Announcements

Programming assignment #2 ⇒ Due 02.29.24

- Just follow the the instructions as stated in the assignment, do not do anything fancy

Programming assignment #3 ⇒ Due 03.07.24

- Has some analysis (20 points)
- Available extra credit (+20 points)

Lumuto quicksort

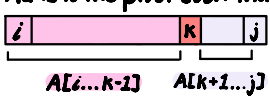
Overview

- Worst case ⇒ $O(n^2)$ when the data is already sorted, which is common. Can account for this using mo3
- Other case ⇒ $O(n \lg n)$ if you choose a good pivot
- Sorts in place (unlike merge sort)
- Divide and conq approach (like merge sort)
- Sedgewick did his PhD on the speed of quick sort in 1975. Also median-of-3

Not in textbook, in assignment

Steps

1) Partition array A[i...j] into two regions A[i...k-1] and A[k+1...j] where A[k] is the pivot such that all elements in A[i...k-1] < A[k] and all elements A[k+1...j] > A[k]. Computing k is part of the problem



2) Call quicksort recursively on each of the two sub-arrays

Pseudo code

```
quicksort(A,p,r) {    //A is an array, p is left and r is right
    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){
    med = med_of_3(A,p,r);    //do not use mo3 if you have some number of elements, TBD
    swap(A[med], A[r]);
    x = A[r]    //select pivot
    i = p-1    //boundry of "less than x region", initially out of bounds
    for (j = p; j<r; j++){
        if(A[j] < x){
            i++;
```

pivot

```
            swap(A[i], A[j]);
        }
    }
    swap(A[i+1], A[r]);
    return i+1
}
```

Prevent a bad pivot

* Use median of three technique (m-o-3).
  - Compare A[p], A[r], A[p+((r-p)/2)] (prevent overflow) and return index of median
  - Should only do this when the array to sort is greater than some size
    ↳ Determine by testing

Hoare partition

Why

- Eliminates worser case in which the array to sort has many duplicates

Pseudo code

```
hoare_partition(A, p, r) {
    //min of three
    int x = A[p];
    int i = p-1;
    int j = r+1;
    while(true) {
        do {i = i+1;} while (A[i] < x);
        do {j = j-1;} while (A[i] > x);
        if (i >= j) return;
        swap(A[i], A[j]);
    }
}
//Make sure to adjust quicksort() to not omit one position
```