PARALLEL AND SEQUENTIAL ALGORITHMS AND DATA STRUCTURES

LECTURE 11
Randomized Algorithms

華中科技大学
HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Randomized Algorithms

- Exploit randomness during computation
  - Pivot selection in Quicksort
  - Average case analysis
  - Primality testing
- Question: How many comparisons are needed to find the *second* largest number on a sequence of n numbers?
  - Naive algorithm: $2n-3$ comparisons why?
  - Divide and Conquer algorithm: $3n/2$ comparisons why?
  - Simple randomized algorithm: $n-1+2\log n$ comparisons *on the average* why?

SYNOPSIS

- Expectation versus High Probability
- Finding The Two Largest
- Find the k^{th} smallest element
- Quicksort
- Analysis of Quicksort

Expected vs. High Probability Bounds.

- **Expected bounds**
  - the average case across all random choices used in the algorithm
  - Once in a while, the work could be much larger
- **High-probability bounds**
  - it is very unlikely that the cost will be above some bound
  - an algorithm on n elements has O(n) work with probability at least $1-1/n^5$
  - This means that only once in about n^5 tries will the algorithm require more than O(n) work

華中科技大學
HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Expected vs. High Probability Bounds.

- we had 100 students take exams, most of the time each student takes 1 hour, but that once on every 100 exams or so, each student gets hung up and takes 101 hours
  - ➢ The average for each student is (99*1+1*101)/100 = 2 hours
  - ➢ the **expected maximum** will be close to 100 hours
    - ✓ on most exams with a hundred students one student will get hung up, so the expected maximum will be close to 100 hours, not 2 hours
  - ➢ Every student will finish in 2 hours with probability $1\text{-}1/n^5$

華中科技大學
HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Expected vs. High Probability Bounds.

- properties of summing vs. taking a maximum
  - analyze work using expectation
  - analyze span using high probability

why?

SYNOPSIS

- Discrete Probability: Let's Toss Some Dice
- Finding The Two Largest
- Find the k^{th} smallest element
- Quicksort
- Analysis of Quicksort

华中科技大学
HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Finding The Top Two Elements

```
max2 S =
let
  replace ((m1, m2), v) =
    if v ≤ m2 then (m1, m2)
    else if v ≤ m1 then (m1, v)
    else (v, m1)
  val init = if S1 ≥ S2 then (S1, S2) else (S2, S1)
in
  iter replace init S⟨3, ..., n⟩
end
```

- **We will do exact analysis**
- **$1+2(n-2)=2n-3$ comparisons in the worst case (Why?)**
- **A Divide and Conquer algorithm gives $3n/2-2$ (how?)**

Worst Case Analysis

```
max2 S =
let
    replace ((m_1, m_2), v) =
        if v ≤ m_2 then (m_1, m_2)
        else if v ≤ m_1 then (m_1, v)
        else (v, m_1)
    val init = if S_1 ≥ S_2 then (S_1, S_2) else (S_2, S_1)
in
    iter replace init S⟨3, ..., n⟩
end
```

- **An already sorted sequence (e.g.,<1,2,3,…,*n*>) will need exactly 2*n*–3 comparisons**
- **But this happens with 1/*n*! chance (Why?)**

A Randomized Algorithm

- The worst-case analysis is overly pessimistic (why?)
- Consider the following variant
  - On input of a sequence *S* of *n* elements
    - ✓1. Let *T*=*permute*(*S*,*π*), where *π* is a random permutation (i.e., we choose one of the *n*! permutations)
    - ✓2. Run the *max2* algorithm on *T*
  - No need to really generate the permutation!
    - ✓Just pick an unprocessed element randomly until all elements are processed
    - ✓It is convenient to model this by one initial permutation!

Analysis

```
1  max2 S =
2  let
3     replace ((m1, m2), v) =
4        if v ≤ m2 then (m1, m2)
5        else if v ≤ m1 then (m1, v)
6        else (v, m1)
7     val init = if S1 ≥ S2 then (S1, S2) else (S2, S1)
8  in
9     iter replace init S⟨3, ..., n⟩
10 end
```

- X_i : an indicator random variable, denoting whether Line 5 gets executed for the value at S_i
- Y is the number of comparisons

Analysis

```
1   max2 S =
2   let
3     replace ((m₁, m₂), v) =
4       if v ≤ m₂ then (m₁, m₂)
5       else if v ≤ m₁ then (m₁, v)
6       else (v, m₁)
7     val init = if S₁ ≥ S₂ then (S₁, S₂) else (S₂, S₁)
8   in
9     iter replace init S⟨3, ..., n⟩
10  end
```

- **Y=?**

$$Y(e) = \underbrace{1}_{\text{Line 7}} + \underbrace{n-2}_{\text{Line 4}} + \underbrace{\sum_{i=3}^{n} X_i(e)}_{\text{Line 5}}$$

Analysis

- This expression in true regardless of the random choice we're making
- We're interested in computing the expected value of *Y*
- By linearity of expectation, E[Y] =?

$$\begin{aligned}\mathbf{E}[Y] &= \mathbf{E}\left[1 + (n-2) + \sum_{i=3}^{n} X_i\right] \\ &= 1 + (n-2) + \sum_{i=3}^{n} \mathbf{E}[X_i]\,.\end{aligned}$$

Analysis

- Problem boils down to computing E[X_i], for i=3,…,n!
- What is the probability that $T_i > m_2$?
  - $T_i > m_2$ holds when T_i is either the largest or the second largest in {T_1,…, T_i}
- So, what is the probability that T_i is one of the two largest elements in a randomly permuted sequence of length i?
  - 1/i +1/i = 2/i
- $E[X_i]=1\cdot 2/i = 2/i$

華中科技大學

HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Analysis

$$\begin{aligned}
\mathbf{E}[Y] &= 1 + (n-2) + \sum_{i=3}^{n} \mathbf{E}[X_i] \\
&= 1 + (n-2) + \sum_{i=3}^{n} \frac{2}{i} \\
&= 1 + (n-2) + 2\left(\tfrac{1}{3} + \tfrac{1}{4} + \ldots \tfrac{1}{n}\right) \\
&= n - 4 + 2\left(1 + \tfrac{1}{2} + \tfrac{1}{3} + \tfrac{1}{4} + \ldots \tfrac{1}{n}\right) \\
&= n - 4 + 2H_n
\end{aligned}$$

- H_n is the n^{th} Harmonic number
- $H_n \leq 1 + \log_2 n$
- $E[Y] \leq n - 2 + 2\log_2 n$

SYNOPSIS

- Discrete Probability: Let's Toss Some Dice
- Finding The Two Largest
- Find the k^{th} smallest element
- Quicksort
- Analysis of Quicksort

Finding The K^{th} Smallest Element

- **Input: a sequence of *n* numbers (not necessarily sorted)**
- **Output: the K^{th} smallest value in *S* (i.e.,(**nth (sort S) k)**)**
- ***Requirement*: *O*(*n*) expected work and O($\log^2 n$) span**
- **We can't really sort the sequence!**

Finding The K^{th} Smallest Element

$$
\begin{array}{l}
\mathtt{kthSmallest}(k, S) \;=\; \textbf{let} \\
\quad p = S_0 \\
\quad L = \langle x \in S \mid x < p \rangle \\
\quad R = \langle x \in S \mid x > p \rangle \\
\textbf{in} \\
\quad \textbf{if}\;\; (k < |L|) \;\; \textbf{then} \;\; \mathtt{kthSmallest}(k, L) \\
\quad \textbf{else}\;\; \textbf{if}\;\; (k < |S| - |R|) \;\; \textbf{then} \;\; p \\
\quad \textbf{else}\;\; \mathtt{kthSmallest}(k - (|S| - |R|), R)
\end{array}
$$

- Let $X = \max\{|L|,|R|\} / |S|$, which

$$W(n) = W(X \cdot n) + O(n)$$

$$S(n) = S(X \cdot n) + O(\log n)$$

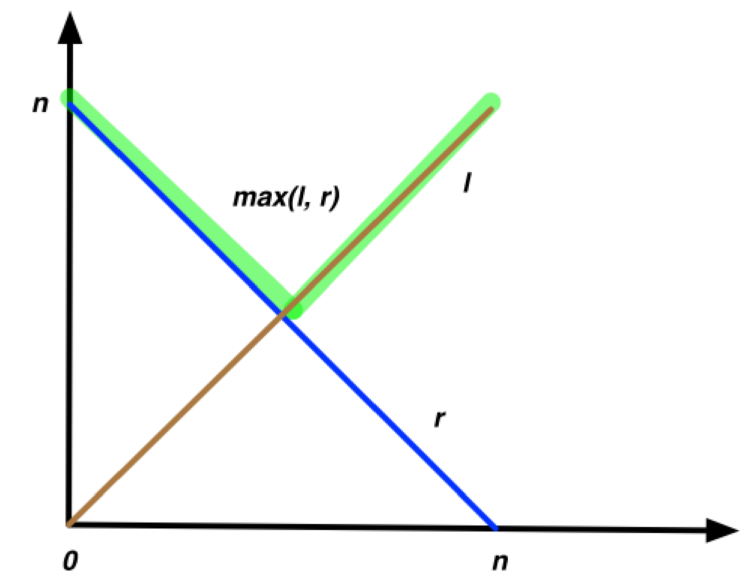
how?

Finding The K^{th} Smallest Element

- We want to find E [*X*]?
  - The probability that we land on a point on the x axis is 1/n:

$$\mathbf{E}[X] = \frac{1}{n}\sum_{i=0}^{n-1} \max\{i, n-i-1\}/n$$

  - the size of L and size of R:



  - E[X] $\leq \frac{1}{n}\sum_{j=n/2}^{n-1} \frac{2}{n} \cdot j \leq \frac{3}{4}$

$$\sum_{i=x}^{y} i = \frac{1}{2}(x+y)(y-x+1).$$

Finding The K^{th} Smallest Element

- **Theorem 10.16. Starting with size n, the expected size of S in algorithm *kthSmallest* after i recursive calls is $(3/4)^i n$**
  - **Let *Yi* be the random variable representing the size of the result after step (recursive call) i**

$$Y_i = n\prod_{j=1}^{i} X_j$$

$$\mathbf{E}[Y_i] = \mathbf{E}\left[n\prod_{j=1}^{i} X_j\right] = n\prod_{j=1}^{i}\mathbf{E}[X_j] \le \left(\frac{3}{4}\right)^i n$$

why?

Finding The *K^th* Smallest Element

- **W=?**
  - **The work at each step is linear:** $W_{contract}(n) \leq k_1 n + k_2$

$$
\begin{aligned}
\mathbf{E}\left[W_{\mathtt{kthSmallest}}(n)\right] &\leq \sum_{i=0}^{n}\left(k_1 n \left(\frac{3}{4}\right)^i + k_2\right) \\
&\leq k_1 n \left(\sum_{i=0}^{n}\left(\frac{3}{4}\right)^i\right) + k_2 n \\
&\leq 4k_1 n + k_2 n \\
&\in O(n)
\end{aligned}
$$

why?

華中科技大學
HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Finding The K^{th} Smallest Element

- S=?
  - The span at each step is linear: $S_{contract}(n) \le k_1 \log n + k_2$
- Bound the span of the algorithm
  - Consider step $i = 10 \log_2 n$
  - expected size upper bounded by $n(3/4)^{10\log_2 n}$
  - $n(3/4)^{10\log_2 n} \approx n \times n^{-10\log2(4/3)} \approx n^{-3.15}$
  - According Markov's Inequality:

$$\mathbf{Pr}\left[Y_{10\log_2 n} \ge 1\right] \le E[Y_{10\log_2 n}]/1 = n^{-3.15}$$

What this mean?

  - the number of steps is O(log n) with high probability
  - Each step has span O(log n) so the overall span is $O(\log^2 n)$ with high probability

SYNOPSIS

- Discrete Probability: Let's Toss Some Dice
- Finding The Two Largest
- Find the k^{th} smallest element
- Quicksort
- Analysis of Quicksort

Quicksort

- Originally invented and analyzed by Hoare in 1960's
- I strongly urge to watch Jon Bentley on "Three beautiful Quicksorts" at
  - www.youtube.com/watch?v=QvgYAQzg1z8

Sequential Quicksort

```
int i, j;
for( i = low, j = high - 1; ; )
{
    while( a[ ++i ] < pivot );
    while( pivot < a[ --j ] );
    if( i >= j )
    break;
    swap( a, i, j );
}
// Restore pivot
swap( a, i, high - 1 );
quicksort( a, low, i - 1 ); // Sort small elements
quicksort( a, i + 1, high ); // Sort large elements
```

Quicksort

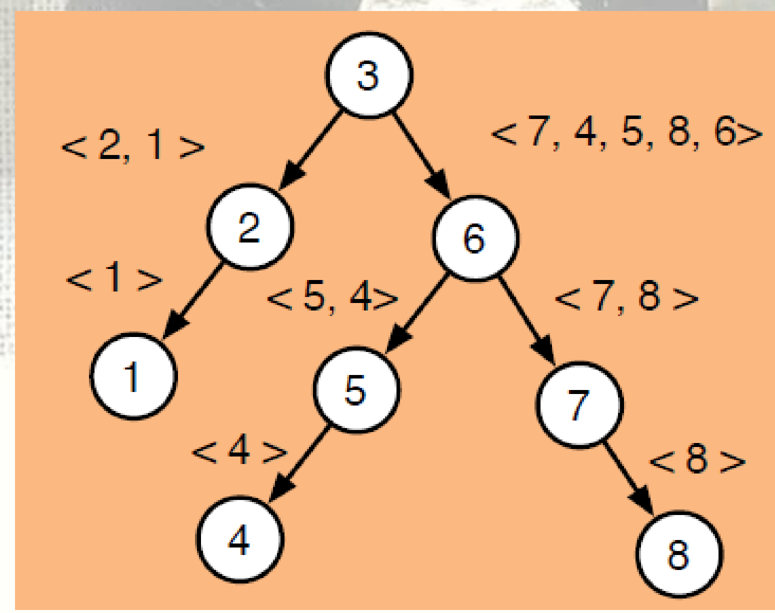
- **Is there parallelism in quicksort?**

```
sort S =
if |S| = 0 then S
else
    let
        p = pick a pivot from S
        S1 = ⟨ s ∈ S | s < p ⟩
        S2 = ⟨ s ∈ S | s = p ⟩
        S3 = ⟨ s ∈ S | s > p ⟩
        (R1, R3) = (sort S1 || sort S3)
    in
        R1 ++ S2 ++ R3
    end
```

Quicksort

- **Each call to Quicksort either makes**
  - **No recursive calls (base case), or**
  - **Two recursive calls**
- **Call tree is a binary**
- **Depth the call tree determines the span of the algorithm**

<7, 4, 2, 3, 5, 8, 1, 6>



Picking The Pivot

- **Always pick the first element**
  - Worst case $O(n^2)$ work — **Why?**
  - In practice, almost sorted inputs are not uncommon
- **Pick the median of 3 elements (e.g., first, middle and last elements)**
  - could possible divide evenly
  - worst case is still bad
- **Pick an element at random**
  - we hope this divides evenly in expectation
  - leading to expected $O(n\log n)$ work and $O(\log^2 n)$ span

SYNOPSIS

- **Discrete Probability: Let's Toss Some Dice**
- **Finding The Two Largest**
- **Find the *k*th smallest element**
- **Quicksort**
- **Analysis of Quicksort**

华中科技大学

HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Picking The Pivot

- **Pick first element**
  - **Worst case $O(n^2)$ work**
  - **Expected $O(n\log n)$ work**
    - **Averaged over all possible orderings**
  - **Work well on the average**
  - **Slow on some, possibly common, cases**
- **Pick a random element**
  - **Expected worst-case $O(n\log n)$ work**
    - **For input in any order, the expected work is $O(n\log n)$**
  - **No input has expected $O(n^2)$ work**
  - **With a small probability, we could be unlucky and have $O(n^2)$ work**

Randomized Quicksort

- Assign a uniformly random priority to each number in [0,1]

```
fun quicksort(S) =
   if |S| = 0 then S
   else let
      val p = pick as pivot the highest priority element from S
      val S1 = ⟨s ∈ S | s < p⟩
      val S2 = ⟨s ∈ S | s = p⟩
      val S3 = ⟨s ∈ S | s > p⟩
      val (R1, R3) = (quicksort(S1) || quicksort(S3))
    in
      append(R1, append(S2, R2))
    end
```

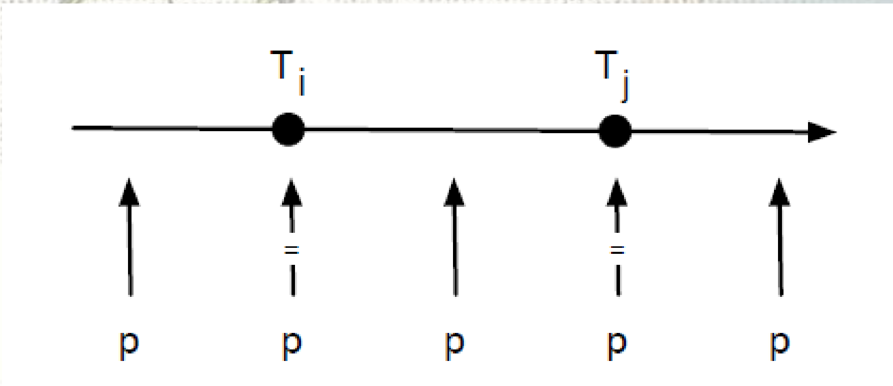
- Once the priorities are assigned, the algorithm is deterministic

Randomized Quicksort

- **Count comparisons made!**
  - **Almost all the work is comparisons**
    X_n = # of comparisons *quicksort*
    makes on input of size n
- **Find E[X_n] for any input sequence S**
- **Notation:**
  - **Let T=*sort (S)***
  - **T_i and T_j refer to elements in the final sorted order and $i < j$ and $T_i \leq T_j$**
  - **p_i refers to priority chosen for T_i**
  - **$A_{i,j}$ =1 if T_i and T_j were ever compared during the sort**

Analyzing Quicksort

- **Crucial point is how to model $A_{i,j}$**
- **In any one call to** quicksort, **there are three cases**
  - Pivot p is either T_i or $T_j \Rightarrow A_{i,j} = 1$
  - $T_i < p < T_j \Rightarrow T_i \in S_1$, $T_j \in S_2$, $A_{i,j} = 0$
  - Either $p < T_i$ or $p > T_j \Rightarrow T_i, T_j \in S_1$ or $T_i, T_j \in S_2$
- **If two elements are compared in a** quicksort **call, they will never be compared again in any other call!**



Why?

Analyzing Quicksort

$$X_n \leq \sum_{i=1}^{n} \sum_{j=i+1}^{n} A_{ij}$$

- **The non-optimized code compares each element to pivot 1 times**

...
val S_1= <s∈S| s<p>
val S_2= <s∈S| s=p>
val S_3= <s∈S| s>p>
...

- **By linearity of expectation**

$$\mathbf{E}[X_n] \leq \sum_{i=1}^{n} \sum_{j=i+1}^{n} \mathbf{E}[A_{ij}]$$

Analyzing Quicksort

- Consider first when the pivot is one of T_i, T_{i+1},..., T_j
- T_i and T_j are compared ⇔ p_i or p_j is the highest priority among {p_i, p_{i+1},..., p_j}
  - Assume T_k, $i<k<j$ has higher priority
  - For any subdivision···,T_i,···, T_k,···, T_j ， T_k will become a pivot and separate T_i and T_j
  - T_i and T_j will never be compared!

华中科技大学
HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Analyzing Quicksort

$$\begin{aligned}
\mathbf{E}[A_{ij}] &= \mathbf{Pr}[A_{ij} = 1] \\
&= \mathbf{Pr}[p_i \text{ or } p_j \text{ is the maximum in } \{p_i, \ldots, p_j\}] \\
&= \frac{2}{j - i + 1} \text{ (Why ?)}
\end{aligned}$$

- ***j*–*i* +1 elements between p_i and p_j and each is equally likely to be the maximum**
- **We want either p_i or p_j , hence 2/(*j*–*i*+1)**
- **T_i is compared to T_{i+1} with probability 1**

Analyzing Quicksort

$$
\begin{aligned}
\mathbf{E}[X_n] &\le \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathbf{E}[A_{ij}] \\
&= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\
&= \sum_{i=1}^{n-1} n \sum_{k=2}^{n-i+1} \frac{2}{k} \\
&\le 2 \sum_{i=1}^{n-1} H_n \\
&= 2nH_n \in O(n \log n)
\end{aligned}
$$

(Recall: $H_n = \ln n + O(1)$.)

Analyzing Quicksort

- **Indirectly, average work for basic deterministic quicksort is *O*(*n*log*n*)**
  - **Just shuffle data randomly and apply the basic algorithm**
  - **≡ to picking random priorities**

Exercise 10.20. *Rewrite the quicksort algorithm so that it takes a comparison function* $\texttt{cmp} : \alpha \times \alpha \to \texttt{order}$ *and a sequence of type* α *seq, and only uses the comparison once when comparing the pivot with each key. The type* $\texttt{order}$ *is the set* $\{\texttt{Less}, \texttt{Equal}, \texttt{Greater}\}$.

Alternative Analysis

- Write a recurrence for the number of comparisons :

$$X(n) = X(Y_n) + X(n-Y_n-1) + n-1$$

- Random variable Y_n is the size of S_1

$$\begin{aligned}
\mathbf{E}[X(n)] &= \mathbf{E}[X(Y_n) + X(n-Y_n-1) + n - 1] \\
&= \mathbf{E}[X(Y_n)] + \mathbf{E}[X(n-Y_n-1)] + n - 1 \\
&= \frac{1}{n}\sum_{i=0}^{n-1}(\mathbf{E}[X(i)] + \mathbf{E}[X(n-i-1)]) + n - 1
\end{aligned}$$

Why?

Alternative Analysis

$$\begin{aligned}\mathbf{E}[X(n)] &= \frac{1}{n}\sum_{i=0}^{n-1}(\mathbf{E}[X(i)] + \mathbf{E}[X(n-i-1)]) + n - 1 \\ &= \frac{2}{n}\sum_{i=0}^{n-1}\mathbf{E}[X(i)] + n - 1\end{aligned}$$

- **With telescoping, this also solves as *O*(*n*log*n*)**

Expected Span

- S is split into *L(ess)*,*E(qual)* and *(g)R(eater)*
- Let $X_n = \max\{|L|,|R|\}$
- We use filter to partition
    - $S(n) = S(X_n) + O(\log n)$ Why?

- The only thing is to calculate the depth of the pivot tree!

華中科技大學
HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Expected Span

- the depth of the pivot tree is O(log n) by relating it to the number of contraction steps of the randomized *kthSmallest*
- In *kthSmallest*, we have analyzed
  - each node had depth greater than 10 lg n with probability at most $1/n^{3.15}$
  - ***quicksort*** has any node of depth 10 lg n is also $1/n^{3.15}$?
  - It is wrong!

華中科技大學
HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Expected Span

- we have **multiple nodes** the probably increases that at least one will go above the bound
- Here is where we get to apply the **union bound**
  - For a collection of events $A_1,\ldots,A_n$ the bound is

$$\mathbf{Pr}\left[\bigcup_{1\le i\le n} A_i\right] \le \sum_{i=1}^{n} \mathbf{Pr}\left[A_i\right]$$

华中科技大学
HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Expected Span

- Here is where we get to apply the **union bound**
  - the individual events are the depths of each node being larger 10 lg n
  - the union is the probability that any of the nodes has depth larger than 10 lg n
  - There are n events each with probability $1/n^{3.15}$, so the union bound states

$$\mathbf{Pr}\left[\text{depth of quicksort pivot tree} > 10 \lg n\right] \leq \frac{n}{n^{3.15}} = \frac{1}{n^{2.15}}$$

Why?

Alternative Analysis

- **Bernoulli's Inequality**

$$(1+x)^{\alpha} \geq 1+\alpha \cdot x \quad (x>-1, \alpha>0)$$

- **Assume**

$$x=-\frac{1}{n^{3.15}}, \alpha=n$$

- **Then, we can get**

$$\left(1-\frac{1}{n^{3.15}}\right)^{n} \geq 1-n \cdot \frac{1}{n^{3.15}}=1-\frac{1}{n^{2.15}}$$

华中科技大学

HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Alternative Analysis

$$\Pr[\sum_{i=1}^{n}\sum_{j=i+1}^{n} A_{ij} > 0] = 1 - \Pr[\sum_{i=1}^{n}\sum_{j=i+1}^{n} A_{ij} = 0]$$

$$1 - \Pr[\prod_{i=1}^{n} x_i = 0]$$

$$= 1 - \prod_{i=1}^{n} \Pr[x_i = 0]$$

$$= 1 - (1 - \frac{1}{n^{3.15}})^n$$

$$\leq 1 - (1 - n \cdot \frac{1}{n^{3.15}})$$

$$= \frac{1}{n^{2.15}}$$

SUMMARY

- **Discrete Probability: Let’s Toss Some Dice**
- **Finding The Two Largest**
- **Finding The kth Smallest Element**
- **Quicksort**
- **Analysis of Quicksort**