Sorting

THE SORTING SYSTEM

Because a school establishing cliques doesn't cause any problems.
Merge sort

1. Split pile in half

2. Sort each half (possibly recursively with merge sort)

3. Recombine lists
Merge sort
Merge sort
Merge sort
Merge sort
Merge sort
Merge sort
Merge sort
Merge sort
Merge sort
Merge sort

Merge(L[1, ..., n_l], R[1, ..., n_r])

i=1, j=1, k=1

while i < n_l OR j < n_r

    if L[i] < R[j]
        A[k] = L[i], i=i+1
    else
        A[k] = R[j], j=j+1

k = k+1
Merge sort

Sort: \{4, 5, 3, 8, 1, 6, 2\}
Sort: {4, 5, 3, 8, 1, 6, 2} - Split
{4, 5, 3} {8, 1, 6, 2} - Split
{4, 5} {3} {8, 1} {6, 2} – Split
{4} {5} {3} {8} {1} {6} {2} – Merge
{4, 5} {3} {1, 8} {2, 6} – Merge
{3, 4, 5} {1, 2, 6, 8} – Merge
{1, 2, 3, 4, 5, 6, 8}
Merge sort

Correctness:
Base: A[] empty (sorted), at L&R[1]
Step: In the while loop, the smallest element in L[] or R[] will be added as the largest element in A[]
Termination: while loop end after all elements in L[] and R[] have been added to A[]
Merge sort

Run time:
\[ T(n) = \]
Merge sort

Run time: (recurrence relation)

\[ T(n) = \begin{cases} 
O(1) & \text{if } n=1, \\
& \text{otherwise} \\
& \text{Divide} + 2T(n/2) + \text{Merge} 
\end{cases} \]

\[ T(n) = \begin{cases} 
O(1) & \text{if } n=1, \\
& \text{otherwise} \\
& O(1) + 2T(n/2) + O(n) 
\end{cases} \]

\[ T(n) = O(n \log n) \]
Divide & conquer

Master's theorem: (proof 4.6)
For $a \geq 1$, $b \geq 1$, $T(n) = a \cdot T(n/b) + f(n)$

If $f(n)$ is... (3 cases)
- $O(n^c)$ for $c < \log_b a$, $T(n)$ is $\Theta(n^{\log_b a})$
- $\Theta(n^{\log_b a})$, then $T(n)$ is $\Theta(n^{\log_b a} \log n)$
- $\Omega(n^c)$ for $c > \log_b a$, $T(n)$ is $\Theta(f(n))$
Master's theorem: TL;DR

If you have something of the form:

\[ T(n) = a \cdot T(n/b) + f(n) \]

acts like \( n^{\log_b a} \)

Case 1: \( f(n) \) grows faster, then
overall growth just \( f(n) \)

Case 2: \( n^{\log_b a} \) grows faster, then
overall growth just \( n^{\log_b a} \)

Case 3: Both grow same, tack on \( \lg n \):
\( n^{\log_b a} \cdot \lg(n) \)
Master's theorem

What are the running times of...

(1) \( T(n) = 4T(n/2) + n^2 \)

(2) \( T(n) = 4T(n/4) + n^2 \)

(3) \( T(n) = 8T(n/2) + n^2 \)
Master's theorem

What are the running times of...
(1) \( T(n) = 4T(n/2) + n^2 \)
    \( \mathcal{O}(n^2 \lg(n)) \)
(2) \( T(n) = 4T(n/4) + n^2 \)
    \( \mathcal{O}(n^2) \)
(3) \( T(n) = 8T(n/2) + n^2 \)
    \( \mathcal{O}(n^3) \)
Important note on “significantly”: must grow a power larger

\[ n^2 \text{ vs. } n^3 = \text{“significant”} \]
\[ n^2 \text{ vs. } n^{2.0000001} = \text{“significant”} \]

\[ n^2 \text{ vs. } n^2 \log(n) = \text{NOT “significant”} \]
Which works better for multi-cores: insertion sort or merge sort? Why?
Divide & conquer

Which works better for multi-cores: insertion sort or merge sort? Why?

Merge sort! After the problem is split, each core and individually sort a sub-list and only merging needs to be done synchronized.
Quicksort

1. Pick a pivot (any element!)

2. Sort the list into 3 parts:
   - Elements smaller than pivot
   - Pivot by itself
   - Elements larger than pivot

3. Recursively sort smaller & larger
Quicksort

Pivot

Larger

Smaller
Quicksort

Partition(A, start, end)
x = A[end]
i = start – 1
for j = start to end -1
    if A[j] ≤ x
        i = i + 1
        swap A[i] and A[j]
    swap A[i+1] with A[end]
Quicksort

Sort: \{4, 5, 3, 8, 1, 6, 2\}
Quicksort

Sort: {4, 5, 3, 8, 1, 6, 2} – Pivot = 2
{4, 5, 3, 8, 1, 6, 2} – sort 4
{4, 5, 3, 8, 1, 6, 2} – sort 5
{4, 5, 3, 8, 1, 6, 2} – sort 3
{4, 5, 3, 8, 1, 6, 2} – sort 8
{4, 5, 3, 8, 1, 6, 2} – sort 1, swap 4
{1, 5, 3, 8, 4, 6, 2} – sort 6
{1, 5, 3, 8, 4, 6, 2},{1, 2, 5, 3, 8, 4, 6}
For quicksort, you can pick any pivot you want.

The algorithm is just easier to write if you pick the last element (or first).
Quicksort

Sort: \{4, 5, 3, 8, 1, 6, 2\} - Pivot = 3

\{4, 5, 2, 8, 1, 6, 3\} – swap 2 and 3

\{4, 5, 2, 8, 1, 6, 3\}

\{4, 5, 2, 8, 1, 6, 3\} – swap 2 & 4

\{2, 5, 4, 8, 1, 6, 3\} (first red ^)

\{2, 1, 4, 8, 5, 6, 3\} – swap 1 and 5

\{2, 1, 4, 8, 5, 6, 3\}\{2, 1, 3, 8, 5, 6, 4\}
Quicksort

Correctness:
Base: Initially no elements are in the “smaller” or “larger” category
Step (loop): If $A[j] < \text{pivot}$ it will be added to “smaller” and “smaller” will claim next spot, otherwise it stays put and claims a “larger” spot
Termination: Loop on all elements...
Quicksort

Runtime:
Worst case?

Average?
Quicksort

Runtime:
Worst case?
Always pick lowest/highest element, so $O(n^2)$

Average?
Quicksort

Runtime:
Worst case?
Always pick lowest/highest element, so $O(n^2)$

Average?
Sort about half, so same as merge sort on average
Quicksort

Runtime:
Worst case?
Always pick lowest/highest element, so $O(n^2)$

Average?
Sort about half, so same as merge sort on average
Can bound number of checks against pivot:

Let $X_{i,j} = \text{event } A[i] \text{ checked to } A[j]$

$\sum_{i,j} X_{i,j} = \text{total number of checks}$

$E[\sum_{i,j} X_{i,j}] = \sum_{i,j} E[X_{i,j}]$

$= \sum_{i,j} \Pr(A[i] \text{ check } A[j])$

$= \sum_{i,j} \Pr(A[i] \text{ or } A[j] \text{ a pivot})$
Quicksort

= \sum_{i,j} \Pr(A[i] \text{ or } A[j] \text{ a pivot})

= \sum_{i,j} \left(\frac{2}{j-i+1}\right) \quad \text{// } j-i+1 \text{ possibilities}

< \sum_i O(lg n)

= O(n \ lg \ n)
Quicksort

Which is better for multi core, quicksort or merge sort?

If the average run times are the same, why might you choose quicksort?
Quicksort

Which is better for multi core, quicksort or merge sort? Neither, quicksort front ends the processing, merge back ends.

If the average run times are the same, why might you choose quicksort?
Quicksort

Which is better for multi core, quicksort or merge sort? Neither, quicksort front ends the processing, merge back ends

If the average run times are the same, why might you choose quicksort? Uses less space.
So far we have been looking at comparative sorts (where we only can compute $<$ or $>$, but have no idea on range of numbers)

The minimum running time for this type of algorithm is $\Theta(n \lg n)$