CS-21 LECTURE #4

QUICK SORT:

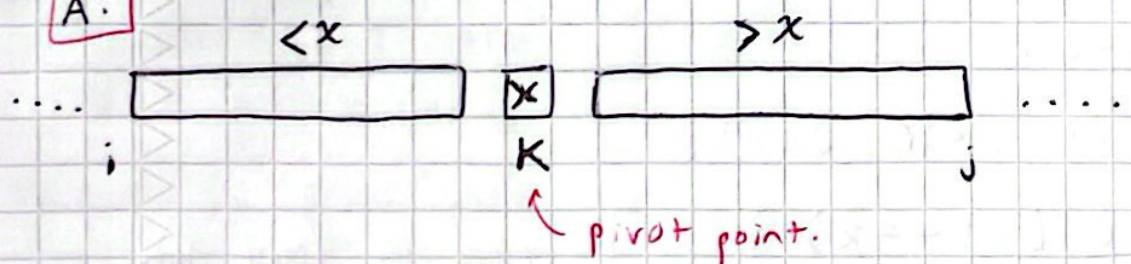
└ Divide & conquer algorithm: Breaks Big problem into smaller sub problems.

Divide : partition (re-arrange) an array $A[i \ldots j]$ into two (possibly one, if one is empty) sub arrays.

$A[i \ldots k-1]$ & $A[k+1 \ldots j]$

such that all elements in $A[i \ldots k-1]$ are less than $A[k]$ & all elements in $A[k+1 \ldots j]$ are greater than $A[k]$

A:



* Finding "x" (at A[k]) i.e. the pivot is "the hard part"

Conquer: solve two array halves recursively.

Combine: Already done; no action needed.

array — left index — right index

```
Quicksort (A, p, r) {
    if (p < r) {
        q = partition (A, p, r);      ← Picks the pivot...
        Quicksort (A, p, q-1);
        Quicksort (A, q+1, r);
    }
}
```

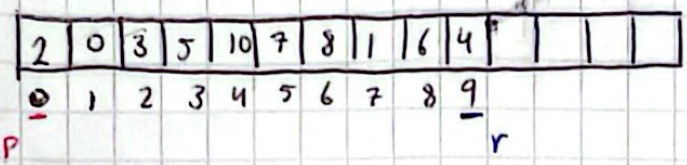
All a wrapper function that calls quicksort

```
partition (A, p, r) {
    x = A[r];                         // select last element as pivot...
    i = p - 1;                        // index just before range
    for (j = p; j < r; j++) {         // for loop from p stop b4 r
        if (A[j] <= x) {              // Look for small item
            i++;                      // increment i, then swap
            swap (A[i], A[j]);
        }
    }
    swap (A[i+1], A[r]);              // swap pivot
    return i+1;                       // return index of pivot
}
```

find pivot

THIS IS THE LOMUTO QUICK SORT!!!

Not good with duplicates...



Quicksort(A, 0, 9)

x pivot = 4 i = -1

i = 0

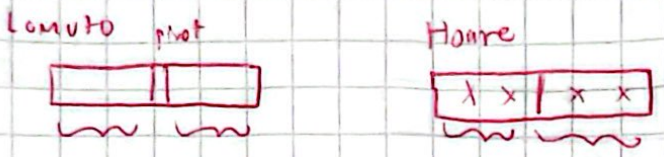
swap(A[0], A[0])

i = 1

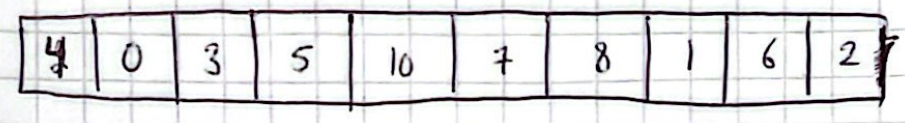
Median of 3 Strategy ← NOT FOR PRE-SORTING ELEMENTS!

REQUIRED TO IMPLEMENT!!!

- Select val @ p & r, and $\frac{p+r}{2}$ ← Midpoint between p & r
- Compare A[p] A[mid] & A[r] then find median value & swap into A[r]
- Before you select pivot, run median of 3 function
  - Purpose: Return index of median value then swap!
- To test if M3 works, see if sorted data & unsorted data finish setting sorted in same time.




```
Hoare_Partition (A, p, r) {
    x = A[p];    // first elm is the pivot...
    i = p-1 ;
    j = r+1 ;    // two indexes outside of p-r range...
    while (true) {
        do {
            ~~j = j + 1~~
            i = i + 1;
        } while (A[i] < x)
        do {
            j = j - 1;
        } while (A[j] > x)
        if ( i >= j)
            return j ;
        Swap (A[i] A[j]);
    } // while (true)
}
```



i → -1

j → 10