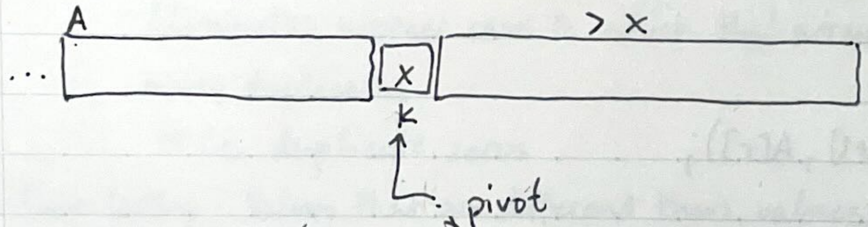
AS03 - Quicksort
↳ Is "shorter" than heap assignment

Will talk about whether a sort is stable next week

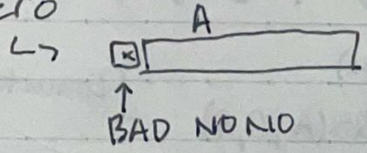
Lumuto Quicksort
- Worst case => $O(n^2)$ when the data is already sorted, which is common. Can account for this using mo3 (median_of_three)
- Other case => $O(n \log n)$ if you choose a good pivot
- Sorts in place (unlike merge sort
- Divide & Conquer approach (like merge sort
-

STEPS: (reorganize)
1) Partition ^array $A[i \ldots j]$ into two (possibly one, if one is empty) sub arrays $A[i \ldots k-1]$ and $A[k+1 \ldots j]$ such that all elements in $A[i \ldots k-1]$ are less than $A[k]$ and all elements in $A[k+1 \ldots j]$ are greater than $A[k]$.



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

Conquer: Sort two array halves recursively
Combine: already done; no action needed
- Quick sort can handle two arrays that are inequal sizes
- Worst case is when it's at zero



Quicksort question (implementing Lomuto ver as well) will be on the exam (can implement the Hoare ver)
Study Lomuto ver of quicksort

PSEUDO CODE

Lomuto Ver. of Quicksort

```
quicksort (A, p, r) {          //A is an array, p is left index, r is right index
  if (p<r) {
    q = partition(A, p, r);
    quicksort (A, p, q-1);     // do not "look ahead"
    quicksort (A, q+1, r);
  }
}
```

```
partition (A, p, r)            <- before you select pivot, run mo3 -> & then swap
Find point ->            x = A[r];          // select last element as pivot
index, just
before range ->          i = p-1;           // index (of smaller region)
For loop from p,
stop before r            for (j = p; j < r; j++) {
                           if (A[j] <= x) {
look for a small             i++;
item, incr. i                swap (A[i], A[j]);
then swap                  }
                         }
swap pivot ->            swap (A[i+1], A[r]);
Return index ->          return i+1;
}
```

Quick Sort Ex

A

2	0	3	5	10	7	8	1	6	4
0	1	2	3	4	5	6	7	8	9

x pivot 4

↑
i=-1

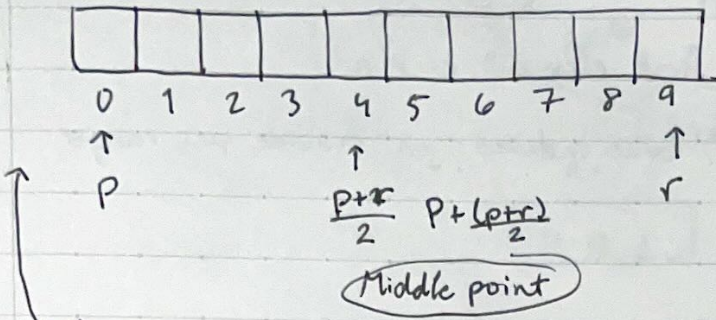
QS (4)
↳ QS(A, 0, 9)
] Lomuto version of quicksort

i=0 $\xrightarrow{i++}$ i=1

*Might ask to write the code for mo3 for exam

treat inputs as ints for p3

MEDIAN OF 3 STRATEGY (to prevent; wc selection of pivot)



Compare A[p], A[mid], A[r]
↳ Find median value & swap into A[r]
- Purpose: return index of median value.

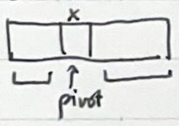
*Don't use mo3 to presort elements, very bad
↳ Don't have it sort the largest value to the end

how to determ a const for mo3, try 10, 100, 1000
- Make big moves to narrow down a size to use mo3
↳ Have a README file explaining your reasoning
*Make sure you're using p&r
↳ BC you're using p&r for your size
↳ EC. is not testing it w/ mo3

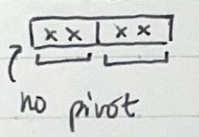
HOARE PARTITION
↳ Why?
- Eliminates worser case in which the array to sort has many duplicates
↳ i.e. duplicate zeros

Base Testing: Values that are different then values that repeat ← For Lomuto QS

Lomuto:



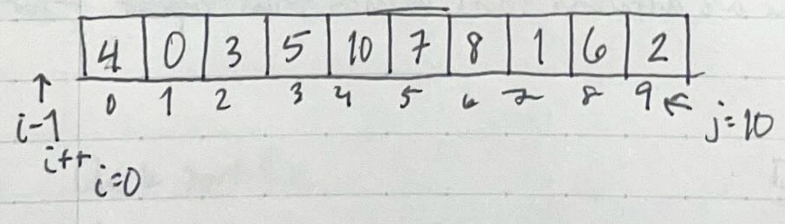
Hoare:



Quick sort hungarian dance

PSEUDO CODE

```
hoare_partition(A, p, r){
        int x = A[p];  // first element is pivot
        int i = p-1;  ] // two indexes, just outside per range
        int j = r+1;
Is inf. loop → while (true) {
            do {
                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
```



x is 4