Applied Algorithms

CSCI-B505 / INFO-I500

Lecture 19.

Randomized Algorithms - I

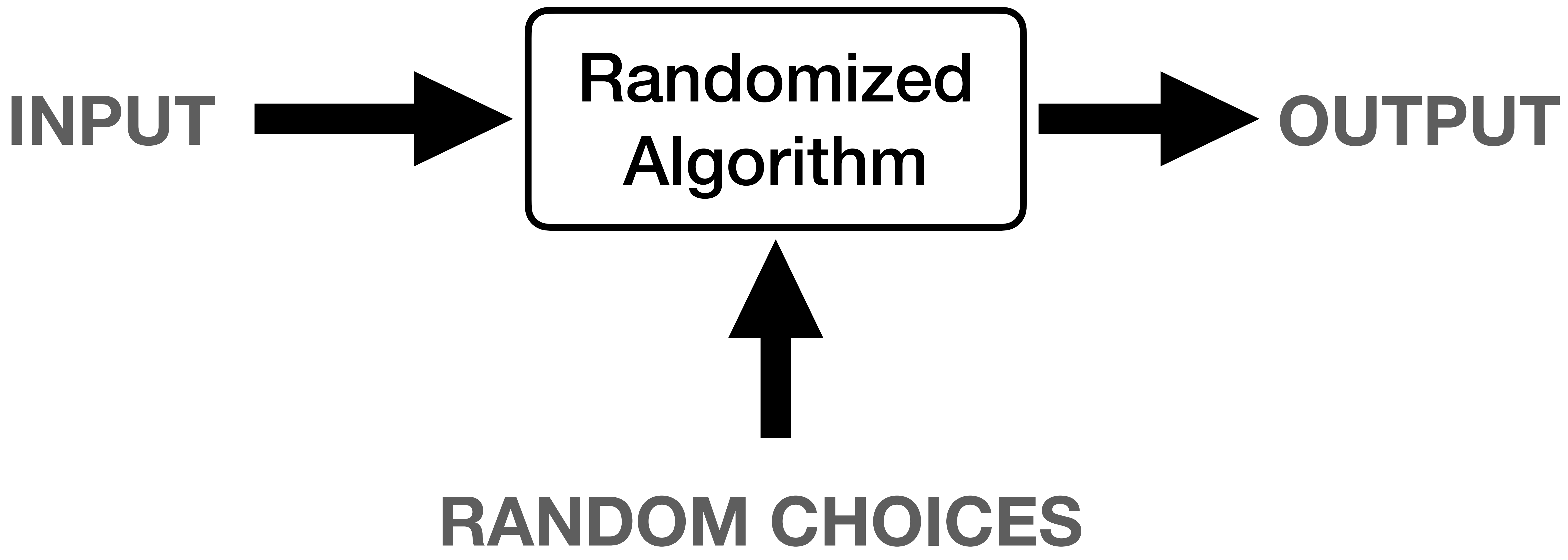
M. Oguzhan Kulekci

- Deterministic vs. Randomized algorithms
- Probabilistic analysis versus average case analysis
- Randomized quick sort
- Randomized primality testing
- Types of randomized algorithms, Las Vegas vs Monte Carlo
- Approximate median
- Majority tree evaluation

Deterministic vs Randomized Algorithms

- Assume we run the algorithm several times with the same input.
  - Is it possible to end up with a different output? OR
  - Is it possible to have different number of steps (time) ?
    - **NO !** Means it is a deterministic algorithm !

Deterministic vs Randomized Algorithms



- Assume we run the algorithm several times with the same input.
  - Is it possible to end up with a different output? OR
  - Is it possible to have different number of steps (time) ?
    - **YES !** Means it is a RANDOMIZED algorithm !

Randomized Algorithms

- Usually very short and simple to implement
- A good way of attacking a hard problem
- Remember our previous heuristics on graph coloring or bin packing
- It may not be that much easy to prove that it works well
    - **Probabilistic analysis**
    - *Notice probabilistic analysis does not make any assumption on the input, where average-case analysis makes it*

Randomized Quick Sort

- Variable Execution Time

- Original quick sort is deterministic!
- Given the input, the selection of pivots and thus everything is exactly the same.
- The **average** number of comparisons by **deterministic** QS is $O(n \log n)$

- Randomized QS selects the pivot elements randomly.
- At every execution, different selection of the pivot elements result in different number of comparisons, and hence, different running times.
- The **expected** number of comparisons by **randomized** QS is $O(n \log n)$

Randomized Quick Sort

- Variable Execution Time

The expected number of comparisons done by randomized QS is at most $2n \ln n$

- Assume X_{ij} is the probability of comparing the ith smallest element with jth.
- Then, total number of comparisons is $C = \Sigma_{i=1}^{n} \Sigma_{j=i+1}^{n} X_{ij}$.
- $E[C] = \Sigma_{i=1}^{n} \Sigma_{j=i+1}^{n} E[X_{ij}]$
- $E[X_{ij}] = 2/(j - i + 1)$ **There will be a pivot selected between the ith and jth smallest elements, and only when this is equal to one of them, comparison happens.**

$$E[C] = \Sigma_{i=1}^{n} \Sigma_{j=i+1}^{n} \frac{2}{j - i + 1} = \Sigma_{i=1}^{n} 2(\frac{1}{2} + \frac{1}{3} + .. + \frac{1}{n - i + 1}) \leq 2n \ln n$$

- *See Cormen for more details…*

The number of comparisons, and thus, the execution time, depends on the random selection of the pivots !

Primality Testing

- Variable Output

- How can we check whether a given integer n is prime or not ?
- Try to divide n by all integers 2 to $\lceil \sqrt{n} \rceil$
- It becomes a difficult test, when n is large, e.g. n is a 1024 bit number ?

- Fermat's little theorem:
    - If n is prime, then $a^{n-1} = 1 \mod n$ for all a not divisible by n
- For example,
    - $n = 17, a = 3, 3^{16} = 1 \mod 17$
    - $n = 16, a = 3, 3^{15} = 11 \mod 16$

Primality Testing

- Variable Output

- By using the Fermat's little theorem, choose a random value for a, and check it.
- There might be a values, where $a^{n-1} = 1 \mod n$, but n is not prime.
  - For example, $n = 9$, $a = 8 \rightarrow 8^{9-1} = (8^2)^4 = 1 \mod 9$
- Then, how about repeating this many times.

- Pick randomly k values that are not dividing n.
- Check with the Fermat's little theorem, and report prime if all are satisfied.
- If the probability of a composite giving a residue of 1 is, say p, then this will report a composite as a prime with probability p^k, which is very low, but not 0!
- More, there are numbers (Charmichael numbers, 1 over 50 billion) that satisfies Fermat, but they are composite!

Types of Randomized Algorithms

Las Vegas algorithms:

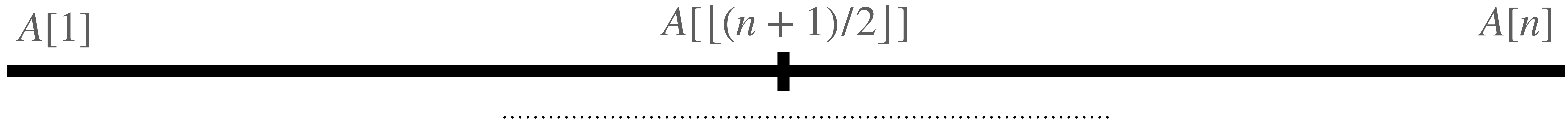
Always produce the **correct** answer, but the running time varies according to the random choice, e.g., randomized quick sort.

Monte Carlo algorithms:

The **running time** is **upper-bounded**, but there might be an error on the output according to the random choice, e.g., primality testing.

In a randomized algorithm either the **time** or the **output** changes according to some random choices made in the algorithm.

Approximate Median



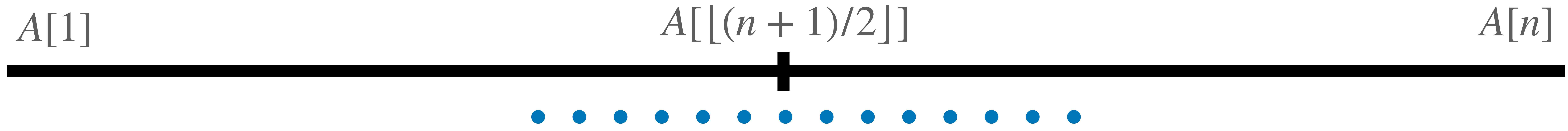
- Given n unsorted integers, what is the **median** value?
- Median is the number with the **rank** $\lfloor (n+1)/2 \rfloor$ on the sorted sequence.
- We can sort all numbers in $O(n \log n)$ and then return the median.

- Most of the time, the result is acceptable if the **rank** k of the returned value is between

$$\left\lfloor (\frac{1}{2} - \gamma)(n+1) \right\rfloor \leq k \leq \left\lceil (\frac{1}{2} + \gamma)(n+1) \right\rceil, \text{ for } 0 \leq \gamma \leq 1/2$$

- We can improve the time complexity via randomization.

Approximate Median



Algorithm $\mathit{ApproxMedian1}(\delta, A)$

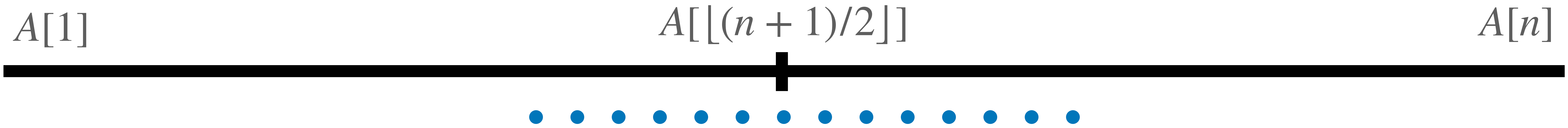
$\triangleright$ $A[1..n]$ is array of n distinct numbers.
$r \leftarrow \mathit{Random}(1, n)$
$x^* \leftarrow A[r]$; $k \leftarrow 1$
for $i \leftarrow 1$ **to** n
    do if $A[i] < x^*$ **then** $k \leftarrow k + 1$
if $\lfloor(\frac{1}{2} - \delta)(n+1)\rfloor \leqslant k \leqslant \lceil(\frac{1}{2} + \delta)(n+1)\rceil$
    then return x^*
    else return "error"

- Runs in $O(n)$ time
- Returns a valid answer if the randomly selected item is δ-approximate.
- What is the probability of success?

$$\frac{\lceil(\frac{1}{2}+\delta)(n+1)\rceil - \lfloor(\frac{1}{2}-\delta)(n+1)\rfloor + 1}{n} \approx 2\delta.$$

- Depends on δ
- We can repeat it, until we reach success

Approximate Median

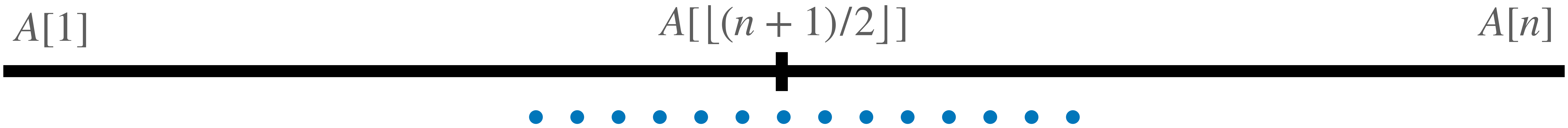


Algorithm *ApproxMedian2*(δ, A)

1. $j \leftarrow 1$
2. **repeat** *result* $\leftarrow$ *ApproxMedian1*(A, δ); $j \leftarrow j + 1$
3. **until** (*result* $\neq$ "error") **or** $(j = c + 1)$
4. **return** *result*

- Runs in $O(c \cdot n)$ time
- The probability that it cannot find a valid answer in c iterations is $(1 - 2\delta)^c$
- Running time is upper-bounded but the output is not guaranteed. **(Monte Carlo)**
- The success is not related with what the input data is.
- What if we want to reach a valid answer ?

Approximate Median



Algorithm *ApproxMedian3*(δ, A)

1. **repeat** *result* $\leftarrow$ *ApproxMedian1*(A, δ)
2. **until** *result* $\neq$ "error"
3. **return** *result*

- Sooner or later it will give us the correct result (**Las Vegas type**)
- Running time is now a random variable, say T
- What is $E[T]$ = ?

$$E[T] = E[\text{number of calls} x O(n)] = O(n) \cdot 1/2\delta = O(n/\delta)$$

Types of Randomized Algorithms

Any Las Vegas algorithm whose expected running time is $E[T(n)] = f(n)$, can be converted into a Monte Carlo algorithm that runs in $k \cdot f(n)$ time with an expected success ratio of $(k-1)/k$.

- Run the LV algorithm in exactly $k \cdot f(n)$ time.
- If it finishes, then we already have the correct result, success !
- Unsuccessful, if it does not !
- What is the probability that it does not finish in $k \cdot f(n)$ time?
- Due to Markov inequality $Pr(X \geq a) \leq \dfrac{E[X]}{a}$, $Pr(T(n) \geq k \cdot f(n)) \leq \dfrac{f(n)}{k \cdot f(n)} = \dfrac{1}{k}$
- It will be successful with probability $1 - \dfrac{1}{k} = \dfrac{k-1}{k}$.

Reading assignment

- Cormen chapter 5, Kleinberg&Tardos chapter 13
- Search internet for many discussions on the visited problems