PARALLEL AND SEQUENTIAL ALGORITHMS AND DATA STRUCTURES

LECTURE 6

Randomized Algorithms

明德厚学 求是创新

華中科技大學
HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

SYNOPSIS

- Introduction
- Order Statistics
- The Quick Sort Algorithm

Randomized Algorithms

- **Definition (Randomized Algorithm)**
  - **an algorithm is randomized if it makes random choices. Algorithms typically make their random choices by consulting a source of randomness such as a (pseudo-)random number generator.**
  - **E.X.**
  - ***Las Vegas algorithms* :** use randomization to weaken the cost guarantees (randomization is used to organize the computation in such a way that the impact is on the cost but not on the correctness)
  - ***Monte Carlo algorithms*** : use randomization to weaken the correctness guarantees of the computation

Advantages of Randomization

- **Simplicity**
  - ➢ randomization can simplify the design of algorithms, sometimes dramatically.
- **Efficiency**
  - ➢ randomization can improve efficiency, e.g., by facilitating "symmetry breaking" without relying on communication and coordination.
    - ✓ symmetry breaking: an algorithm's ability to distinguish between choices that otherwise look equivalent
- **Robustness**
  - ➢ randomization can improve the robustness of an algorithm, e.g., by reducing certain biases.

Disadvantages of Randomization

- **Complexity of Analysis**
  - **it usually complicates their analysis**
- **Uncertainty**
  - **Randomization can increase uncertainty**
  - **In some applications, such as real-time systems, this uncertainty may be unacceptable**

Analysis of Randomized Algorithms

- ***Expected bounds***
  - Expected bounds inform us about **the average cost across all random choices made by the algorithm**.
- ***high probability Bounds***
  - High-probability bounds inform us that **it is very unlikely that the cost will be above some bound**.
  - For an algorithm, we say that some property is true with high probability if it is true with probability p(n) such that $\lim_{n \to \infty} (p(n)) = 1,$

where n is an algorithm specific parameter, which is usually the instance size.

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

华中科技大学
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 $1-1/n^5$

Expected vs. High Probability Bounds.

- **Analyzing Expected Work**
  - ➢ **Expected bounds are quite convenient when analyzing work.**
    - ✓ **This is because the linearity of expectations allows adding expectations across the components of an algorithm to get the overall expected work.**
- **Analyzing Expected Span**
  - ➢ **High-probability bounds allow us to bound the expectation of the maximum of a number of random variables by showing that it is highly unlikely for any one of them to be large.**

華中科技大學
HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

SYNOPSIS

- Introduction
- Order Statistics
- The Quick Sort Algorithm

The Order Statistics Problem

- **Definition (Order Statistics Problem)**
  - Given a sequence, an integer k where 0≤k<|a|, and a comparison operation < that defines a total order over the elements of the sequence, find the kth order statistics, i.e., kth smallest element (counting from zero) in the sequence
  - We can solve this problem by reducing it to sorting. algorithm requires O(nlgn) work.
  - Can we achieve linear work and O($\lg^2 n$) span?

華中科技大學
HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Finding The Top Two Elements

max2 S =
let
    replace $((m_1, m_2), v)$ =
        if $v \le m_2$ **then** (m_1, m_2)
        else if $v \le m_1$ **then** (m_1, v)
        else (v, m_1)
    val *init* = **if** $S_1 \ge S_2$ **then** (S_1, S_2) **else** (S_2, S_1)
in
    iter replace init $S\langle 3, \ldots, n\rangle$
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 ((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
```

- 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!

华中科技大学
HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

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] =?

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

華中科技大學

HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

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]=1·2/i = 2/i

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
- $H_n \le 1+\log_2 n$
- $E[Y] \le n-2+2\log_2 n$

Randomized Algorithm for Order Statistics

- **Algorithm (Contraction-Based Select)**

$\mathit{select}\ a\ k =$
let
    $p =$ pick a uniformly random element from a
    $\ell = \langle x \in a \mid x < p \rangle$
    $r = \langle x \in a \mid x > p \rangle$
in
    if $(k < |\ell|)$ **then** $\mathit{select}\ \ell\ k$
    else if $(k < |a| - |r|)$ **then** p
    else $\mathit{select}\ r\ (k - (|a| - |r|))$
end

Analysis with the Order Statistics Algorithm

```
select a k =
let
  p = pick a uniformly random element from a
  ℓ = ⟨x ∈ a | x < p⟩
  r = ⟨x ∈ a | x > p⟩
in
  if (k < |ℓ|) then select ℓ k
  else if (k < |a| − |r|) then p
  else select r (k − (|a| − |r|))
end
```

- Let n=|a| and consider the partition of a into ℓ and r.

$$X(n) = \frac{\max\{|\ell|, |r|\}}{n}$$

- We can get

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

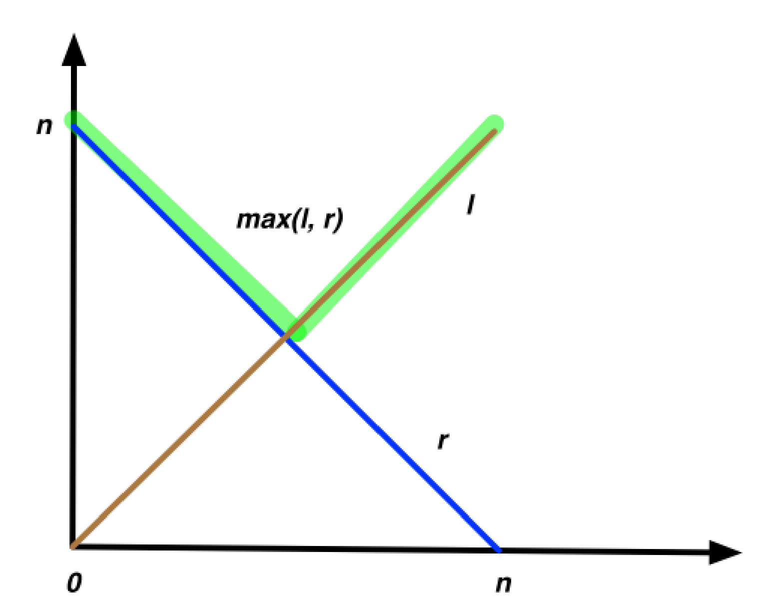
$$S(X(n) \le S(X(n) \cdot n) + O(\lg n)$$

why?

Analysis with the Order Statistics Algorithm

- We want to find E [*X*]?
  - the size of L and size of R:

  - The probability that we land on a point on the x axis is 1/n:
  - E[X]=？



$$\mathbf{E}[X(n)] = \frac{1}{n}\sum_{i=0}^{n-1}\frac{\max\{i, n-i-1\}}{n} \leq \frac{1}{n}\sum_{j=n/2}^{n-1}\frac{2}{n}\cdot j \leq \frac{3}{4}$$

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

华中科技大学
HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Analysis with the Order Statistics Algorithm

- Theorem. 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?

华中科技大学
HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Analysis with the Order Statistics Algorithm

- **W=?**
  - **The work at each level of the recursive calls is linear in the size of the input:**

    $W_{select}(n) \le k_1 n + k_2$,

    where n is the input size.

$$\mathbf{E}\left[W_{\texttt{select}}(n)\right] \le \sum_{i=0}^{n}\left(k_1 \mathbf{E}\left[Y_i\right] + k_2\right)$$

$$\mathbf{E}\left[W_{\texttt{select}}(n)\right] \le \sum_{i=0}^{n}\left(k_1 n \left(\frac{3}{4}\right)^i + k_2\right)$$

$$\le k_1 n \left(\sum_{i=0}^{n}\left(\frac{3}{4}\right)^i\right) + k_2 n$$

$$\le 4k_1 n + k_2 n$$

$$\in O(n).$$

why?

Analysis with the Order Statistics Algorithm

- **S=?**
  - **Because the span at each level is $O(\lg n)$ and because the depth is at most n, we can bound the span of the algorithm by $O(n \lg n)$ in the worst case.**
  - **But we expect the average span to be better because chances of picking a poor pivot over and over again, which would be required for the linear span is unlikely.**
- **A High-Probability Bound for Span**
  - **Consider depth d=10lg$n$.**
  - **At this depth, the expected instance size upper bounded by**

$$n\left(\frac{3}{4}\right)^{10 \lg n}.$$

  - **With a little math this is equal to $n \times n^{-10\lg(4/3)} \approx n^{-3.15}$.**

Analysis with the Order Statistics Algorithm

- **A High-Probability Bound for Span**
  - According Markov's Inequality:

$$\mathbf{P}\left[Y_{10 \lg n} \geq 1\right] \leq \frac{E[Y_{10 \lg n}]}{1} = \frac{1}{n^{3.15}} \leq \frac{1}{n^3}.$$

  - 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

- Introduction
- Order Statistics
- The Quick Sort Algorithm

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
```

華中科技大學

HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Quicksort

- **Is there parallelism in quicksort?**

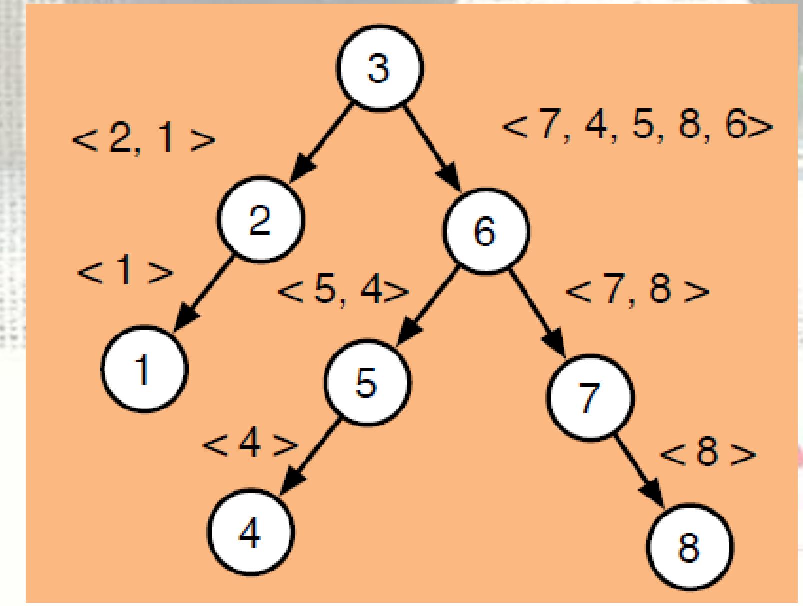
```
quicksort a =
  if |a| = 0 then a
  else
    let
      p = pick a pivot from a
      a1 = ⟨x ∈ a | x < p⟩
      a2 = ⟨x ∈ a | x = p⟩
      a3 = ⟨x ∈ a | x > p⟩
      (s1, s3) = (quicksort a1) || (quicksort a3)
    in
      s1++a2++s3
    end
```

华中科技大学
HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Quicksort

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

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



Pivot Selection

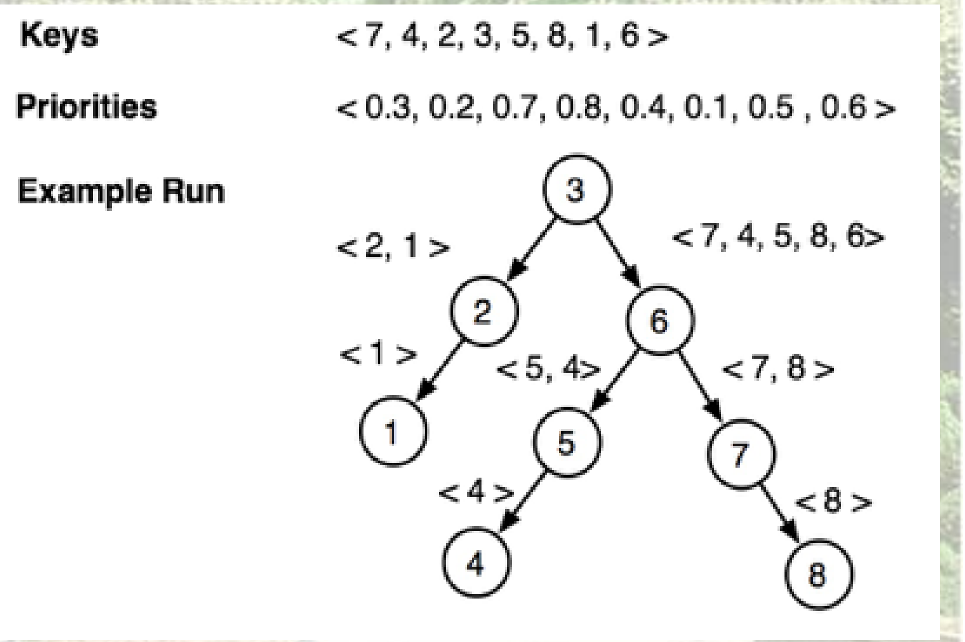
- Always pick the first element
  - Worst case $O(n^2)$ work — **Why?**
  - In practice, almost sorted inputs are not uncommon
- Pick the median of 3 elements (e.g., first, middle and last elements)
  - could possible divide evenly
  - worst case is still bad
- Pick an element at random
  - we hope this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

Pivot Selection

- **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

A Direct Analysis

- **two assumptions**
  - ➢ **We "simulate" randomness with priorities**
    - ✓ **before the start of the algorithm, we assign each key a priority uniformly at random from the real interval** $[0,1]$ **such that each key has a unique priority.**
    - ✓ **The algorithm then picks in Line 5 the key with the highest priority.**
  - ➢ **We assume a version of** quicksort **that compares the pivot** p **to each key in the input sequence once (instead of** 3 **times)**
  - ➢ **Notice: once the priorities are decided at the beginning, the algorithm is completely deterministic.**



A Direct Analysis

- **Random Variables**
  - **the random variable $Y(n)$**
    - **<u>the number of comparisons</u> <u>quicksort</u> <u>makes on input of size n</u>**
    - **find an upper bound on $\mathbf{E}[Y(n)]$**
  - **random variable X_{ij}**
    - **<u>indicates whether keys with rank i and j are compared</u>**
    - **consider the final sort of the keys t=sort(a) and for any element element t_i**
    - **Consider two positions $i,j \in \{0,\ldots,n-1\}$ in the sequence t and define following random variable**

$$X_{ij} = \begin{cases} 1 & \text{if } t_i \text{ and } t_j \text{ are compared by } \textit{quicksort} \\ 0 & \text{otherwise.} \end{cases}$$

A Direct Analysis

- **in any run of** quicksort**,** **<u>each pair of keys is compared at most once</u>**

$$Y(n) \leq \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} X_{ij}$$

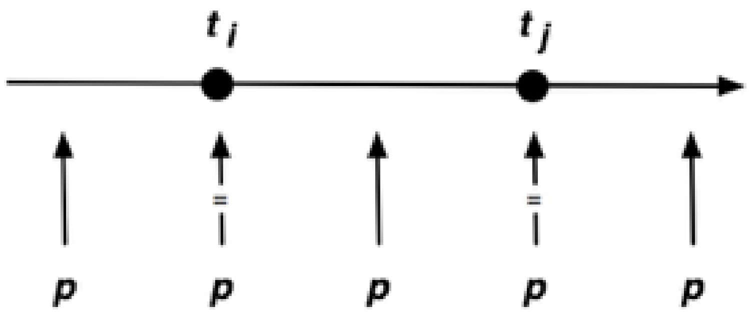
- **By linearity of expectation**

$$\mathbf{E}\left[Y(n)\right] \leq \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} \mathbf{E}\left[X_{ij}\right]$$

- **Since each** X_{ij} **is an indicator random variable,** $\mathbf{E}[X_{ij}]=\mathbf{P}[X_{ij}=1]$

A Direct Analysis

- **Calculating $\mathbf{P}[X_{ij}]$:** <u>**in any run of** quicksort**, each pair of keys is compared at most once**</u>
  - **For X_{ij}, where $i<j$, we distinguish between three possible scenarios**
    - ✓ **$p=t_i$ or $p=t_j$; in this case t_i and tj are compared and $X_{ij}=1$**
    - ✓ **$t_i<p<t_j$; in this case t_i is in a_1 and t_j is in a_3 and t_i and t_j will never be compared and $X_{ij}=0$.**
    - ✓ **$p<t_i$ or $p>t_j$; in this case t_i and t_j are either both in a_1 or both in a_3, respectively. Whether t_i and t_j are compared will be determined in some later recursive call to quicksort.**



A Direct Analysis

- **Lemma(Comparisons and Priorities)**
  - For $i<j$, let t_i and t_j be the keys with rank i and j, and p_i or p_j be their priorities. The keys t_i and t_j are compared if and only if either p_i or p_j has the highest priority among the priorities of keys with ranks $i \ldots j$.

A Direct Analysis

- **Bounding $\mathbf{E}[Y(n)]$**

$$\begin{aligned}\mathbf{E}[X_{ij}] &= \mathbf{P}[X_{ij} = 1] \\ &= \mathbf{P}[p_i \text{ or } p_j \text{ is the maximum among } \{p_i, \ldots, p_j\}] \\ &= \frac{2}{j - i + 1}.\end{aligned}$$

- **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**

A Direct Analysis

- **We can write the expected number of comparisons made in randomized** quicksort **is**

- **used the fact that** Hn=ln *n*+O(1)

Recall that $H_n = \sum_{k=1}^{n} \frac{1}{k}$ is the "harmonic number" for n.

$$
\begin{aligned}
\mathrm{E}\left[Y(n)\right] &\le \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} \mathrm{E}\left[X_{ij}\right] \\
&= \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} \frac{2}{j-i+1} \\
&= \sum_{i=0}^{n-1} \sum_{k=2}^{n-i} \frac{2}{k} \\
&\le 2 \sum_{i=0}^{n-1} H_n \\
&= 2nH_n \in O(n \lg n).
\end{aligned}
$$

華中科技大學
HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

A Direct Analysis

- **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**

Expected Span

- **Lemma(Quicksort and Order Statistics)**
  - **The path from the root to the ith node of the pivot tree is the same as the steps of select on $k=i$. That is to the say that <u>the distribution of pivots selected along the path</u> and <u>the sizes of each problem</u> is identical.**

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!

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

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

華中科技大學
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**

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

華中科技大學

HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Alternative Analysis

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

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

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

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

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

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

Another Alternative Analysis

- **Write a recurrence for the number of comparisons :**

  Y(n)=Y(X(n))+Y(n−X(n)−1)+n−1

- **the random variable** X(n) **is the size of the set** a1

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

Why?

Another 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*(*nlogn*)**

Another Alternative Analysis

- we have the following recurrence for span for input size n

$$S(n)=S(X(n))+O(\lg n).$$

- For the analysis, we shall condition the span on the random variable denoting the size of the maximum half and apply Total Expectations Theorem.

$$\mathbf{E}[S(n)] = \sum_{m=n/2}^{n} \mathbf{P}[X(n) = m] \cdot \mathbf{E}[S(n) \mid (X(n) = m)].$$

华中科技大学
HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Another Alternative Analysis

$$
\begin{aligned}
\mathbf{E}[S(n)] &= \sum_{x=n/2}^{n} \mathbf{P}[X(n)=x]\cdot \mathbf{E}[S(n)\mid (X(n)=x)] \\
&\le \mathbf{P}\left[X(n)\le \frac{3n}{4}\right]\cdot \mathbf{E}\left[S(\frac{3n}{4})\right] + \mathbf{P}\left[X(n) > \frac{3n}{4}\right]\cdot \mathbf{E}[S(n)] + c\cdot \lg n \\
&\le \frac{1}{2}\mathbf{E}\left[S(\frac{3n}{4})\right] + \frac{1}{2}\mathbf{E}[S(n)] + c\cdot \lg n \\
\Longrightarrow\ \mathbf{E}[S(n)] &\le \mathbf{E}\left[S(\frac{3n}{4})\right] + 2c\lg n.
\end{aligned}
$$

- **This is a recursion in** E[S(·)] **and solves easily to** $E[S(n)]=O(\lg^2 n)$.

Exercise

- Consider a game in which we draw some number of tasks at random such that a task has length n with probability 1/n and has length 1 otherwise. The expected length of a task is therefore bounded by 2. Imagine now drawing n tasks and waiting for all them to complete, assuming that each task can proceed in parallel independently of other tasks. Prove that the expected completion time is not constant.
- Prove that the pivot tree has O(lgn) height, and is therefore balanced, with high probability.