2/8/2018 CS2

1. finish QSort Analysis
2. show, skip QSel Analysis
3. Sorting lower bound stuff
4. Code up Bucket Sort
   - run against quick sort or java's sort
5. Code up counting sort
6. Radix for kicks?

Avg Case Analysis QuickSort

2/8/18 (2)

array[low..high]

```
QS(array, low, high) {
    if (low >= high) return;
    int mid = partition(array, low, high);
    QS(array, low, mid-1);
    QS(array, mid+1, high);
}
```

high - low + 1

O(n)

Prob $\frac{1}{n}$ Left = 0, Right = n-1

Prob $\frac{1}{n}$ Left = 1, Right = (n-2)

Prob $\frac{1}{n}$ Left = 2, Right = n-3

$T(n) = T(0) + T(n-1) + O(n)$

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

$T(n) = T(2) + T(n-3) + O(n)$

Expectation = $\sum P_x \cdot X$ for each possible X.

2/16/2018 (1)
2/18/18 (3)

If $T(n)$ represents AVG CASE run time of Quick Sort, Then:

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

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

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

$$(n+1)T(n+1) = 2\sum_{i=0}^{n} T(i) + c(n+1)^2 \quad (\text{new eq plug in } n+1)$$

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

$$\frac{(n+1)T(n+1)}{(n+1)(n+2)} = \frac{(n+2)T(n)}{(n+1)(n+2)} + \frac{c(2n+1)}{(n+1)(n+2)} \quad (\text{trick: divide by both products})$$

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

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

2/06/2018 (16)
2/8/18 (4)

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

$2n+1$ is "close to" $2\cdot(n+1)$

replace $2n+1$ with $2n+2$ and get a slightly higher answer.

$$\sim \quad S(n+1) = S(n) + \frac{2c}{n+2}$$

$$S(n) = S(n-1) + \frac{c'}{n+1}, \quad c' \in \text{Const}$$

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

$$= S(n-3) + \frac{c'}{n-1} + \frac{c'}{n} + \frac{c'}{n+1}$$

$$= S(0) + \sum_{i=2}^{n+1} \frac{c'}{i}$$

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

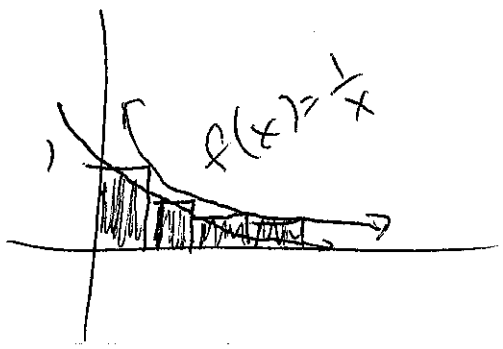
$$= \cdots S(0) + \sum_{i=1}^{n} f(i)$$

$$S(n) = 1 + \sum_{i=2}^{n+1} \frac{c'}{i} = 1 + c' \sum_{i=2}^{n+1} \frac{1}{i}$$

$$\leq c' \sum_{i=1}^{n+1} \frac{1}{i} = c' H_{n+1}$$

$$\sim c' \ln(n+1)$$

$$= O(\lg(n))$$



⇒ Let $S(n) = \frac{T(n)}{n+1}$.

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

$$T(n) = (n+1)\,O(\lg n)$$

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

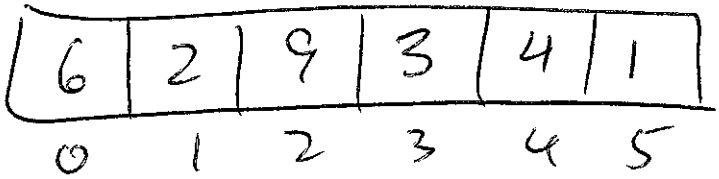
Proving algorithms can't be improved

(1) Sorts that ONLY swap adjacent elements can't run any faster in the avg case than $\Omega(n^2)$.

(2) All comparison based sorts can't run faster than $\Omega(n \lg n)$ in their worst cases.

Proof of (1)

Inversion: a pair (i, j) such that (in an array) $i < j$ and $array[i] > array[j]$



inversions $(0,1), (0,3), (0,4), (0,5)$
$(1,5)$
$(2,3), (2,4), (2,5)$
$(3,5)$
$(4,5)$

(1) Max # inversions in array of size n

(2) Avg # inversion in array of size n

(1) $n, n-1, n-2, \ldots, 1$

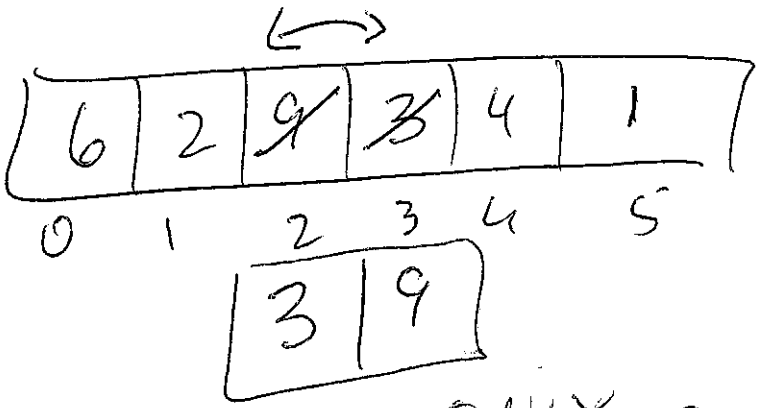
inversions = # of pairs i, j

$= \binom{n}{2} = \frac{n(n-1)}{2} = \Omega(n^2)$

(2) Avg # inversions

In ~~Array~~ all permutations, when we look at index i and index j the # of perms with a in index i and b in index j equals the # of perms with b in index i and a in index j.

$$= \frac{\text{max inversions}}{2} = \frac{n(n-1)}{4} = \Omega(n^2)$$



If an algorithm ONLY swaps adj elements, it can only remove 1 inversion per swap.

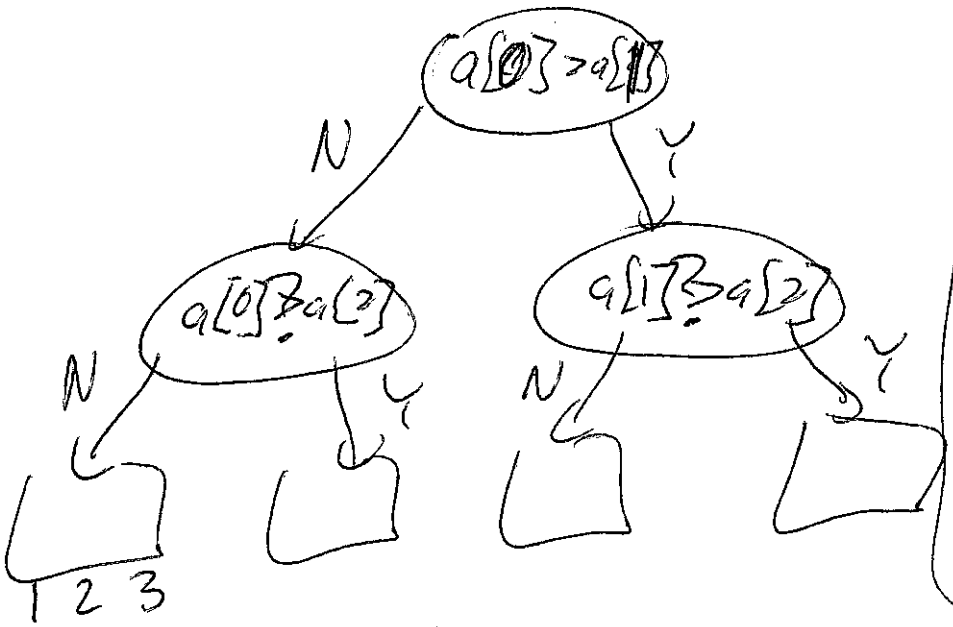
Relevant for - BUBBLE, INSERTION

Any comparison based sorting alg has a worst case performance of $\Omega(n \lg n)$

$n!$ different inputs to the sorting problem of size n.

$n = 3$

1, 2, 3
1, 3, 2
2, 1, 3
2, 3, 1
3, 2, 1



4 possible outcomes

for $n=3$ need 3 comparisons since $2^3 \geq 3!$ but $2^2 < 3!$

Worst case Runtime is such that this computation tree has at least $n!$ nodes

What is the smallest k such that

$$2^k > n!$$

$$2^k \gtrsim \left(\frac{n}{e}\right)^n \left(\text{really } \sqrt{2\pi n}\left(\frac{n}{e}\right)^n (1+\dots) \right)$$

$$\log_2 2^k \geq \log_2 \left(\frac{n}{e}\right)^n$$

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

$$k > n\left(\log_2 n - \log_2 e\right) \qquad e$$

$$k > n \log_2 n - cn$$

O(n) SORTS (NOT COMPARISON BASED)

(a) make assumptions about input

(b) restrict the input in some way

(c) expected not worst case...

BUCKET SORT

COUNTING SORT

RADIX SORT

BUCKET

(a) make assumption - DATA IS FAIRLY EVENLY DISTRIBUTED IN THE POSSIBLE RANGE OF VALUES

N items range $[L, \ldots H]$

Create N buckets.

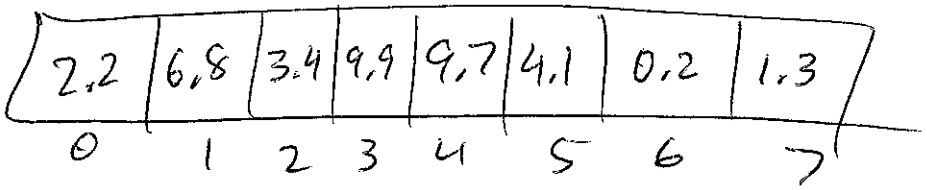
Each bucket is a linked list

Range of each bucket is $D = \frac{H-L}{N}$

bucket 0 $[L, L+D)$

bucket 1 $[L+D, L+2D)$

k $[L+kD, L+(k+1)D)$

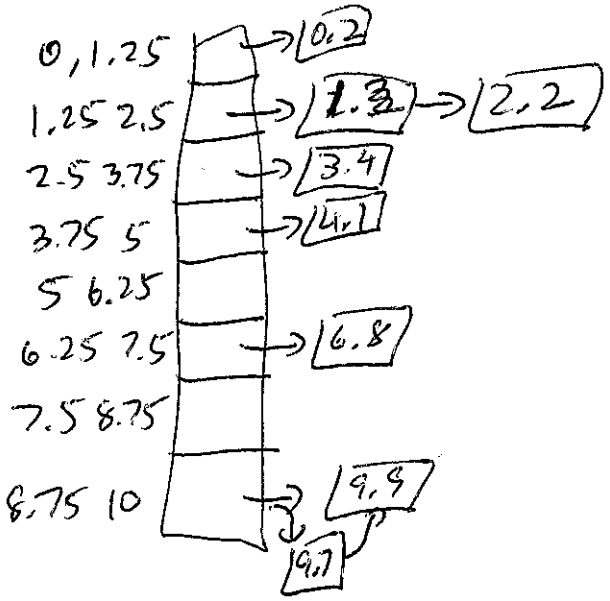


$N = 8$

$L = 0$

$H = 10$

$D = \frac{10}{8} = 1.25$



Problem

How long to insert into a bucket if randomly distributed we find the avg is O(1).

Counting Sort

All items must be integers in range from [L, H], typically [0, MAX].

MAX

[0, 6]

N = 12

3	2	0	0	3	4	6	1	6	3	5	5
0	1	2	3	4	5	6	7	8	9	10	11

Step 1

2	1	1	3	1	2	2
0	1	2	3	4	5	6

freq

$O(N + MAX)$

hope MAX ≤ N

Step 2

subtract 1 from freq[0]

1	1	1	3	1	2	2
0	1	2	3	4	5	6

Step 3

1	2	3	6	7	9	11
0	1	2	3	4	5	6

```
for(i=1; i<=MAX; i++)
    freq[i] += freq[i-1]
```

Do adjusted cumulative freq array.

3	2	0	0	3	4	6	1	6	3	5	5

0	0	1	2	3	3	3	4	5	5	6	6

loop through orig array backwards + copy values into the appropriate indexes.