CSE 3500

Hw1 due on 10.4.16 @ 3:30PM

Exam I on 10-18-16 @5PM

Let $A(n)$ be the Average Run TIME of Quick Sort on any input of size n.

$X = k_1, k_2, \ldots, k_n$.

Let $\pi_1, \pi_2, \ldots, \pi_n$ be the sorted seq.

$$X_{ij} = \begin{cases} 1 & \text{if } \pi_i \text{ and } \pi_j \text{ will be compared} \\ 0 & \text{otherwise.} \end{cases}$$

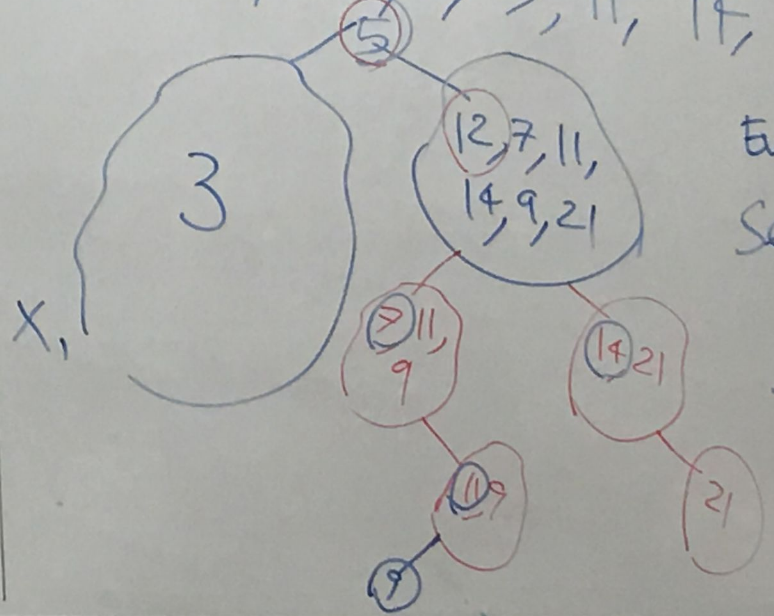
Then, $A(n) = E\left[\sum_{j=i+1}^{n} \sum_{i=1}^{n} X_{ij}\right]$

$$= \sum_{j=i+1}^{n} \sum_{i=1}^{n} E\left[X_{ij}\right].$$

$$= \sum_{j=i+1}^{n} \sum_{i=1}^{n} p_{ij}. \quad \text{———} \quad (1)$$

p_{ij} = PROB. that π_i and π_j will be compared.

$X = 5, 12, 7, 3, 11, 14, 9, 21$



Every element serves as the PIVOT at some level of RECURSION.

Box B

$\pi_1, \pi_2, \dots, \pi_{i-1}, \boxed{\pi_i, \pi_{i+1}, \dots, \pi_{j-1}, \pi_j}, \pi_{j+1}, \dots, \pi_n$

① If π_i IS PICKED AS the PIVOT BEFORE ANY OTHER ELEMENT IN BOX B, then π_i and π_j will be compared.

② If π_j is picked as the PIVOT before any other element in B, then π_i and π_j will be compared.

3) IF ONE OF the ELEMENTS $\pi_{i+1}, \pi_{i+2}, \ldots, \pi_{j-1}$ IS PICKED As the PIVOT before π_i or π_j then π_i and π_j WILL NOT BE COMPARED

As A RESULT,

$$P_{ij} = \frac{2}{(j-i+1)} \quad (2)$$

SUBSTITUTING 2 IN 1 :

$$A(n) = \sum_{j=i+1}^{n} \sum_{i=1}^{n} \frac{2}{(j-i+1)}.$$

$$= \sum_{i=1}^{n} \boxed{\sum_{j=i+1}^{n} \frac{2}{(j-i+1)}}$$

$$= \sum_{i=1}^{n} \frac{2}{2} + \frac{2}{3} + \cdots + \frac{2}{(n-i+1)}.$$

$$\leq 2 \sum_{i=1}^{n} \left(1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \right)$$

NOTE: $\sum_{i=1}^{n} \frac{1}{i} = \Theta \left(\int \frac{1}{i} di \Big]_{1}^{n} \right)$

$$A(n) \le \sum_{i=1}^{n} \Theta(\log n) = O(n \log n).$$

$\square$

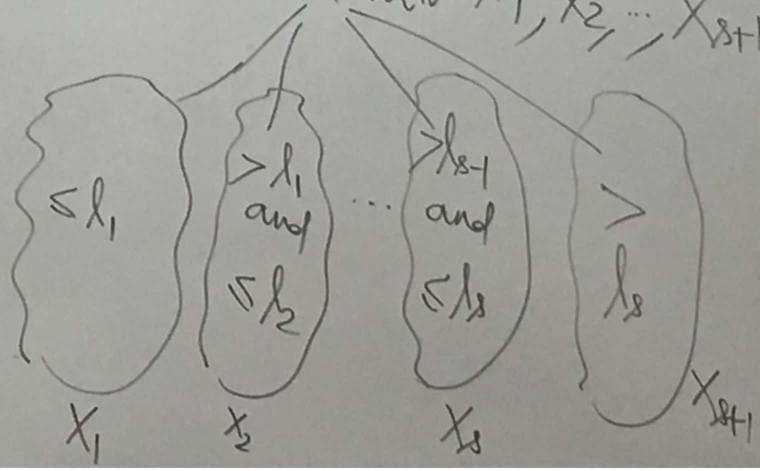
A RANDOMIZED ALGORITHM:

FRAZER & McKELLAR (1971)

(1) $X = k_1, k_2, \dots, k_n$. PICK A RANDOM SAMPLE S.

(2) SORT the sample to get $\ell_1, \ell_2, \dots, \ell_s$, $s = |S|$

(3) Partition X into $X_1, X_2, \dots, X_{s+1}$



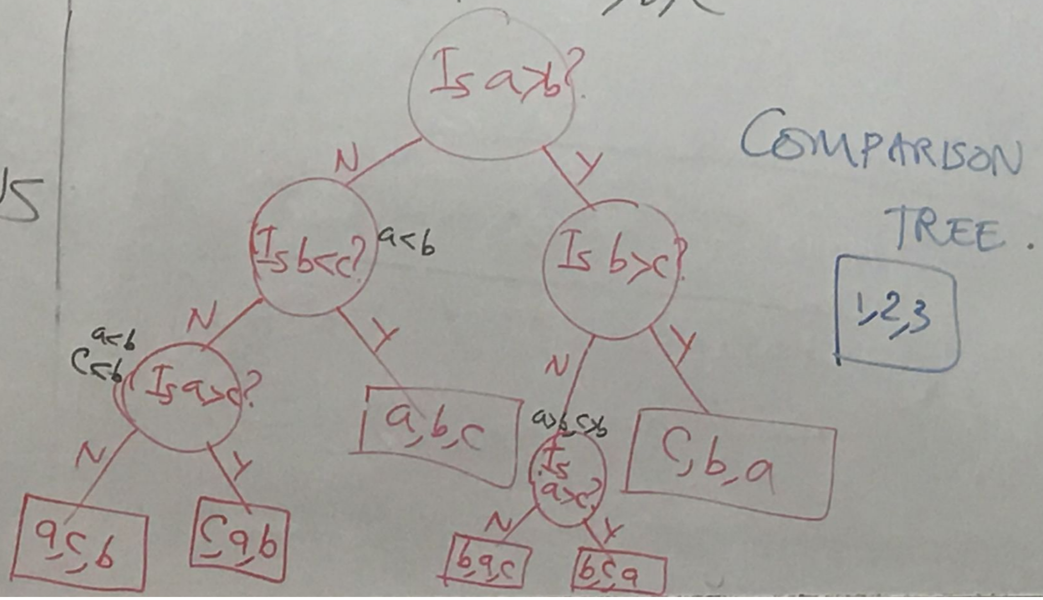
④ for $i = 1$ to $(s+1)$ do

Sort and output X_i ;

LEMMA: ANY COMPARISON BASED SORTING ALG. NEEDS $\log(n!)$ COMPARISONS ON n elements.

N5

A COMPARISON TREE TO SORT a, b, c



NOTE: (1) FOR ANY GIVEN INPUT WE TRAVERSE through only one path in the tree.

(2) IN the WORST CASE WE MAKE h Comparisons, h being the height of the tree.

(3) for n elements there are n! POSSIBLE ANSWERS.

4) There has to be at least one LEAF corresponding to each possible answer.

$\Rightarrow$ There has to be $\geq n!$ LEAVES.

$\Rightarrow$ The height of the tree has to be $\geq \log(n!)$.

$\Rightarrow$ IN the WORST CASE we need at least $\log(n!)$ COMPARISONS to sort n elements.

STIRLING'S APPROXIMATION:

$$n! \approx \left(\frac{n}{e}\right)^n \sqrt{2\pi n}\left(1+\frac{1}{\theta(1)}\right).$$

$$\log(n!) = \theta(n \log n).$$

BUCKET SORT:

INPUT: $X = k_1, k_2, \dots, k_n$, EACH k_i IS AN INTEGER $\in [1:m]$ $1 \le i \le n$.

Output: Sorted X.

① CREATE AN ARRAY OF LISTS: $A[1:m]$, ONE LIST FOR EACH POSSIBLE VALUE.

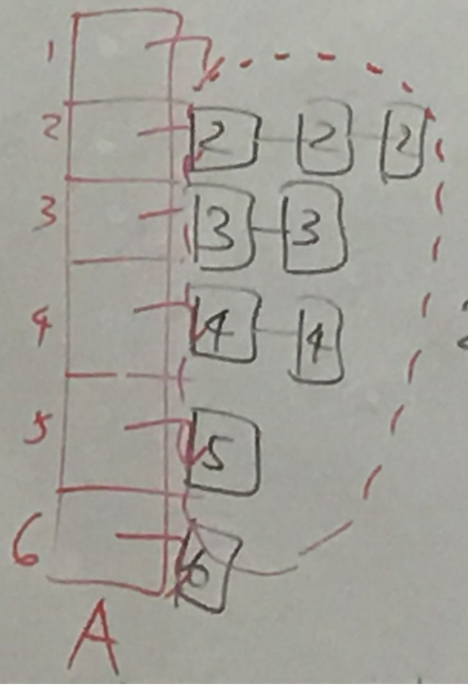
② **for** $1 \le i \le n$ **do**

INSERT k_i into LIST $A[k_i]$ at the TAIL;

③ **for** $1 \le i \le m$ **do** output the elements in $A[i]$;

Example:

$X = \cancel{3}, \cancel{2}, \cancel{3}, \cancel{5}, \cancel{4}, \cancel{2}, \cancel{6}, \cancel{4}, 2$



2, 2, 2, 3, 3, 4, 4, 5, 6

RUN TIME

$= m + n + (m+n)$

$= \Theta(m+n).$

CSE 3500 Algorithms and Complexity – Fall 2016
Lecture 10: September 29, 2016

Quick sort: Average Run Time

- In the last lecture we started analyzing the expected run time of quick sort. Let $X = k_1, k_2, \ldots, k_n$ be the input sequence and let $\pi_1, \pi_2, \ldots, \pi_n$ be the sorted order of X. We obtained the following equation for the expected run time $A(n)$ of quick sort on an input of n elements (where p_{ij} is the probability that the algorithm will compare π_i with π_j):

$$A(n) = \sum_{j=i+1}^{n} \sum_{i=1}^{n} p_{ij}. \tag{2}$$

- Note that in the quick sort algorithm each input key serves as the pivot element at some level of recursion.

- From out of the elements $\pi_i, \pi_{i+1}, \ldots, \pi_{j-1}, \pi_j$, if either π_i or π_j serves as the partitioning element before any of the elements $\pi_{i+1}, \pi_{i+2}, \ldots, \pi_{j-2}, \pi_{j-1}$, then π_i and π_j will be compared in the algorithm; On the other hand if any of the elements $\pi_{i+1}, \pi_{i+2}, \ldots, \pi_{j-2}, \pi_{j-1}$ is picked as the pivot before either π_i or π_j, then these two elements will not be compared.

- As a result, we infer that $p_{ij} = \frac{2}{j-i+1}$. Substituting this in equation 2, we get:

$$A(n) = \sum_{j=i+1}^{n} \sum_{i=1}^{n} \frac{2}{j-i+1} = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{2}{j-i+1} = \sum_{i=1}^{n} \left[\frac{2}{2} + \frac{2}{3} + \cdots + \frac{2}{n-i+1}\right]$$

$$\leq \sum_{i=1}^{n} 2\left[1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}\right].$$

- Recall that we can aproximate sums with integrals. Using this technique, we see that: $\sum_{i=1}^{n} \frac{1}{i} = \Theta\left(\int_1^n \frac{1}{i} di\right) = \Theta(\log n)$.

- Therefore, it follows that $A(n) \leq \sum_{i=1}^{n} \Theta(\log n)$. In other words, $A(n) = O(n \log n)$. The following Lemma follows.

  Lemma. *The expected run time of quick sort on n elements is $O(n \log n)$.* $\square$

Randomized Sorting Algorithms

- Quick sort can be converted to a randomized algorithm by picking the pivot element randomly. In this case we can show that the expected run time is $O(n \log n)$ (where the expectation is computed in the space of all possible outcomes for coin flips).
- We can modify quick sort as follows. Pick a random sample of s elements (for some relevant value of s), find the median of this sample, and use this median as the pivot element. For example, s could be $5, 11, 15$, etc. This algorithm will perform better than quick sort in practice.
- In 1971 Frazer and McKellar came up with the following algorithm:

  1) Pick a random sample of s elements from$X$;
  2) Sort the sample and let $l_1, l_2, \ldots, l_s$ be the sorted sample;
  3) Partition X into $s+1$ parts as follows. $X_1 = \{q \in X : q \leq l_1\}$;
     $X_i = \{q \in X : l_{i-1} < q \leq l_i\}$, for $i = 2, 3, \ldots, s$, and $X_{s+1} = \{q \in X : q > l_s\}$;
  4) **for** $i = 1$ **to** $s+1$ **do**
     Sort and output X_i.

- The above algorithm is one of the best known algorithms for sorting. This algorithm has been implemented over a variety of computing models and architectures. The number of comparisons made by this algorithm is very close to the lower bound of $\log n!$.

A Lowerbound for Sorting

- We will prove the lower bound on the comparison tree model. This model accounts for only the comparisons made in the algorithm.
- A comparison tree is a binary tree. In a comparison tree comparison between a pair of elements is done at every node. Based on the outcome of this comparison we move down to an appropriate child of this node. The leaves of the tree correspond to answer nodes.
- A comparison tree is constructed for every input size.
- For a given instance of the problem, we start at the root of the tree, perform the comparison dictated by the root. Based on the outcome of this comparsion, we move to a relevant child, perform the comparison dictated by this node, and so on. This is continued until we reach a leaf. The leaf we reach will have the correct answer.
- We make the following observations: 1) For any given instance of sorting, we traverse through only one path in the tree; 2) Thus the worst case run time is the height of the

tree; 3) There has to be at least one leaf corresponding to every possible answer; and 4) Since there are $n!$ possible permutations for any sequence of n elements, a comparison tree that sorts n elements should have at least $n!$ leaves.

- We know that the height of a binary tree with N leaves is at least $\log N$.
- Therefore, the height of a comparison tree that sorts n elements will be at least $\log n!$.
- In turn, it follows that the number of comparisons needed to sort n elements, in the worst case, is at least $\log n!$.
- We can show that $\log n! = \Theta(n \log n)$ (for example using Stirling's approximation for $n!$).

Bucket Sorting

- We might be able to sort n elements in time better than $\log n!$ if we have additional information about the elements (than just knowing that they come from a linear order).
- For example consider a sequence X of n elements where each element is either zero or one. We can sort X in linear time as follows: We add the bits in X. Let q be this integer. Note that q is the number of ones in X. We output $n - q$ zeros followed by q ones.
- We can extend the above idea to derive the bucket sort algorithm.
- Let $X = k_1, k_2, \ldots, k_n$ be the input sequence where each k_i is an integer in the range $[1, m]$ (for some integer m). We can sort X by keeping a bucket for each possible value. We do one pass through the input and place each element in the right bucket based on its value. Followed by this we output the buckets in order.
- A pseudocode for tha above algorithm is:

  1) Create an array $A[1 : m]$ of m empty lists;
  2) **for** $i = 1$ **to** n **do**
     Insert k_i to the tail of the list $A[k_i]$;
  3) **for** $i = 1$ **to** m **do**
     Output the elements in the list $A[i]$ starting from the head;

- In the above algorithm we spend $O(m)$ time for step 1, $O(n)$ time for step 2, and $O(m+n)$ time for step 3 (since there are m lists and there are a total of n elements in all the lists together). Thus, the run time of this bucket sort algorithm is $O(m+n)$.
- Bucket sort will take linear time if $m = O(n)$.

- The run time of bucket sort will be better than that of any comparison sorting algorithm (such as heap sort) as long as $m = o(n \log n)$.
- In the next lecture we will see that we can sort n integers in $O(n)$ time if they come from the range $[0, n^c]$, c being any constant.