Partition Example, lo=3, hi=10

| | A | E | E | O | M | P | L | R | X | T | S |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | lo | | | | i | | | j, hi |

```
i++;
E temp = a[i];
a[i] = a[hi];
a[hi] = temp;

return i;
```

# Partition Example, lo=3, hi=10

| | A | E | E | O | M | P | L | R | X | T | S |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | lo | | | | i | | | j, hi |

```
i++;
E temp = a[i];
a[i] = a[hi];
a[hi] = temp;
return i;
```

Partition for lo=3, hi=10 is complete. i=7

Call quickSort recursively on left subarray with lo=3, hi = 6.

# Quicksort Example - Sort [O,M,P,L]

```
// quicksort the subarray from a[lo] to a[hi]
private static <E extends Comparable<E>> void quickSort(E[] a, int lo, int hi) {
    if (lo < hi){
       int pivot = partition(a, lo, hi);
       quickSort(a, lo, pivot-1);
       quickSort(a, pivot+1, hi);
    }
}
```

Call partition with lo=3, hi=6

| | A | E | E | O | M | P | L | R | X | T | S |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | lo | | | hi | | | | |

# Partition Example, lo=3, hi=6

| | A | E | E | O | M | P | L | R | X | T | S |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | i | lo | | | hi | | | | |

```
    // partition the subarray a[lo..hi] so that a[lo..pivot-1] <= a[pivot] <= a[pivot+1..hi] and return
the partitioning index pi.
    private static <E extends Comparable<E>> int partition(E[] a, int lo, int hi) {
        int i = lo - 1;
        E pivot = a[hi];
```

# Partition Example, lo=3, hi=6

| | A | E | E | O | M | P | L | R | X | T | S |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | i | lo, j | | | hi | | | | |

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=3, hi=6

| | A | E | E | O | M | P | L | R | X | T | S |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | i | lo | j | | hi | | | | |

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=3, hi=6

| | A | E | E | O | M | P | L | R | X | T | S |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | i | lo | | j | hi | | | | |

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=3, hi=6

| | A | E | E | O | M | P | L | R | X | T | S |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | lo, i | | | j, hi | | | | |

```
i++;
E temp = a[i];
a[i] = a[hi];
a[hi] = temp;

return i;
```

# Partition Example, lo=3, hi=6

| | A | E | E | L | M | P | O | R | X | T | S |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | lo, i | | | j, hi | | | | |

```
i++;
E temp = a[i];
a[i] = a[hi];
a[hi] = temp;

return i;
```

# Partition Example, lo=3, hi=6

| | A | E | E | L | M | P | O | R | X | T | S |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | lo, i | | | j, hi | | | | |

```
i++;
E temp = a[i];
a[i] = a[hi];
a[hi] = temp;

return i;
```

Partition for lo=3, hi=6 is complete. i=3

Call quickSort recursively on right subarray with lo=4, hi = 6.

# Quicksort Example - Sort [M, P, O]

```
// quicksort the subarray from a[lo] to a[hi]
private static <E extends Comparable<E>> void quickSort(E[] a, int lo, int hi) {
    if (lo < hi){
       int pivot = partition(a, lo, hi);
       quickSort(a, lo, pivot-1);
       quicksort(a, pivot+1, hi);
    }
}
```

Call partition with lo=4, hi=6

| | A | E | E | L | M | P | O | R | X | T | S |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | | lo | | hi | | | | |

# Partition Example, lo=4, hi=6

| | A | E | E | L | M | P | O | R | X | T | S |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | i | lo | | hi | | | | |

```
    // partition the subarray a[lo..hi] so that a[lo..pivot-1] <= a[pivot] <= a[pivot+1..hi] and return
the partitioning index i.
    private static <E extends Comparable<E>> int partition(E[] a, int lo, int hi) {
        int i = lo - 1;
        E pivot = a[hi];
```

# Partition Example, lo=4, hi=6

| | A | E | E | L | M | P | O | R | X | T | S |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | i | lo, j | | hi | | | | |

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=4, hi=6

| | A | E | E | L | M | P | O | R | X | T | S |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | | lo, j, i | | hi | | | | |

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=4, hi=6

| | A | E | E | L | M | P | O | R | X | T | S |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | | lo, j, i | | hi | | | | |

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=4, hi=6

| | A | E | E | L | M | P | O | R | X | T | S |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | | lo, i | j | hi | | | | |

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=4, hi=6

| | A | E | E | L | M | P | O | R | X | T | S |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | | lo | i | j, hi | | | | |

```
i++;
E temp = a[i];
a[i] = a[hi];
a[hi] = temp;

return i;
```

# Partition Example, lo=4, hi=6

| | A | E | E | L | M | O | P | R | X | T | S |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | | lo | i | j, hi | | | | |

```
i++;
E temp = a[i];
a[i] = a[hi];
a[hi] = temp;
return i;
```

# Partition Example, lo=4, hi=6

| | A | E | E | L | M | O | P | R | X | T | S |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | | lo | i | j, hi | | | | |

```
i++;
E temp = a[i];
a[i] = a[hi];
a[hi] = temp;
return i;
```

Partition for lo=4, hi=6 is complete. i=5

Call quickSort recursively on right subarray with lo=8, hi = 10.

# Quicksort Example - Sort [X,T,S]

```
// quicksort the subarray from a[lo] to a[hi]
private static <E extends Comparable<E>> void quickSort(E[] a, int lo, int hi) {
    if (lo < hi){
       int pivot = partition(a, lo, hi);
       quickSort(a, lo, pivot-1);
       quickSort(a, pivot+1, hi);
    }
}
```

Call partition with lo=8, hi=10

| | A | E | E | L | M | O | P | R | X | T | S |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | | | | | | lo | | hi |

# Partition Example, lo=8, hi=10

| | A | E | E | L | M | O | P | R | X | T | S |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | | | | | i | lo | | hi |

```
    // partition the subarray a[lo..hi] so that a[lo..pivot-1] <= a[pivot] <= a[pivot+1..hi] and return
the partitioning index i.
    private static <E extends Comparable<E>> int partition(E[] a, int lo, int hi) {
        int i = lo - 1;
        E pivot = a[hi];
```

# Partition Example, lo=8, hi=10

| | A | E | E | L | M | O | P | R | X | T | S |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | | | | | i | lo, j | | hi |

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=8, hi=10

| | A | E | E | L | M | O | P | R | X | T | S |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | | | | | i | lo | j | hi |

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=8, hi=10

| | A | E | E | L | M | O | P | R | X | T | S |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | | | | | | lo, i | | j, hi |

```
i++;
E temp = a[i];
a[i] = a[hi];
a[hi] = temp;

return i;
```

# Partition Example, lo=8, hi=10

| | A | E | E | L | M | O | P | R | S | T | X |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | | | | | | lo, i | | j, hi |

~~i++;~~

```
E temp = a[i];
a[i] = a[hi];
a[hi] = temp;
```

```
return i;
```

# Partition Example, lo=8, hi=10

| | A | E | E | L | M | O | P | R | S | T | X |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | | | | | | lo, i | | j, hi |

```
i++;
E temp = a[i];
a[i] = a[hi];
a[hi] = temp;
```

```
return i;
```

Partition for lo=8, hi=10 is complete. i=8

Call quickSort recursively on right subarray with lo=9, hi = 10.

# Quicksort Example - Sort [T,S]

```
// quicksort the subarray from a[lo] to a[hi]
private static <E extends Comparable<E>> void quickSort(E[] a, int lo, int hi) {
    if (lo < hi){
       int pivot = partition(a, lo, hi);
       quickSort(a, lo, pivot-1);
       quickSort(a, pivot+1, hi);
    }
}
```

Call partition with lo=9, hi=10

| | A | E | E | L | M | O | P | R | S | T | X |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | | | | | | | lo | hi |

# Partition Example, lo=9, hi=10

| | A | E | E | L | M | O | P | R | S | T | X |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | | | | | | i | lo | hi |

```
    // partition the subarray a[lo..hi] so that a[lo..pivot-1] <= a[pivot] <= a[pivot+1..hi] and return
the partitioning index i.
    private static <E extends Comparable<E>> int partition(E[] a, int lo, int hi) {
        int i = lo - 1;
        E pivot = a[hi];
```

# Partition Example, lo=9, hi=10

| | A | E | E | L | M | O | P | R | S | T | X |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | | | | | | i | lo, j | hi |

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=9, hi=10

| | A | E | E | L | M | O | P | R | S | T | X |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | | | | | | | lo, j, i | hi |

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=9, hi=10

| | A | E | E | L | M | O | P | R | S | T | X |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | | | | | | | lo, j, i | hi |

```
for(int j = lo; j < hi; j++ ){
   if(a[j].compareTo(pivot) <= 0 ) {
      i++;
      E temp = a[i];
      a[i] = a[j];
      a[j] = temp;
   }
}
```

# Partition Example, lo=9, hi=10

| | A | E | E | L | M | O | P | R | S | T | X |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | | | | | | | lo | hi, i, j |

```
i++;
E temp = a[i];
a[i] = a[hi];
a[hi] = temp;

return i;
```

# Partition Example, lo=9, hi=10

| | A | E | E | L | M | O | P | R | S | T | X |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | | | | | | | lo | hi, i, j |

```
i++;
E temp = a[i];
a[i] = a[hi];
a[hi] = temp;
return i;
```

# Partition Example, lo=9, hi=10

| | A | E | E | L | M | O | P | R | S | T | X |
|---|---|---|---|---|---|---|---|---|---|---|---|
| -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | | | | | | | | | lo | hi, i, j |

```
i++;
E temp = a[i];
a[i] = a[hi];
a[hi] = temp;

return i;
```

Partition for lo=9, hi=10 is complete.
i=10

Entire array has been sorted (we don't call quickSort on arrays of size 1)

# Great algorithms are better than good ones

- Your laptop executes $10^8$ comparisons per second
- A supercomputer executes $10^{12}$ comparisons per second

| | Insertion sort | | | Mergesort | | | Quicksort | | |
|---|---|---|---|---|---|---|---|---|---|
| **Computer** | **Thousand inputs** | **Million inputs** | **Billion inputs** | **Thousand inputs** | **Million inputs** | **Billion inputs** | **Thousand inputs** | **Million inputs** | **Billion inputs** |
| **Home** | Instant | 2 hours | 300 years | instant | 1 sec | 15 min | Instant | 0.5 sec | 10 min |
| **Supercomputer** | Instant | 1 sec | 1 week | instant | instant | instant | instant | instant | instant |

# Quicksort analysis: best case

- Quicksort divides everything exactly in half.
- Similar to merge sort.
- Number of compares is $\sim n \log n$.

# Quicksort analysis: worst case

- Data are already sorted or we pick the smallest or largest key as pivot.
- Number of compares is ~$n^2$ - quadratic!
- Extremely unlikely (less likely than the probably that your computer is struck by lightning) if we first shuffle and our shuffling is not broken.

# Things to remember about quick sort

- 39% more compares than merge sort but in practice it is faster because it does not move data much.
  - If good implementation, even in sorted arrays it can be linearithmic. If not, we end up with quadratic.
- $O(n \log n)$ average, $O(n^2)$ worst, in practice faster than mergesort.
- In-place sorting.
- **Not** stable.

# Quicksort practical improvements

- Use insertion sort for small subarrays.
- Best choice of pivot is the median of a small sample.

- For years, Java used quicksort for collections of primitives and mergesort for collections of objects due to stability.
  - Has moved to dual-pivot quick sort (Yaroslavskiy, Bentley, and Bloch, 2009) and timsort (Peters, 1993), respectively.

# Sorting: the story so far

| Which Sort | In place | Stable | Best | Average | Worst | Remarks |
|---|---|---|---|---|---|---|
| Selection | X | | $O(n^2)$ | $O(n^2)$ | $O(n^2)$ | $n$ exchanges |
| Insertion | X | X | $O(n)$ | $O(n^2)$ | $O(n^2)$ | Use for small arrays or partially ordered |
| Merge | | X | $O(n \log n)$ | $O(n \log n)$ | $O(n \log n)$ | Guaranteed performance; stable |
| Quick | X | | $O(n \log n)$ | $O(n \log n)$ | $O(n^2)$ | $n \log n$ probabilistic guarantee; fastest in practice |

# Lecture 13: Quicksort

- This week's assignment
- Quicksort

# Readings:

- Recommended Textbook:
  - Chapter 2.3 (Pages 288-296)
- Recommended Textbook Website:
  - Quicksort: https://algs4.cs.princeton.edu/23quicksort/
    - We use a different implementation

# Code

- Lecture 13 code

# Practice Problem

- What would the resulting array for the first call to partition be for the following array: [E,A,S,Y,Q,U,E,S,T,I,O,N].

# ANSWER

- What would the resulting array and new pivot index for the first call to partition be for the following array: [E,A,S,Y,Q,U,E,S,T,I,O,N]
- [E, A, E, I, N, U, S, S, T, Y, O, Q] and pivot: at index 4.