COP 3502 3/5/26

Efficient Sorts	Avg Case	Worst Case
1) Merge Sort	O(nlgn)	O(nlgn)
2) Quick Sort	O(nlgn)	$O(n^2)$

It's impossible to comparison sort faster than O(nlgn). [Prove any comparison sort $\Omega(n \lg n)$.]

Merge Sort

8 3 1 7 | 2 6 4 5 } Sort left, right each list n/2

1 3 7 8 | 2 4 5 6

→ 1 2 3 4 5 6 7 8 O(n) runtime

Merge of 2 lists sizes n, m is O(n+m) or O(max(n,m))

Intuitive way int len1 int len2

int* merge(int* a, int* b)

Merge implementation

void merge(int* array, int low, int mid, int end)

* Can't do in place! MALLOC temp array size end-low+1.

Because we allocate an extra array to do merge it's $O(n)$ but w/a higher constant.

```
void mergeSort (int* array, int lowIdx, int highIdx) {
    if (lowIdx >= highIdx) return;
    int mid = (lowIdx + highIdx)/2;
    mergeSort (array, lowIdx, mid);      ] #1
    mergeSort (array, mid+1, highIdx);   ] #2
    merge (array, lowIdx, mid, highIdx);
}
```

Let $T(n)$ = run tm of Merge Sort of n elems.

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

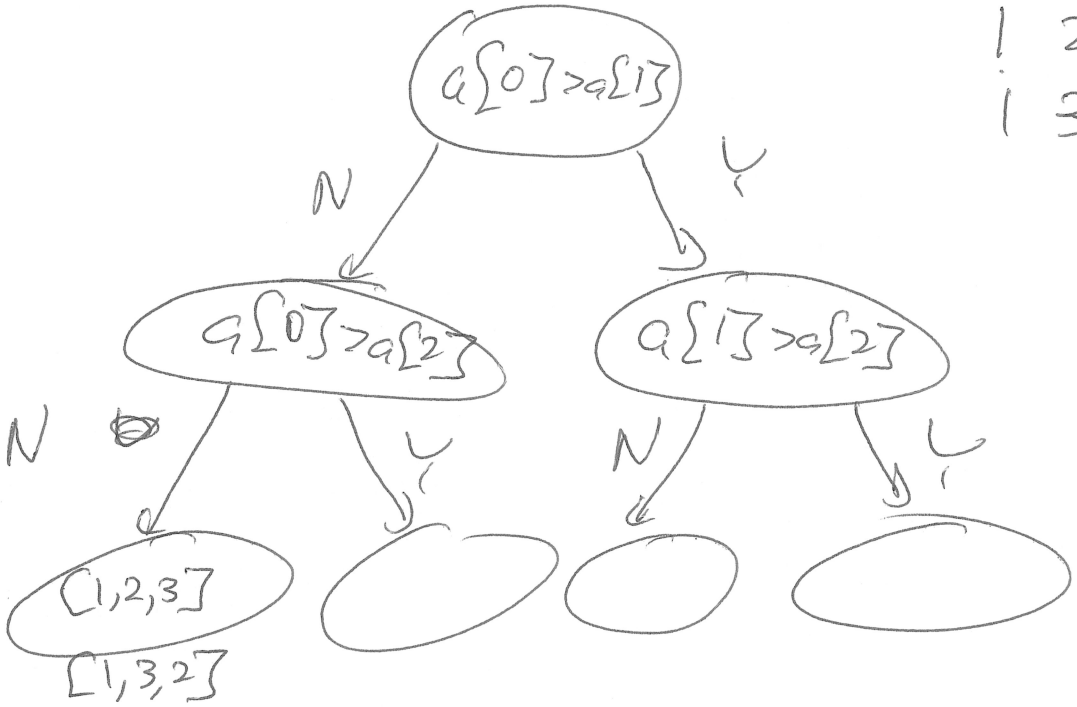
$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

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

$$B^k = A$$
$$2^1 = 2$$

$\boxed{O(n \log n)}$

Proof O(nlgn) is best we can do for comparison sorting



1 2 3
1 3 2

For any comparison sort w/ $\leq k$ comparisons it could at most sort 2^k different inputs

# of possible inputs $= n!$

$2^k \geq n!$

$\log_2 2^k \geq \log_2 n!$

$\boxed{k \geq \log_2 n!}$

$k \geq \log_2 \left(\frac{n}{e}\right)^n$

$k \geq n \log_2 \left(\frac{n}{e}\right) = n(\log_2 n - \log_2 e)$

$= O(n \lg n)$

Stirling's Approx to $n!$

$\sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$

Sorting

Any algorithm that only swaps adjacent elements, the best we can do is $O(n^2)$

An inversion in an array is 2 indexes i and j such that $i < j$ but $a[i] > a[j]$

5 2 3 1 4

5 2 1 3 4 } only removes 1 inversion!

inversions (5,2) (2,1) (5,3) (3,1) (5,1) (5,4)

Worst case # inversions

$n \quad n-1 \quad n-2, \quad \ldots \quad 1$

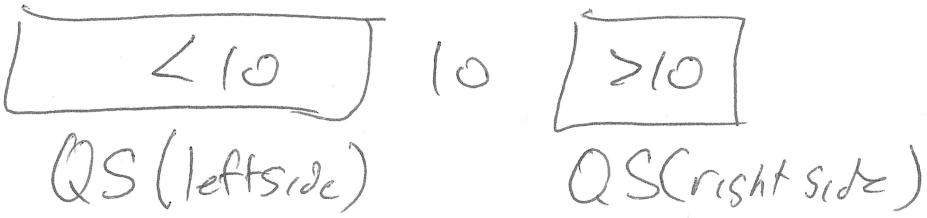
→ has $\frac{(n-1)n}{2}$ inversions

Avg # inversions $\frac{(n-1)n}{4}$ (half of all pairs are expected to be inverted!)

Merge Sort is awesome only drawback is Merge's copying into auxiliary array + copying back!

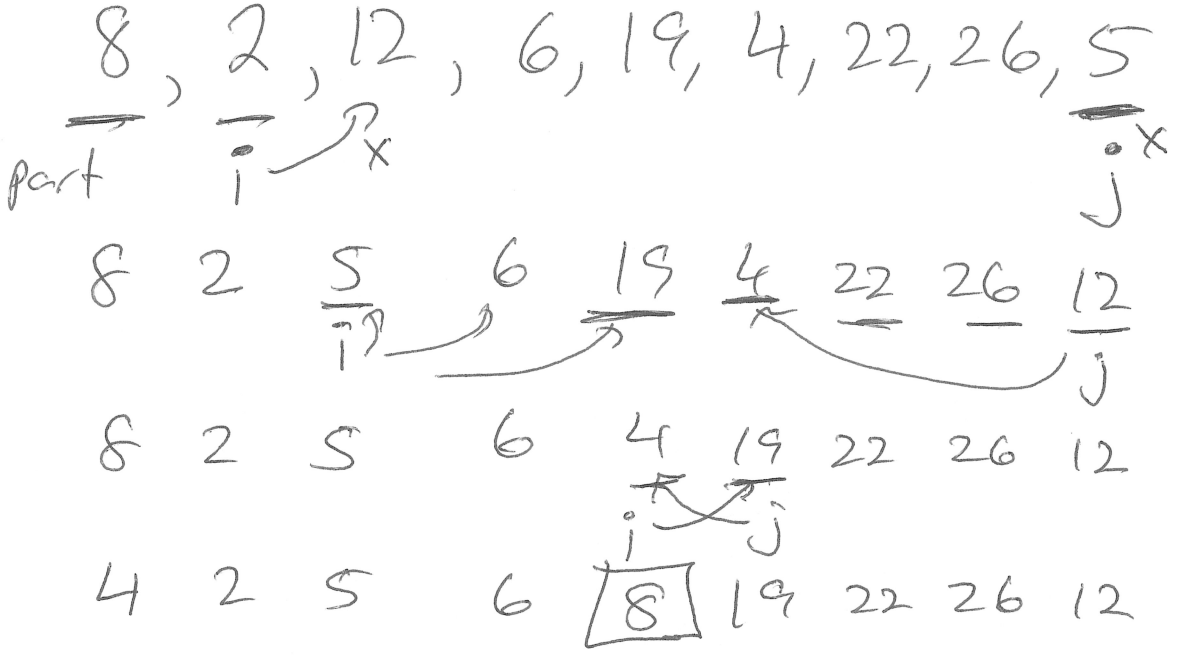
Quick Sort

(1) Partition - pick a partition element (at random), then split rest of the elements into 2 groups < part, > part




```
void qs(int* array, int lowIdx, int highIdx) {
    int mid = partition(array, lowIdx, highIdx);
    qs(array, lowIdx, mid-1);
    qs(array, mid+1, highIdx);
}
```

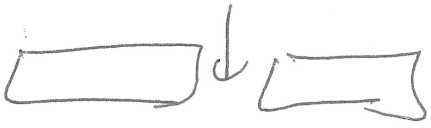
Partition Trace



Run Time

Best Case

even split

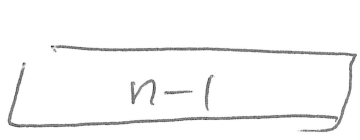


$T(n) = T(\frac{n}{2}) + T(\frac{n}{2}) + O(n)$

$\longrightarrow O(n \log n)$

Worst Case

bad split



$T(n) = \underbrace{T(n-1)}_{\text{Right Side}} + \underbrace{O(n)}_{\text{Partition}}$ $O(n^2)$

AVG CASE ???

QS avg case

Assume each split of partition is equally likely

Left	Right	prob
$n-1$	0	$1/n$
$n-2$	1	$1/n$
$n-3$	2	$1/n$
⋮		⋮
1	$n-2$	$1/n$
0	$n-1$	$1/n$

$$T(n) = \frac{1}{n}\left(T(n-1) + T(0)\right) + \frac{1}{n}\left(T(n-2) + T(1)\right) + \frac{1}{n}\left(T(n-3) + T(2)\right) + \cdots + \frac{1}{n}\left(T(0) + T(n-1)\right) + O(n)$$

$$nT(n) = 2\sum_{i=0}^{n-1} T(i) + n^2$$

$$(n+1)T(n+1) = 2\sum_{i=0}^{n} T(i) + (n+1)^2$$

$$(n+1)T(n+1) - nT(n) = 2T(n) + (2n+1)$$

$$\frac{(n+1)T(n+1)}{(n+1)(n+2)} = \frac{(n+2)T(n)}{(n+1)(n+2)} + \frac{(2n+1)}{(n+1)(n+2)}$$

$$\frac{T(n+1)}{n+2} \approx \frac{T(n)}{n+1} + \frac{2}{n+2}$$

Let $S(n) = \frac{T(n)}{n+1}$

$$S(n+1) \approx S(n) + \frac{2}{n+2}$$

$$= S(n-1) + \frac{2}{n+1} + \frac{2}{n+2}$$

$$= S(n-2) + \frac{2}{n} + \frac{2}{n+1} + \frac{2}{n+2}$$

after $k+1$ iterations

$$= S(n-k) + \frac{2}{n-k+2} + \frac{2}{n-k+3} + \cdots \frac{2}{n+2}$$

$S(0) = 0$

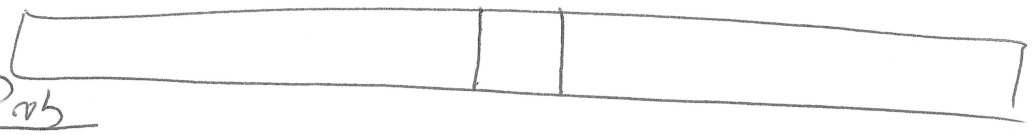
$$= S(0) + \frac{2}{2} + \frac{2}{3} + \cdots \frac{2}{n+2}$$

$$= \sum_{i=2}^{n+2} \frac{2}{i} \longrightarrow \approx H_{n+2}$$

$$S(n+1) = O(\lg n)$$

$$T(n+1) = (n+2)\,S(n+1) = \boxed{O(n \lg n)}$$

Avg Case Binary Search



$n = 2^k - 1$ elements

Prob

$\frac{1}{n} \rightarrow 1$ search

$\frac{2}{n} \rightarrow 2$ search

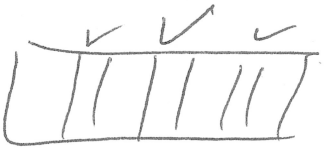
$\frac{4}{n} \rightarrow 3$ searches

$\frac{8}{n} \rightarrow 4$ searches

$\vdots$

$\frac{2^{k-1}}{2^k - 1} \rightarrow k$ searches

$k = 3$



$$S = 1 \times \frac{1}{n} + 2 \times \frac{2}{n} + 3 \times \frac{4}{n} + 4 \times \frac{8}{n} + \cdots k \cdot \frac{2^{k-1}}{2^k - 1}$$

$$S = \frac{1}{n}\left[1\times1 + 2\times2 + 3\times4 + 4\times8 + 5\times16 \cdots k \cdot 2^{k-1} \right]$$

Let $T = 1\times1 + 2\times2 + 3\times4 + \cdots k \cdot 2^{k-1}$

$-2T \quad\quad 1\times2 + 2\times4 + 3\times8 + \cdots (k-1)\cdot 2^{k-1} + k \cdot 2^k$

$$T - 2T = 1\times1 + 1\times2 + 1\times4 + 1\times8 + 2^{k-1} - k \cdot 2^k$$

$$T = k \cdot 2^k - (1 + 2 + 4 + \cdots 2^{k-1})$$

$$T = k \cdot 2^k - 2^k - 1$$

$$T = 2^k(k-1) - 1$$

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

$$n = 2^k - 1$$

$$n+1 = 2^k$$

$$k = \log_2(n+1)$$

$$S = \frac{1}{n} \cdot T = \frac{1}{n} \cdot (n+1)(k-1) - 1$$

$$= \left(1 + \frac{1}{n}\right)\left(\log_2(n+1) - 1\right) - 1$$

$$= O(\lg n)$$