```
String name = ...
println ("Hello, my name is " + name);
```

append strings

HW

first call → "(+ 1 2 (* 1 2) 3)" ← first (again)

second → 1

first call:

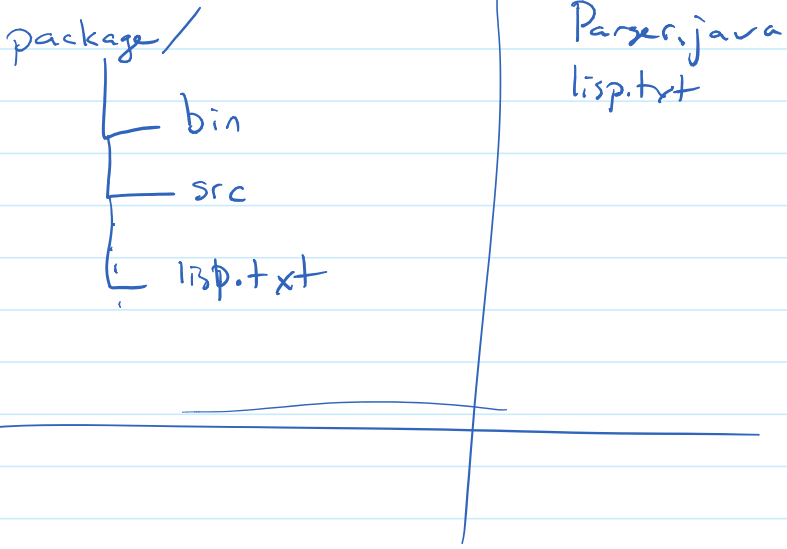
"(+ 1 2 2 3)" → eval → 8

second call:

"(* 1 2)" → eval → 2

first call: "..."

second: "(..."



Reading Files (review)

```
Scanner scan = new Scanner (System.in)
```

```
Scanner scan = new Scanner( new File("name.txt"));
```

java.io.File

```
Scanner scan = null;
try {
    scan = new Scanner(new File(...));
} catch (FileNotFoundException e) {
    print ("File not found");
    exit();
}
... scan.nextLine()
    scan.hasNextLine()
```

java.io.FileNotFoundExc...

Lab 04

```
parseSExp (Scanner scan, ..., Repl repl)
```

Lisp implements Repl

Sorting

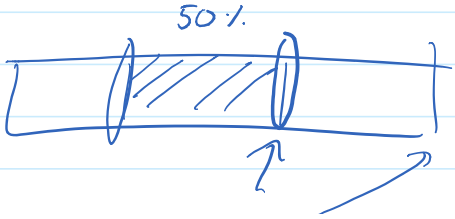
```
quicksort ( arr ): //Not in place
    choose a pivot
    (left, right) = partition(arr, pivot)    O(n)
    quicksort(left)
    quicksort (right)
    return  left + [pivot] + right
```

a list only containing pivot

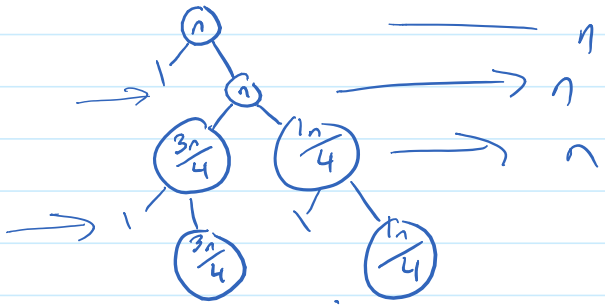
partition moves elements smaller than pivot into left
" " larger " " " right

Recall

worst	$O(n^2)$
best	$O(n \lg n)$
ave	$O(n \lg n)$



best	$\Theta(n \lg n)$
avg	$\Theta(n \lg n)$



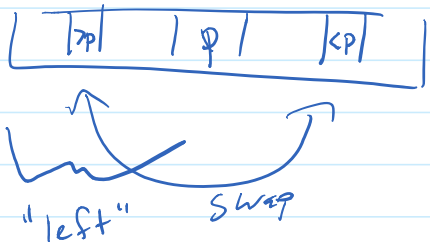
height $\sim ? \log_{4/3} n$

$\Theta(2n \log_{4/3} n)$

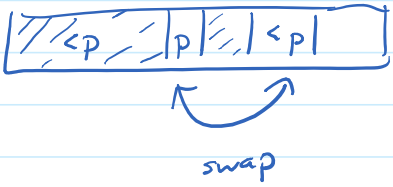
$\Theta(n \lg n)$

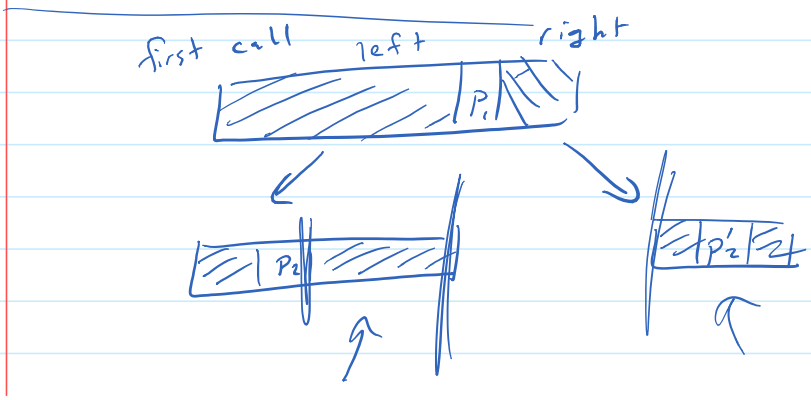
Better In place

case 1:



case 2:

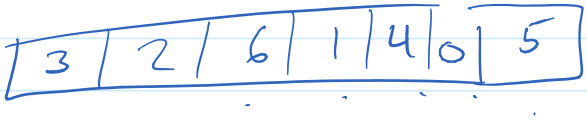




```
quicksort(arr, start, end): // "destructive"
    choose a pivot within the bounds
    (lstart, lend, rstart, rend) = partition(arr, pivot, start, end)
    quicksort(arr, lstart, lend)
    quicksort(arr, rstart, rend)
    return arr
quicksort(arr, 0, arr.size());
          incl.  excl.
```

partition

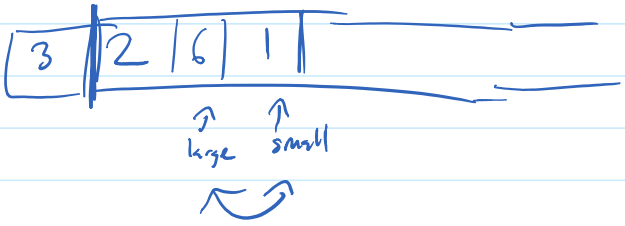
run through the list with a pointer to two elements, making swaps as necessary

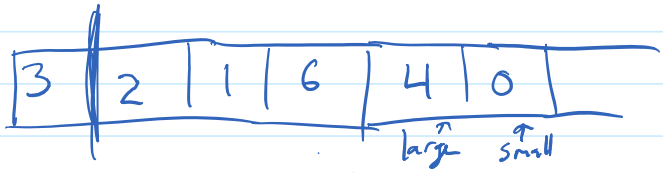


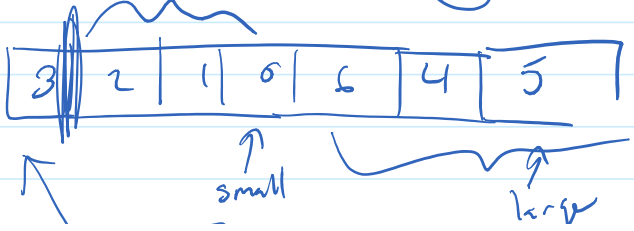
choose the pivot as the first element

first call

3 pivot





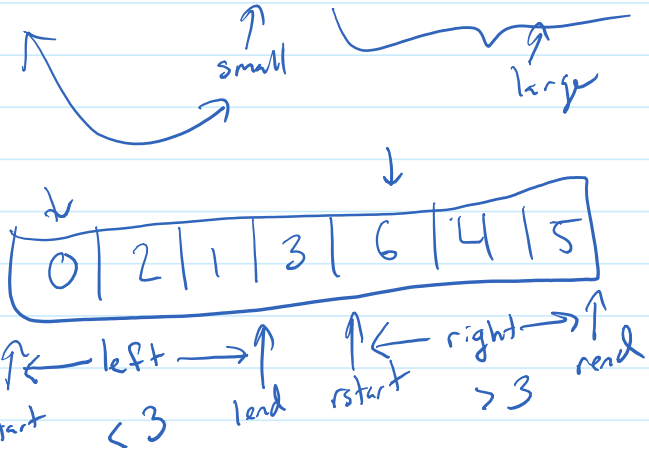


small is small compared to pivot

small = -∞

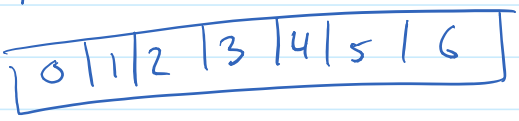
large = -∞

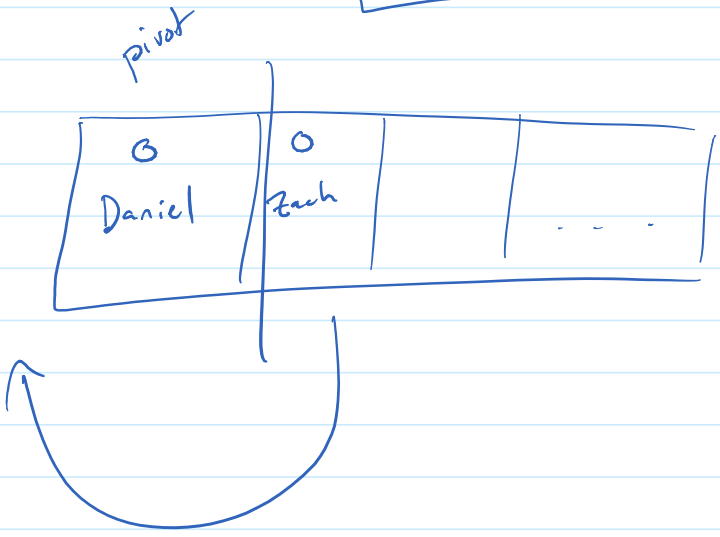
for (elem : list)




```
large = -∞
for (elem : list)
    if (elem < pivot)
        small = elem
    if (elem > pivot)
        large = elem
    : do some swaps
```

```
quicksort (left)
quicksort (right)
```





Three (common/obvious) choices

- First element // sorted or reverse
- Last element // sorted or reverse
- Middle element

Randomized Quicksort

same asymptotics

Merge Sort (Divide & Conquer)

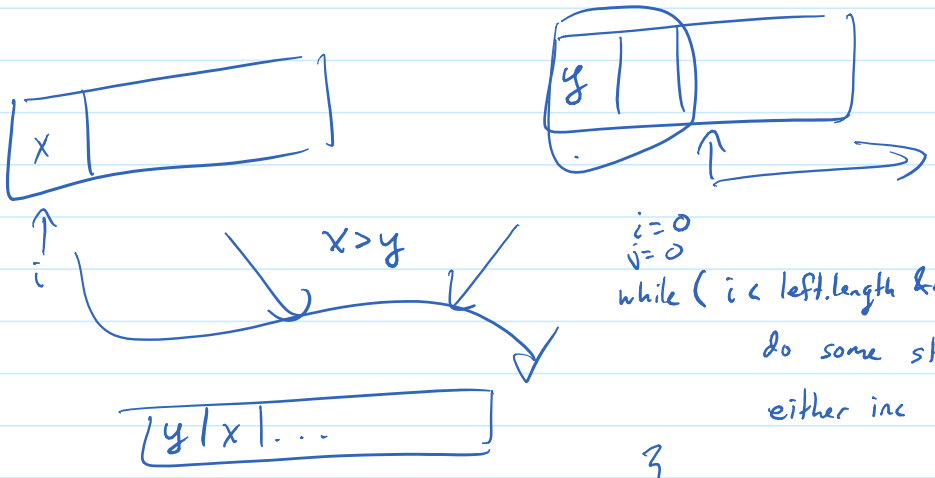
```
Mergesort (arr):
```

(left, right) = divide(arr)

Mergesort (left)

Mergesort (right)

return merge (left, right) $O(n)$




```
i = 0
j = 0
while ( i < left.length && j < right.length)
    do some stuff
    either inc i or j
}
```