COP 3502 10/20/25

1) Mid Terms back Recitation, Grades Entered
2) Create formula post on Webcourses w/ grade lines. (by T or W)
2.5) Merge Sort run-time analysis
3) Quick Sort
   - alg on paper
   - time analysis
   - Simulation

Merge Sort

Let $T(n)$ = run time of Merge Sort on n elements

MergeSort(left-half) ✓
MergeSort(right-half) ✓
Merge(left, right) $n, m \rightarrow O(n+m)$

$n/2$ $n/2$

$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + O(n)$$

MS ½ elems

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

Master Theorem: $A=2, B=2, k=1$

$B^k = 2, A = 2 \rightarrow O(n^{\log_2 2} \cdot \lg n)$

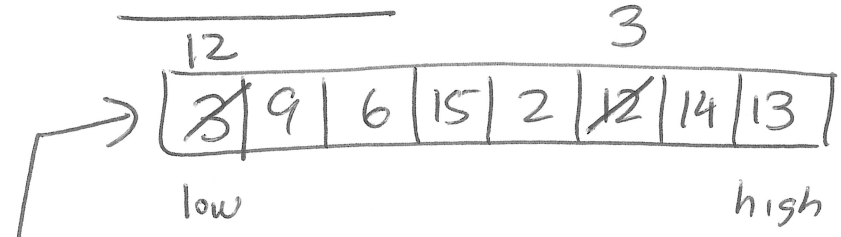
$= \boxed{O(n \lg n)}$

Best, Avg, Worst

optimization → base case if (high-low < 30)

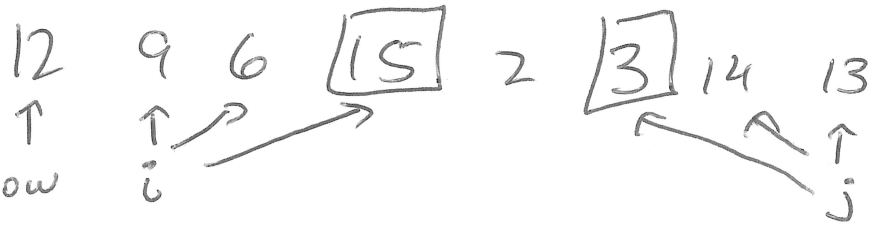
Quick Sort

Partition



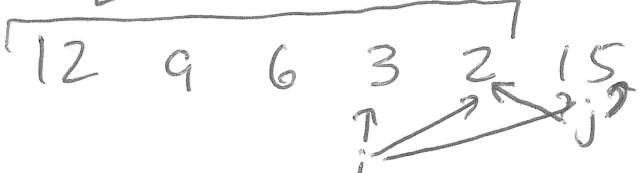
Goal: Split subarray into 2 groups (Partition)

1. Pick a ~~partition~~ partition element.
   a. Use leftmost element
   b. Pick a random element
   c. Pick 3 or 5 random elements + pick the median of those.
2. Swap it into leftmost place



while(i<=j) in bigger loop

```
while (i <=high && a[i]=< a[low]) i++;
while (j >=low && a[j] > a[low]) j--;
if    (i <j) swap(&a[i], &a[j]);
```




```
swap(&a[low], &a[j]);
```

2 9 6 3 [12] 15 14 13 `return j;`

sort sort

```
QuickSort (int * a, int low, int high) {
    if (low >= high) return;
    int mid = partition(a, low, high);
    QuickSort (a, low, mid-1);
    QuickSort (a, mid+1, high);
}
```

Best Case → even split ✓

similar to Merge Sort $T(n) = O(n) + T(\frac{n}{2}) + T(\frac{n}{2})$

$O(n \log n)$

Worst Case → split 0 and n-1 items each time

$T(n) = O(n) + T(n-1)$

If you use iteration, you can prove $O(n^2)$

$T(n) = T(n-1) + f(n)$

Sol $\sum_{i=1}^{n} f(i)$

Avg Case turns out to be $O(n \log n)$ will prove Wed.

In theory quick sort should be slower than merge sort due to uneven split potential.

In practice, quick sort is faster because of a lower ~~con~~ hidden constant, since partition works in place and merge doesn't.

Merge Sort is a stable sort but Quick Sort isn't.

In a stable sort if before the sort, $a[i] == a[j]$ and $i < j$, then after the sort $a[i]$ will still be in a lower index than $a[j]$.