

Parallel And Sequential Data Structures and Algorithms

Randomized Algorithms I

Learning Objectives

- Learn how to apply expectation and **high probability bounds** to analyze work/span of randomized algorithms
- Understand the **Order Statistics** problem and develop an efficient randomized algorithm for it
- Analyze the expected work and span of **QuickSelect** via probability tools developed in earlier lectures

Analyzing Algorithms

Randomized Algorithms

Las Vegas Algorithm: A randomized algorithm whose **cost bounds are a random variable** but will always return the correct result.

Monte Carlo Algorithm: A randomized algorithm whose cost bounds are deterministic, but it may fail or return incorrect answers. We know with probability $p > 0$ it will be correct.

In 15-210 we only cover Las Vegas algorithms.

Example: QuickSort

Algorithm (QuickSort):

```
fun quicksort(S : sequence<T>) -> sequence<T>:  
  if length(S) <= 1:  
    return S  
  p = uniform_random_element(S) // need a source of randomness  
  L = filter(fn x => (x < p), S)  
  R = filter(fn x => (x > p), S)  
  M = filter(fn x => (x == p), S)  
  sortedL, sortedR = parallel (quicksort(L), quicksort(R))  
  return sortedL + M + sortedR
```

Note: This is a functional implementation of Quicksort. The partitioning step is done with three filters rather than in-place (which is harder to parallelize).

Randomized Algorithms in 15-210

- 2nd Largest
- Quickselect
- Quicksort
- Treaps
- Graph connectivity (via graph contraction)
- Minimum spanning tree (via graph contraction)
- Various others in recitations and labs

Why Randomized Algorithms

- Can be (much) simpler than deterministic ones
e.g., primality testing, treaps
- Can be more practical
e.g., quicksort
- Can give better asymptotic bounds
e.g., minimum cut of a graph, hash tables
- Particularly useful in parallel algorithms to "break symmetry"
- Are fun!!!

Finding 2nd Largest

Problem (2nd largest): Given a sequence of unique integers, find the 2nd largest.

```
fun secondLargest(S : sequence<int>) -> int
  vmax = -∞; v2max = -∞
  for i = 1 to |S|
    if (S[i-1] > v2max):
      if (S[i-1] > vmax): vmax ← S[i-1]; v2max ← vmax;
      else: v2max ← S[i-1]
  return v2max
```

How many comparisons in the worst case? And for what input?

For $n = |S|$ does $2n$ comparisons if sequence is **sorted** in increasing order

Finding 2nd Largest

```
fun secondLargest(S : sequence<int>) -> int
  vmax = -∞; v2max = -∞
  for i = 1 to |S|
    if (S[i-1] > v2max):
      if (S[i-1] > vmax): vmax ← S[i-1]; v2max ← vmax;
      else: v2max ← S[i-1]
  return v2max
```

What if the sequence is in a **random order**?

- How many expected comparisons?
- What is our sample space?
- What random variable should we use?

Finding 2nd Largest

```
fun secondLargest(S : sequence<int>) -> int
  vmax = -∞; v2max = -∞
  for i = 1 to |S|
    if (S[i-1] > v2max):
      if (S[i-1] > vmax): vmax ← S[i-1]; v2max ← vmax;
      else: v2max ← S[i-1]
  return v2max
```

- Ω = set of all permutations
- $|\Omega| = |S|!$
- $RV: X_i = \begin{cases} 1 & \text{if second comparison used on step } i \\ 0 & \text{otherwise} \end{cases}$

Finding 2nd Largest

Ω = set of all permutations, $|\Omega| = |S|!$

$$X_i = \begin{cases} 1 & \text{second comparison used on step } i \\ 0 & \text{otherwise} \end{cases}$$

$$Y = n + 1 + \sum_{i=2}^n X_i$$

RV for Total # of comparisons
All n steps do first comparison
Step 1 always does second comp, so we pull it out of the sum.

$$E[Y] = n + 1 + \sum_{i=2}^n E[X_i]$$

What theorem is this using?

$$E[X_i] = \Pr(X_i = 1) = \frac{2}{i} \quad (2/i \text{ chance that } i^{\text{th}} \text{ element is among max two})$$

Sidebar: Harmonic Numbers

Definition: The n^{th} **harmonic number** H_n is defined as the sum of the reciprocals of the first n positive integers, i.e.

$$H_n = \frac{1}{1} + \frac{1}{2} + \cdots + \frac{1}{n} = \sum_{i=1}^n \frac{1}{i}$$

Theorem: The n^{th} harmonic number H_n , satisfies

$$\ln n < H_n < \ln n + 1$$

In other words, $H_n = \Theta(\log n)$, which we are happy to see in algorithms!

Finding 2nd Largest

Ω = set of all permutations, $|\Omega| = |S|!$

$$X_i = \begin{cases} 1 & \text{second comparison used on step } i \\ 0 & \text{otherwise} \end{cases}$$

$$Y = n + 1 + \sum_{i=2}^n X_i$$

$$E[Y] = n + 1 + \sum_{i=2}^n E[X_i] = n + 1 + \sum_{i=2}^n \frac{2}{i} = n + 1 + 2(H_n - 1) < n - 1 + 2 \ln n$$

The **expected number of comparisons** for 2nd largest is: $n + \theta(\ln n)$

Skittles Game



Problem (Skittles Game): The Skittles game is played with a fair coin and a pile of n Skittles. It is a single-player game played in rounds.

In each round, the player flips the coin once. Say there are s skittles at the beginning of the round. When the coin lands on:

- **Heads:** eat $\lfloor s/2 \rfloor$ of the Skittles, so there are $\lfloor s/2 \rfloor$ Skittles remaining.
- **Tails:** do not eat any Skittles.

The game ends when there are no Skittles remaining.

We are interested in the behavior of the **random variable** $R(n)$ which is the number of rounds the Skittles game lasts when starting with n Skittles.

Skittles Game Bound Proof



Theorem (Skittles Game Bound): The Skittles game will end in $5\log_2 n$ rounds with high probability. In other words, $R(n) \in O(\log n)$ w.h.p.

Proof: Let X_r = a random variable for the number of Skittles after r rounds,
Let s_r = the actual number of Skittles after r rounds,
There is a $1/2$ probability of reducing the number of Skittles to $\lfloor s_r/2 \rfloor$.

$$\text{Thus, } X_{r+1} = \begin{cases} \frac{1}{2}s_r & \text{prob. } \frac{1}{2} \\ s_r & \text{prob. } \frac{1}{2} \end{cases} \Rightarrow \mathbb{E}[X_{r+1}|X_r = s_r] = \frac{1}{2}s_r + \frac{1}{2}\left\lfloor \frac{1}{2}s_r \right\rfloor \leq \frac{3}{4}s_r$$

Expecting the Expected



$$X_{r+1} = \begin{cases} \frac{1}{2}s_r & \text{prob. } \frac{1}{2} \\ s_r & \text{prob. } \frac{1}{2} \end{cases} \Rightarrow \mathbb{E}[X_{r+1}|X_r = s_r] \leq \frac{3}{4}s_r \Rightarrow \mathbb{E}[X_{r+1}] \leq \frac{3}{4}\mathbb{E}[X_r]$$

This step depends on something called **The Law of Total Expectation**, which says this: For any two random variables Y and X the following holds:

$$\mathbb{E}[Y] = \sum_x \mathbb{E}[Y|X = x] \Pr(X = x)$$

The notation $\mathbb{E}[Y|X = x]$ refers to *conditional expectation*. It's the expectation of Y given that $X = x$. You don't need to understand this for this class.

Skittles Game Bound Proof (Cont.)



Theorem (Skittles Game Bound): The Skittles game will end in $5\log_2 n$ rounds with high probability. In other words, $R(n) \in O(\log n)$ w.h.p.

Proof:

- We now unroll the recurrence to get $\mathbb{E}[X_r] = n \left(\frac{3}{4}\right)^r$
- Additionally, $\Pr[R(n) > r] = \Pr[X_r \geq 1] \leq \frac{\mathbb{E}[X_r]}{1} = n \left(\frac{3}{4}\right)^r$
Markov's Inequality

Skittles Game Bound Proof (Cont.)



Theorem (Skittles Game Bound): The Skittles game will end in $5\log_2 n$ rounds with high probability. In other words, $R(n) \in O(\log n)$ w.h.p.

Proof:

- To match this bound with the definition of high probability, we **want to find a value of r** that satisfies $n \left(\frac{3}{4}\right)^r < \frac{1}{n^k}$
- Taking logs, and rearranging gives us

$$r > (k + 1)\log_{\frac{4}{3}} n = (k + 1)\frac{\log_2 n}{\log_2(4/3)} \approx (k + 1)2.409\log_2 n$$

Skittles Game Bound Proof (Cont.)



Theorem (Skittles Game Bound): The Skittles game will end in $5\log_2 n$ rounds with high probability. In other words, $R(n) \in O(\log n)$ w.h.p.

Proof:

Since $2k \geq k + 1$ and $5 \geq 2 \cdot 2.409$, we can choose $r = k \cdot 5 \log_2 n$.

Hence, $\Pr[R(n) > k \cdot 5 \log_2 n] < \frac{1}{n^k}$ so

$R(n) \leq 5 k \log_2 n$ w.h.p. and $R(n) \in O(\log n)$ w.h.p. ■

Skittles Game Lemma



Lemma (Skittles Lemma): Let $X_0 = n$ and let X_{r+1} be computed by a random process from X_r that guarantees that:

- $0 \leq X_{r+1} \leq X_r$,
- $\mathbb{E}[X_{r+1}] \leq \alpha \mathbb{E}[X_r]$ where $0 < \alpha < 1$ is a constant.

Let $R(n)$ be a random variable which is the first round r when $X_r < 1$. Then $R(n)$ is $O(\log n)$ w.h.p., and $\mathbb{E}[X_r] \leq n \alpha^r$

(Specifically, $R(n) \leq \beta \log_2 n$ w.h.p., where $\beta = -2/\log_2 \alpha$.)

Order Statistics

Order Statistics

Definition (Rank): The **rank- k** element of a sequence is the k th element when the sequence is sorted in ascending order (zero-indexing). For example, 7 is the rank-5 element of $[7, 1, 1, 9, 5, 6, 7, 6]$.

Problem (Order Statistics): The **order statistics** problem takes a sequence of integers S and an integer k as input. Given these, our algorithm is supposed to return the rank- k element of S .

How could we solve this easily?

Inefficient Approach

- We could sort the sequence and return the k^{th} element

```
fun rank_k(S : sequence<T>, k : int) -> T:  
    return sort(S)[k]
```

- This costs $O(|S| \log |S|)$ work and $O(\log^2 |S|)$ span
- We sort the entire sequence just to get a single value out of it!

How can we make this faster...?

Randomization!

QuickSelect

QuickSelect (S, k)

- Pick a uniformly random pivot p in S
- Split S into two sequences (L, R) around p
 - L contains all values smaller than p , R contains all values larger
- Recurse on the side that contains the element we want
 - If $k < |L|$, the element we want must be in L
 - If $k \geq |S| - |R|$, the element we want must be in R
 - Otherwise, our pivot is the element we want

QuickSelect ([7,1,1,9,5,6,7,6], 5)

Looking for rank 5 in S

S 7, 1, 1, 9, 5, 6, 7, 6 $p = 5$

Looking for rank 2 in R

L 1, 1 5 7, 9, 6, 7, 6 R $p = 7$

Returns rank 0 in the pivot area

L 6, 6 7, 7 9 R

QuickSelect

```
fun quickSelect(S : sequence<T>, k : int) -> T:  
  p = a uniformly random element of S  
  L = filter(fn x => (x < p), S)  
  R = filter(fn x => (x > p), S)  
  if k < |L|: return quickSelect(L, k )  
  else if k >= |S|-|R|: return quickSelect(R, k - (|S|-|R|))  
  else: return p
```

Analysis of QuickSelect

Theorem (QuickSelect Recursion Depth): On a sequence of length n , quickselect terminates in $5 \log_2 n$ recursive calls with high probability.

Proof:

- Let X_r = length of sequence on which quickSelect is being called on round r^{th} recursive call (i.e., round).
 - Initially $r = 0$ and $X_0 = n$
- The algorithm may terminate at this round, but if it does not, then we let X_{r+1} be the size of the recursive subcall
- Note that L and R from the partitioning step have fewer elements than S because the pivot element is in S but not in L or R . Thus, $X_{r+1} < X_r$

Analysis of QuickSelect

Theorem (QuickSelect Recursion Depth): On a sequence of length n , quickselect terminates in $5 \log_2 n$ recursive calls with high probability.

Proof: In the worst case, the rank k will be such that the algorithm always does the recursive call *into the larger part*.

- Let $n' = X_r$, the current size of the array.
- When n' elements are split by a uniformly random pivot, the larger side of the split is uniformly distributed in $[n'/2, n' - 1]$
- The average value of these is at most $3/4 n'$
- Thus, $\mathbb{E}[X_{r+1} | X_r] < \frac{3}{4} X_r \Rightarrow \mathbb{E}[X_{r+1}] < \frac{3}{4} \mathbb{E}[X_r]$.

Analysis of QuickSelect

Theorem (QuickSelect Recursion Depth): On a sequence of length n , quickselect terminates in $5 \log_2 n$ recursive calls with high probability.

Proof:

- We just showed: $\mathbb{E}[X_{r+1}] < \frac{3}{4} \mathbb{E}[X_r]$, and $X_{r+1} < X_r$
- So, the **Skittles Lemma** applies and gives the desired result.

Analysis of QuickSelect

Theorem (Cost of QuickSelect): The expected work of quickselect on a sequence of length n is $O(n)$. The span is $O(\log^2 n)$ with high probability.

Proof (Work): On each round we do $O(n)$ work for the filters where n is the size of the input sequence for that round, which is bounded by $c_1 n + c_2$

- Intuitively we might try recurrence of the form $W(n) = W(3/4n) + O(n)$...

This is unjustified!! $\mathbf{E}[W(X_r)] \neq W(\mathbf{E}[X_r])$ in general.

- But we know from the skittles lemma that $\mathbf{E}[X_r] < n \left(\frac{3}{4}\right)^r$ so we have

$$\mathbf{E}[W(n)] < \sum_{r=0}^{\infty} c_1 \mathbf{E}[X_r] + c_2 < \sum_{r=0}^{\infty} c_1 n \left(\frac{3}{4}\right)^r + c_2 \in O(n)$$

Analysis of QuickSelect

Theorem (Cost of QuickSelect): The expected work of quickselect on a sequence of length n is $O(n)$. The span is $O(\log^2 n)$ with high probability.

Proof (Span): On each round of quickselect, we do $O(\log |S|) \leq O(\log n)$ span where S is the input sequence for that round

- Picking a uniformly random element takes constant time
- The filter operations have logarithmic span
- The recursion depth is $O(\log n)$ with high probability, therefore

$$S(n) = O(\log n \cdot \log n) \Rightarrow O(\log^2 n)$$

Summary

- The Skittles game gives us an example of how to derive/prove **high probability bounds**
- The Skittles Game generalizes into the **Skittles Lemma** which is a powerful tool for analyzing randomized algorithms
- QuickSelect give us an expected $O(n)$ work and $O(\log^2 n)$ span w.h.p. algorithm for solving the **Order Statistics** problem
- In general, randomized algorithms can give us solutions that work more **efficiently in expectation** than a corresponding deterministic algorithm