BASIC STEP. Ropeat Flip an n-sided com to get i; Flip an n-sided Coin to get j; LEPEATED ELEMENT IDENTIFICATION . if it j and ki = kj then Sutput ki and QUIT; NIVT: X=K, K2 ... Ky There are of copies of The other dements are distinct. Follower OUTPUT: The Repeated element

ANALYSIS. ROB. OF SUCCESS IN SME BASIC STEP $=\frac{N}{2}\left(\frac{N}{2}-1\right)$ N 21 UN 210

PROB. OF Failure in one basic Step is $\xi \xi$. Prob. SF Failure in the FIRST K (4) K Basic Steps is $\xi(\xi)$.

We want this prob. to be $\leq n^{\alpha}$. This happens when $(\frac{4}{5})^{k} \leq n^{\alpha}$. $k \log(\frac{4}{5}) \leq -\alpha \log n$ $\Rightarrow k \geq \frac{\alpha \log n}{\log(\frac{4}{5})}$.

NOTE: REN TIME OF Den. We say that the the Repeated Element identification RUN TIME of a alg. is E(logn). D (as Vegas) Algorithmy is O(f(n)) if the RUN TIME is Note: h= 29 h 5 (Rfin) +n≥no with a prob. of 2(1-n') (and h, BEWG CONSTANTS.

Example: DETERMINISTIC ALG. NOTE: $NPUT: X = k_1, k_2 \cdots k_n$ try Det. alg. () SORTING - O(mlog h) OUTRT: twy element of has to spend (2) Output the largest B the FIRST (2+1) Devents.=> O(n) TIME. at least in steps X that is > The MEDIAN in the WORST OF X. CASE.

A MONTE CARLO ALG.

Pick k elements at nandom; Five and output the MAX OF THIS Sample. ANALYSIS. PROB. that a Random demont is incorrect is \$\frac{1}{2}. The output is incorrect iff all the k elements are < the MODIAN. Prob. of this happening is \$\frac{1}{2}\t

Some FACTS. We want this O Pouters: $\chi^{e} = 1; (\chi^{a})^{b} = (\chi^{b})^{a} = \chi^{ab};$ prob. to be sind. This will happen if χ^q . $\chi^b = \chi^{a+b}$. (1) k s n Ren -k s-xlog n - Olgn Ologanithms: If ab = c then $\int dg_a C = b$. $\int dg_a C = C$ > k > x logn

*
$$\log_{a}(ab) = \log_{a} a + \log_{b} b$$

* $\log_{a}(ab) = \log_{a} a + \log_{b} b$
* $\log_{a}(ab) = b\log_{a} a$
* $\log_{a}(ab) = b\log_{a} a$
* $\log_{b}(ab) = b\log_{b} a$

X is the real. (F) [x] = the largest integer ex. [6]=6; [3.4]=3. X7 = the Smallest integer > x. ex. [12]=12; [5.6]=6.

(5) CHEDMETRIC SERIES. * a + ar + ar + ar + ar = a(r + 1)(r-1) * [+2+2+...+2] = 2-1* If |r|<1, then, $a + ar + ar^{2} = a$ $\overline{(1-r)}$ * If |x| <1 then 1+x+x+... = 1-x).



Area of the rectangle = f(a). $\sum_{i=a}^{b} f(i) = \Theta\left(\int_{a} f(i) di\right).$ Area under the Curve from a to (a+1) is at find is f(a). a a $\int f(i) di \leq f(a)$. $\int f(i) di \geq f(a)$.

DATA STRUCTURES,

A Data Structure is a data storage that SUPPORTS Some OPERATURNS. Example: A DICTIONARY SUPPORTS: 1) SEARCH(X): X IS ABBITRARY 2) INSERT (X): X IS ARBITRARY (3) DELETE(X): X 15 ARBITRARY. APPLICATIONS; LIBRARY, EMPLOYEE Data basp.

MIN PRIORITY QUEUE SUPPORTS: () FIND-MIN: Retwon the Smallest element 2) INSERT (X): X IS ARBITRARY 3 Decte-MIN: delete the MIN element.

SORTING. INPUT: X=K, K2..., Kn let Q be a priority queue that is empty for Isisn do INSERT (Fi); for Isish do Sutput Delete-MIN;

CSE 3500 Algorithms and Complexity – Fall 2016 Lecture 3: September 6, 2016

Identification of the Repeated Element: A Las Vegas algorithm

- This problem is defined as follows. Input: $X = k_1, k_2, \ldots, k_n$. It is known that X has $\frac{n}{2}$ copies of one element and the other elements are distinct (i.e., they occur exactly once each). Output: the repeated element.
- In the last lecture we discussed several deterministic algorithms for solving this problem. The best of these algorithms takes O(n) time. We also argued that any deterministic algorithm for solving this problem needs $\frac{n}{2} + 2$ time.
- We discussed the following Las Vegas algorithm for slving the repeated element identification problem:

```
repeat

Flip an n-sided coin to get i;

Flip an n-sided coin to get j;

if i = j and k_i = k_j then output k_i and quit;

forever
```

- **Definition:** The run time of a Las Vegas algorithm is $\tilde{O}(f(n))$ if the run time is no more than $c\alpha f(n)$ for all $n \ge n_0$, with a probability of $\ge (1 n^{-\alpha})$, where c and n_0 are some constants.
- Claim: The run time of the above algorithm is $\widetilde{O}(\log n)$.

Proof: Call the sequence of three statements within the **repeat** loop as a *basic step*. Probability that we get success in one run of the basic step is $\frac{\binom{n}{2}\binom{n}{2}-1}{n^2}$. This probability is at least $\frac{1}{5}$ for all $n \ge 10$. Thus, the probability of failure in one basic step is $\le \frac{4}{5}$. This means that the probability that the first k basic steps are all unsuccessful is $\le \left(\frac{4}{5}\right)^k$. We want this probability to be very small, specifically no more than $n^{-\alpha}$. Equating the two, we get the following value for k:

$$k = \frac{\alpha \log n}{\log(5/4)}.$$

We have shown that the probability that the above algorithm takes more than $\frac{\alpha \log n}{\log(5/4)}$ basic steps is $\leq n^{-\alpha}$. Note that each basic step takes a constant time. Therefore, the run time of the algorithm is $\leq c\alpha \log n$ with a probability of $\geq (1 - n^{-\alpha})$, for some constant *c*. Thus it follows that the run time of the algorithm is $\tilde{O}(\log n)$. \Box

Finding an Element as Large as the Median

- Consider the following problem. Input: $X = k_1, k_2, \ldots, k_n$; Output: An element of X that is at least as large as the median of X.
- For this problem also, several deterministic algorithms can be developed: 1) Sort X and pick an element at the middle or to its right. This will take $\Theta(n \log n)$ time; 2) Find and output the maximum of X. This takes $\Theta(n)$ time; 3) Find and output the maximum of $k_1, k_2, \ldots, k_{n/2}$. This also takes $\Theta(n)$ time.
- One could argue that any deterministic algorithm will need $\Omega(n)$ time to solve this problem as follows: If an adversary chooses the input and if this adversary has perfect knowledge about the algorithm, (s)he can force the algorithm to look at $\geq \frac{n}{2}$ elements of X. If the algorithm is ready to give an answer looking at less than n/2 elements, the adversary can choose the other elements of X in such a manner to make the output of the algorithm incorrect.
- We can devise a Monte Carlo algorithm that takes only $O(\log n)$ time and whose output is correct with a high probability as follows.

Pick a random sample S of $\alpha \log n$ elements from X; Find and output the largest element of S;

The above algorithm will give an incorrect answer only if all the elements in S are less than the median. Probability that a randomly picked element of X is less than the median is $\leq \frac{1}{2}$. Therefore, probability that all the elements in S are less than the median of X is $\leq \left(\frac{1}{2}\right)^{\alpha \log n} = n^{-\alpha}$. In other words, the output of this algorithm is correct with a high probability. Clearly, the run time of the algorithm is $O(\log n)$.

Some Preliminaries

- Exponentials: If x, a, b are real numbers (with $x \neq 0$) then the following are true: $x^0 = 1; (x^a)^b = (x^b)^a = x^{ab}; x^a x^b = x^{a+b};$
- Logarithms: If a, b, c are real numbers > 0 and n is positive integer than the following statements are true: $a = b^{\log_b a}$; $\log_c(ab) = \log_c a + \log_c b$; $\log_b a = \frac{\log_c a}{\log_c b}$; $\log_a(1/b) = -\log_a b$; $\log_b a^n = n \log_b a$; $a^{\log_b n} = n^{\log_b a}$;

- Stirling's Approximation: Let *n* be a positive integer. Then, $n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right);$
- For any non-negative real number x, $\lfloor x \rfloor$ is defined to be the largest integer less than or equal to x. For instance, $\lfloor 5 \rfloor = 5$ and $\lfloor 3.2 \rfloor = 3$. Also, $\lceil x \rceil$ is defined to be the smallest integer greater than or equal to x. For example, $\lceil 7.3 \rceil = 8$ and $\lceil 12 \rceil = 12$.
- Geometric Series. If a and $r \neq 1$ are real numbers and n is any positive integer, then the following statement are true:

$$a + ar + ar^{2} + \dots + ar^{n} = a\left(\frac{r^{n+1}-1}{r-1}\right).$$

As a corollary, it follows that:

$$1 + 2 + 2^2 + \dots + 2^n = 2^{n+1} - 1.$$

• When |r| < 1,

$$a + ar + ar^2 + \dots = \frac{a}{1 - r}$$

- Summations: $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$; $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$; $\sum_{i=1}^{n} i^3 = \left[\frac{n(n+1)}{2}\right]^2$.
- Approximating sums with integrals: We can approximate sums of the form $\sum_{i=a}^{b} f(i)$, where f(i) is an integral function of i, with integrals. Consider the case when the function f(i) is monotonically increasing with i. Note that $f(i) \leq \int_{a}^{a+1} f(i) di$, $f(a + 1) \leq \int_{a+1}^{a+2} f(i) di$, and so on. Adding up these piecewise approximations we realize that $\sum_{i=a}^{b} f(i) \leq \int_{a}^{b+1} f(i) di$. Also, we see that $f(a) \geq \int_{a-1}^{a} f(i) di$, $f(a + 1) \geq \int_{a}^{a+1} f(i) di$, etc. Adding together, we note that $\sum_{i=a}^{b} f(i) \geq \int_{a-1}^{b} f(i) di$. Put together we get:

$$\int_{a}^{b+1} f(i) \, di \ge \sum_{i=a}^{b} f(i) \ge \int_{a-1}^{b} f(i) \, di.$$

• As an example consider $\sum_{i=1}^{n} i^2$. Using the above fact we observe:

$$\int_{1}^{n+1} i^2 di \ge \sum_{i=1}^{n} i^2 \ge \int_{0}^{n} i^2 di.$$

In other words, $\frac{(n+1)^3-1}{3} \ge \sum_{i=1}^n i^2 \ge \frac{n^3}{3}$. I.e., $\sum_{i=1}^n i^2 = \Theta(n^3)$.

Data Structures

• A data structure can be thought of as a black box where data can be stored. The black box supports a set of operations. Depending on the type of data and the set of operations supported, we have different data structures.

- Data structures play vital roles in data analysis as well as algorithms design.
- A *dictionary* supports the following three operations:
 - 1. Search(x): search for the element x in the storage, where x is an arbitrary element;
 - 2. Insert(x): insert the element x, where x is an arbitrary element;
 - 3. Delete(x): delete the element x, x being an arbitrary element.
- Example applications for a dictionary: a library system, an employee database, etc.
- A *min priority queue* supports the following operations:
 - 1. Find_Min(): identify the smallest element in the storage;
 - 2. Insert(x): insert the element x, where x is an arbitrary element;
 - 3. Delete_Min(): delete the smallest element.
- A max priority queue can be defined in a similar manner.
- An example application for a priority queue: sorting. We can sort n given elements using a priority queue as follows. Let $X = k_1, k_2, \ldots, k_n$ be the input.

```
Start with an empty priority queue Q;

for i = 1 to n do

Insert k_i into Q;

for i = 1 to n do

Output Delete_Min();
```

- The above algorithm is called heap sort and it is one of the asymptotically optimal algorithms known for sorting.
- A dictionary and a priority queue can be imlemented using arrays and linked lists. In this case each operation takes O(n) time. In each implementation, at least one of the three operations under concern will take $\Omega(n)$ time. We could achieve a better performance using trees.