CSE12W19-Feb11-W6-M-14-1

Timing channels / timing attacks

	Insertion	Selection	Merge	Quick
Best case time	Sorted array $\Theta(n)$	$\Theta(n^2)$	$\Theta(n * \log_2(n))$	$\Theta(n * \log_2(n))$
Worst case time	Reverse-sorted array $\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n * \log_2(n))$	$\Theta(n^2)$ *avg $\Theta(n * \log_2(n))$
Key operations	`swap(a, j, j-1)` (until in the right place)	`swap(a, i, indexOfMin)` (after finding minimum value)	`l = copy(a, 0, len/2)` `r = copy(a, len/2, len)` `ls = sort(l)` `rs = sort(r)` `merge(ls, rs)`	`p = partition(a, l, h)` `sort(a, l, p)` `sort(a, p + 1, h)`

$\{1, 4, 3, 2\}$ $\{1, 4, 5, 2\}$

- keep sorted flag

- if i = minIndex, break

- 2 indices

$i=1$: $7<8$

$i=2$: $6<8$, $6<7$

$i=3$: $5<8$, $5<7$, $5<6$

```
import java.util.Arrays;
public class Sort {
static void selectionSort(int[] arr) {          -1
  for(int i = 0; i < arr.length; i += 1) {
    int minIndex = i;
    for(int j = i; j < arr.length; j += 1) {
      if(arr[minIndex] > arr[j]) { minIndex = j; }
    } else {break;} ✗        else { j = arr.length; } ✗
    int temp = arr[i];
    arr[i] = arr[minIndex];
    arr[minIndex] = temp;
  }
}

static void insertionSort(int[] arr) {
  for(int i = 0; i < arr.length; i += 1) {
    for(int j = i; j > 0; j -= 1) {
      if(arr[j] < arr[j-1]) {
        int temp = arr[j-1];
        arr[j-1] = arr[j];
        arr[j] = temp;
      }
      else { break; } // new! exit inner loop early
    }
  }
}
}
```

n   1

8 7 6 5 3

3 5 6 7 8

$i=0$

$i=1$: $5<3$ break ✓

$i=2$: $6<5$ break ✓

$i=3$: $7<6$ break ✓

Best case?

A: Array in decreasing order

[✓ B: Array in increasing order]

[✓ C: Array of all equal elements]

D: Something else

E: More than one of A-C

CSE12W19-Feb11-W6-M-14-2

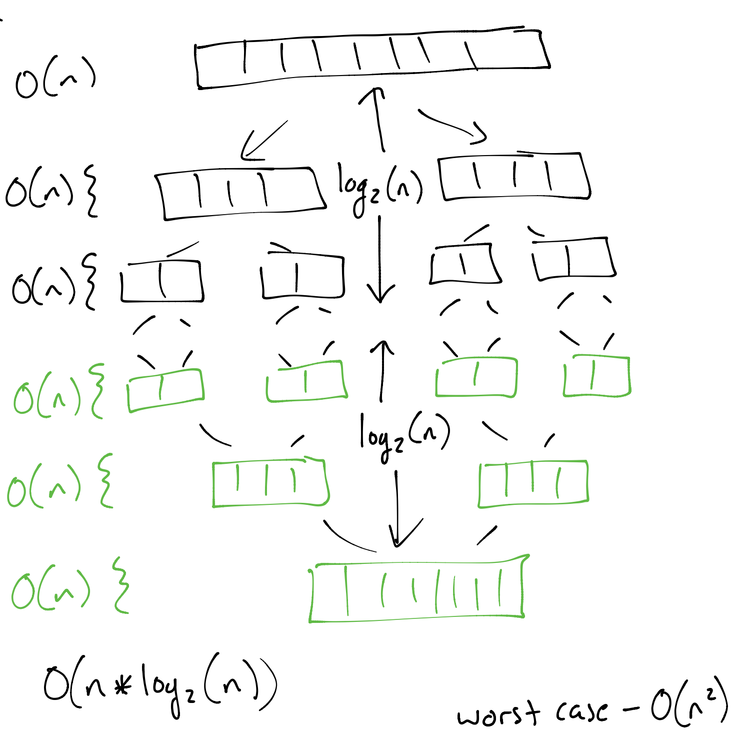
Divide and Conquer

```
import java.util.Arrays;
public class SortFaster {

  static int[] combine(int[] p1, int[] p2) {...}

  static int[] mergeSort(int[] arr) {
    int len = arr.length
    if(len <= 1) { return arr; }
    else {
      int[] p1 = Arrays.copyOfRange(arr, 0, len / 2);
      int[] p2= Arrays.copyOfRange(arr, len / 2, len);
      int[] sortedPart1 = mergeSort(p1);
      int[] sortedPart2 = mergeSort(p2);
      int[] sorted = combine(sortedPart1, sortedPart2);
      return sorted;
    }
  }
```

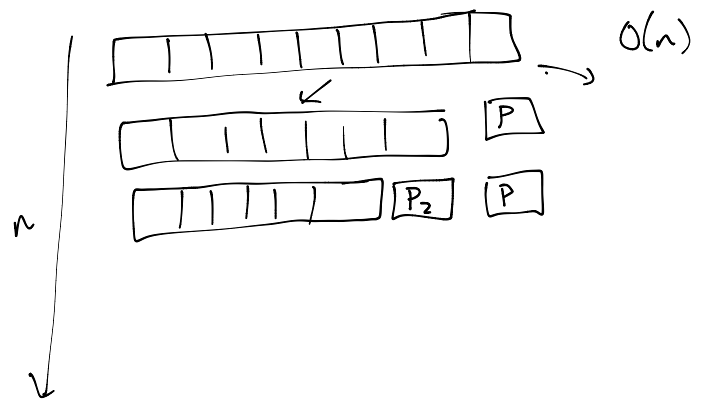
merge

Pivot:
- median value → can you do median in $O(n)$
- random index
- use min/max
- $(high - low)/2 + low$

```
  static int partition(String[] array, int l, int h) {...}

  static void qsort(String[] array, int low, int high) {
    if(high - low <= 1) { return; }
    int splitAt = partition(array, low, high);
    qsort(array, low, splitAt);
    qsort(array, splitAt + 1, high);
  }

  public static void sort(String[] array) {
    qsort(array, 0, array.length);
  }

}
```

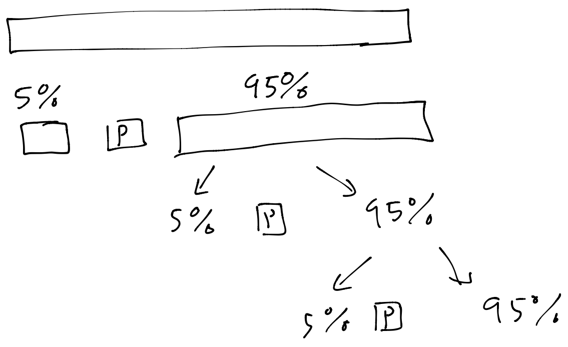


worst case – $O(n^2)$



```
interface Map<Key, Value> {

}
```

Randomized Algorithm



$n * .95$ $\log_{\frac{1}{.95}}(n)$

$n * 0.5$

$\log_{\frac{1}{0.5}}(n)$ $\log_2(n)$