Randomized Algorithms II – High Probability

Lecture 10
September 26, 2013
Part I

Movie...
Part II

Understanding the binomial distribution
Binomial distribution

\(X_n = \) numbers of heads when flipping a coin \(n\) times.

**Claim**

\[ \Pr[X_n = i] = \frac{\binom{n}{i}}{2^n} . \]

Where: \( \binom{n}{k} = \frac{n!}{(n-k)!k!} \).

Indeed, \( \binom{n}{i} \) is the number of ways to choose \(i\) elements out of \(n\) elements (i.e., pick which \(i\) coin flip come up heads).

Each specific such possibility (say 0100010...) had probability \(1/2^n\).
Massive randomness.. Is not that random.

Consider flipping a fair coin \( n \) times independently, head given 1, tail gives zero. How many heads? ...we get a binomial distribution.
Consider flipping a fair coin $n$ times independently, head given 1, tail gives zero. How many heads? ...we get a binomial distribution.
Consider flipping a fair coin $n$ times independently, head given 1, tail gives zero. How many heads? ...we get a binominal distribution.
Consider flipping a fair coin $n$ times independently, head given 1, tail gives zero. How many heads? ...we get a binomial distribution.
Consider flipping a fair coin $n$ times independently, head given $1$, tail gives zero. How many heads? ...we get a binomial distribution.
Consider flipping a fair coin $n$ times independently, head given 1, tail gives zero. How many heads? ...we get a binomial distribution.
Massive randomness.. Is not that random.

Consider flipping a fair coin $n$ times independently, head given 1, tail gives zero. How many heads? ...we get a binomial distribution.
Massive randomness.. Is not that random.

Consider flipping a fair coin $n$ times independently, head given 1, tail gives zero. How many heads? ...we get a binominal distribution.

![Probability distribution graph](image-url)
Massive randomness.. Is not that random.

Consider flipping a fair coin $n$ times independently, head given 1, tail gives zero. How many heads? ...we get a binomial distribution.
Massive randomness.. Is not that random.

Consider flipping a fair coin $n$ times independently, head given 1, tail gives zero. How many heads? ...we get a binomial distribution.
Massive randomness.. Is not that random.

Consider flipping a fair coin $n$ times independently, head given 1, tail gives zero. How many heads? ...we get a binomial distribution.
Massive randomness.. Is not that random.

Consider flipping a fair coin $n$ times independently, head given 1, tail gives zero. How many heads? ...we get a binomial distribution.
Massive randomness.. Is not that random.

This is known as concentration of mass.
This is a very special case of the law of large numbers.
Side note...
Law of large numbers (weakest form)...

Informal statement of law of large numbers
For $n$ large enough, the middle portion of the binomial distribution looks like (converges to) the normal/Gaussian distribution.
Massive randomness.. Is not that random.

Intuitive conclusion
Randomized algorithm are unpredictable in the tactical level, but very predictable in the strategic level.
Part III

QuickSort with high probability
Show that **QuickSort** running time is $O(n \log n)$.

1. **QuickSort** picks a pivot, splits into two subproblems, and continues recursively.
2. Track single element in input.
3. Game ends, when this element is alone in subproblem.
4. Show every element in input, participates $\leq 32 \ln n$ rounds (with high enough probability).
5. $\mathcal{E}_i$: event $i$th element participates $> 32 \ln n$ rounds.
6. $C_{QS}$: number of comparisons performed by **QuickSort**.
7. Running time $O(C_{QS})$.
8. Probability of failure is
   \[
   \alpha = \Pr\left[ C_{QS} \geq 32n \ln n \right] \leq \Pr\left[ \bigcup_i \mathcal{E}_i \right] \leq \sum_{i=1}^n \Pr\left[ \mathcal{E}_i \right].
   \]
   ... by the union bound.
Show that \textbf{QuickSort} running time is $O(n \log n)$.

1. \textbf{QuickSort} picks a pivot, splits into two subproblems, and continues recursively.

2. Track single element in input.

3. Game ends, when this element is alone in subproblem.

4. Show every element in input, participates $\leq 32 \ln n$ rounds (with high enough probability).

5. $\mathcal{E}_i$: event $i$th element participates $> 32 \ln n$ rounds.

6. $C_{QS}$: number of comparisons performed by \textbf{QuickSort}.

7. Running time $O(C_{QS})$.

8. Probability of failure is

$$\alpha = \Pr\left[C_{QS} \geq 32n \ln n\right] \leq \Pr[\bigcup_i \mathcal{E}_i] \leq \sum_{i=1}^{n} \Pr[\mathcal{E}_i].$$

... by the union bound.
Show that QuickSort running time is $O(n \log n)$

1. QuickSort picks a pivot, splits into two subproblems, and continues recursively.

2. Track single element in input.

3. Game ends, when this element is alone in subproblem.

4. Show every element in input, participates $\leq 32 \ln n$ rounds (with high enough probability).

5. $E_i$: event $i$th element participates $> 32 \ln n$ rounds.

6. $C_{QS}$: number of comparisons performed by QuickSort.

7. Running time $O(C_{QS})$.

8. Probability of failure is
   
   $$\alpha = \Pr[C_{QS} \geq 32n \ln n] \leq \Pr[\bigcup_i E_i] \leq \sum_{i=1}^{n} \Pr[E_i].$$
   
   ... by the union bound.
Show that $\text{QuickSort}$ running time is $O(n \log n)$.

1. $\text{QuickSort}$ picks a pivot, splits into two subproblems, and continues recursively.

2. Track single element in input.

3. Game ends, when this element is alone in subproblem.

4. Show every element in input, participates $\leq 32 \ln n$ rounds (with high enough probability).

5. $\mathcal{E}_i$: event $i$th element participates $> 32 \ln n$ rounds.

6. $C_{QS}$: number of comparisons performed by $\text{QuickSort}$.

7. Running time $O(C_{QS})$.

8. Probability of failure is

$$
\alpha = \Pr \left[ C_{QS} \geq 32n \ln n \right] \leq \Pr \left[ \bigcup_i \mathcal{E}_i \right] \leq \sum_{i=1}^{n} \Pr \left[ \mathcal{E}_i \right].
$$

... by the union bound.
Show that \textbf{QuickSort} running time is $O(n \log n)$.

1. \textbf{QuickSort} picks a pivot, splits into two subproblems, and continues recursively.
2. Track single element in input.
3. Game ends, when this element is alone in subproblem.
4. Show every element in input, participates $\leq 32 \ln n$ rounds (with high enough probability).
5. $\mathcal{E}_i$: event $i$th element participates $> 32 \ln n$ rounds.
6. $C_{\text{QS}}$: number of comparisons performed by \textbf{QuickSort}.
7. Running time $O(C_{\text{QS}})$.
8. Probability of failure is
   \[
   \alpha = \Pr[C_{\text{QS}} \geq 32n \ln n] \leq \Pr[\bigcup_i \mathcal{E}_i] \leq \sum_{i=1}^n \Pr[\mathcal{E}_i].
   \]
   ... by the union bound.
Show that **QuickSort** running time is $O(n \log n)$.

1. **QuickSort** picks a pivot, splits into two subproblems, and continues recursively.
2. Track single element in input.
3. Game ends, when this element is alone in subproblem.
4. Show every element in input, participates $\leq 32 \ln n$ rounds (with high enough probability).
5. $\mathcal{E}_i$: event $i$th element participates $> 32 \ln n$ rounds.
6. $C_{QS}$: number of comparisons performed by **QuickSort**.
7. Running time $O(C_{QS})$.
8. Probability of failure is
   
   $\alpha = \Pr\left[C_{QS} \geq 32n \ln n\right] \leq \Pr[\bigcup_i \mathcal{E}_i] \leq \sum_{i=1}^{n} \Pr[\mathcal{E}_i].$
   ... by the union bound.
Show that **QuickSort** running time is $O(n \log n)$.

1. **QuickSort** picks a pivot, splits into two subproblems, and continues recursively.
2. Track single element in input.
3. Game ends, when this element is alone in subproblem.
4. Show every element in input, participates $\leq 32 \ln n$ rounds (with high enough probability).
5. $\mathcal{E}_i$: event $i$th element participates $> 32 \ln n$ rounds.
6. $C_{QS}$: number of comparisons performed by **QuickSort**.
7. Running time $O(C_{QS})$.
8. Probability of failure is

\[ \alpha = \Pr[C_{QS} \geq 32n \ln n] \leq \Pr[\bigcup_i \mathcal{E}_i] \leq \sum_{i=1}^{n} \Pr[\mathcal{E}_i]. \]

... by the union bound.
Show that \textbf{QuickSort} running time is $O(n \log n)$.

1. \textbf{QuickSort} picks a pivot, splits into two subproblems, and continues recursively.
2. Track single element in input.
3. Game ends, when this element is alone in subproblem.
4. Show every element in input, participates $\leq 32 \ln n$ rounds (with high enough probability).
5. $\mathcal{E}_i$: event $i$th element participates $> 32 \ln n$ rounds.
6. $C_{QS}$: number of comparisons performed by \textbf{QuickSort}.
7. Running time $O(C_{QS})$.
8. Probability of failure is
   \[
   \alpha = \Pr \left[ C_{QS} \geq 32n \ln n \right] \leq \Pr \left[ \bigcup_i \mathcal{E}_i \right] \leq \sum_{i=1}^{n} \Pr \left[ \mathcal{E}_i \right].
   \]
   ... by the union bound.
Show that \textbf{QuickSort} running time is $O(n \log n)$.

1. Probability of failure is
   \[ \alpha = \Pr[C_{QS} \geq 32n \ln n] \leq \Pr[\bigcup_i \mathcal{E}_i] \leq \sum_{i=1}^{n} \Pr[\mathcal{E}_i]. \]

2. \textit{Union bound}: for any two events $A$ and $B$:
   \[ \Pr[A \cup B] \leq \Pr[A] + \Pr[B]. \]

3. Assume: $\Pr[\mathcal{E}_i] \leq 1/n^3$.

4. Bad probability...
   \[ \alpha \leq \sum_{i=1}^{n} \Pr[\mathcal{E}_i] \leq \sum_{i=1}^{n} \frac{1}{n^3} = \frac{1}{n^2}. \]

5. $\implies$ \textbf{QuickSort} performs $\leq 32n \ln n$ comparisons, w.h.p.

6. $\implies$ \textbf{QuickSort} runs in $O(n \log n)$ time, with high probability.
Show that **QuickSort** running time is $O(n \log n)$.

1. Probability of failure is
   \[ \alpha = \Pr[C_{QS} \geq 32n \ln n] \leq \Pr[\bigcup_i E_i] \leq \sum_{i=1}^{n} \Pr[E_i]. \]

2. **Union bound**: for any two events $A$ and $B$:
   \[ \Pr[A \cup B] \leq \Pr[A] + \Pr[B]. \]

3. Assume: $\Pr[E_i] \leq 1/n^3$.

4. Bad probability...
   \[ \alpha \leq \sum_{i=1}^{n} \Pr[E_i] \leq \sum_{i=1}^{n} \frac{1}{n^3} = \frac{1}{n^2}. \]

5. $\implies$ **QuickSort** performs $\leq 32n \ln n$ comparisons, w.h.p.

6. $\implies$ **QuickSort** runs in $O(n \log n)$ time, with high probability.
Show that \textbf{QuickSort} running time is $O(n \log n)$

1. Probability of failure is
   \[
   \alpha = \Pr[ C_{QS} \geq 32n \ln n ] \leq \Pr[ \bigcup_i E_i ] \leq \sum_{i=1}^n \Pr[ E_i ].
   \]

2. **Union bound**: for any two events $A$ and $B$:
   \[
   \Pr[ A \cup B ] \leq \Pr[ A ] + \Pr[ B ].
   \]

3. Assume: $\Pr[ E_i ] \leq \frac{1}{n^3}$.

4. Bad probability...
   \[
   \alpha \leq \sum_{i=1}^n \Pr[ E_i ] \leq \sum_{i=1}^n \frac{1}{n^3} = \frac{1}{n^2}.
   \]

5. $\implies$ \textbf{QuickSort} performs $\leq 32n \ln n$ comparisons, w.h.p.

6. $\implies$ \textbf{QuickSort} runs in $O(n \log n)$ time, with high probability.
Show that \textbf{QuickSort} running time is \(O(n \log n)\).

1. Probability of failure is
\[
\alpha = \Pr[C_{QS} \geq 32n \ln n] \leq \Pr[\bigcup_i E_i] \leq \sum_{i=1}^n \Pr[E_i].
\]

2. \textit{Union bound}: for any two events \(A\) and \(B\):
\[
\Pr[A \cup B] \leq \Pr[A] + \Pr[B].
\]

3. Assume: \(\Pr[E_i] \leq \frac{1}{n^3}\).

4. Bad probability...
\[
\alpha \leq \sum_{i=1}^n \Pr[E_i] \leq \sum_{i=1}^n \frac{1}{n^3} = \frac{1}{n^2}.
\]

5. \(\Rightarrow\) \textbf{QuickSort} performs \(\leq 32n \ln n\) comparisons, w.h.p.

6. \(\Rightarrow\) \textbf{QuickSort} runs in \(O(n \log n)\) time, with high probability.
Show that **QuickSort** running time is $O(n \log n)$

1. Probability of failure is
   $$\alpha = \Pr\left[C_{QS} \geq 32n \ln n\right] \leq \Pr[\bigcup_i E_i] \leq \sum_{i=1}^n \Pr[E_i] .$$

2. **Union bound**: for any two events $A$ and $B$:
   $$\Pr[A \cup B] \leq \Pr[A] + \Pr[B].$$

3. Assume: $\Pr[E_i] \leq 1/n^3$.

4. Bad probability...
   $$\alpha \leq \sum_{i=1}^n \Pr[E_i] \leq \sum_{i=1}^n \frac{1}{n^3} = \frac{1}{n^2} .$$

5. $\implies$ **QuickSort** performs $\leq 32n \ln n$ comparisons, w.h.p.

6. $\implies$ **QuickSort** runs in $O(n \log n)$ time, with high probability.
Proving that an element...

...participates in small number of rounds.

1. $n$: number of elements in input for $\text{QuickSort}$.
2. $x$: Arbitrary element $x$ in input.
4. $S_i$: input to $i$th level recursive call that include $x$.
5. $x$ **lucky** in $j$th iteration, if balanced split...
   \[ |S_{j+1}| \leq (\frac{3}{4}) |S_j| \text{ and } |S_j \setminus S_{j+1}| \leq (\frac{3}{4}) |S_j| \]
6. $Y_j = 1 \iff x$ lucky in $j$th iteration.
7. $\Pr[Y_j] = \frac{1}{2}$.
8. Observation: $Y_1, Y_2, \ldots, Y_m$ are independent variables.
9. $x$ can participate $\leq \rho = \log_{\frac{4}{3}} n \leq 3.5 \ln n$ rounds.
10. ...since $|S_j| \leq n(\frac{3}{4})$ # of lucky iteration in $1 \ldots j$.
11. If $\rho$ lucky rounds in first $k$ rounds $\implies |S_k| \leq (\frac{3}{4})^\rho n \leq 1$. 
Proving that an element...

... participates in small number of rounds.

1. \( n \): number of elements in input for \texttt{QuickSort}.
2. \( x \): Arbitrary element \( x \) in input.
3. \( S_1 \): Input.
4. \( S_i \): input to \( i \)th level recursive call that include \( x \).
5. \( x \) \textit{lucky} in \( j \)th iteration, if balanced split...
   \[ |S_{j+1}| \leq \left(\frac{3}{4}\right) |S_j| \text{ and } |S_j \setminus S_{j+1}| \leq \left(\frac{3}{4}\right) |S_j| \]
6. \( Y_j = 1 \iff x \text{ lucky in } j \)th iteration.
7. \( \Pr[Y_j] = \frac{1}{2} \).
8. Observation: \( Y_1, Y_2, \ldots, Y_m \) are independent variables.
9. \( x \) can participate \( \leq \rho = \log_{\frac{4}{3}} n \leq 3.5 \ln n \) rounds.
10. ...since \( |S_j| \leq n \left(\frac{3}{4}\right)^j \# \) of lucky iteration in \( 1 \ldots j \).
11. If \( \rho \) lucky rounds in first \( k \) rounds \( \implies |S_k| \leq \left(\frac{3}{4}\right)^\rho n \leq 1 \).
Proving that an element...
...participates in small number of rounds.

1. \( n \): number of elements in input for \texttt{QuickSort}.
2. \( x \): Arbitrary element \( x \) in input.
3. \( S_1 \): Input.
4. \( S_i \): input to \( i \)th level recursive call that include \( x \).
5. \( x \) \textit{lucky} in \( j \)th iteration, if balanced split...
   \[ |S_{j+1}| \leq \left(\frac{3}{4}\right) |S_j| \text{ and } |S_j \setminus S_{j+1}| \leq \left(\frac{3}{4}\right) |S_j| \]
6. \( Y_j = 1 \iff x \) lucky in \( j \)th iteration.
7. \( \Pr[Y_j] = \frac{1}{2} \).
8. Observation: \( Y_1, Y_2, \ldots, Y_m \) are independent variables.
9. \( x \) can participate \( \leq \rho = \log_{4/3} n \leq 3.5 \ln n \) rounds.
10. ...since \( |S_j| \leq n \left(\frac{3}{4}\right)^\# \) of lucky iteration in \( 1 \ldots j \).
11. If \( \rho \) lucky rounds in first \( k \) rounds \( \implies |S_k| \leq \left(\frac{3}{4}\right)^\rho n \leq 1. \)
Proving that an element...  
... participates in small number of rounds.

1. \( n \): number of elements in input for QuickSort.
2. \( x \): Arbitrary element \( x \) in input.
3. \( S_1 \): Input.
4. \( S_i \): input to \( i \)th level recursive call that include \( x \).
5. \( x \) *lucky* in \( j \)th iteration, if balanced split...
   \[ |S_{j+1}| \leq (3/4) |S_j| \text{ and } |S_j \setminus S_{j+1}| \leq (3/4) |S_j| \]
6. \( Y_j = 1 \iff x \) lucky in \( j \)th iteration.
7. \( \Pr[Y_j] = \frac{1}{2} \).
8. Observation: \( Y_1, Y_2, \ldots, Y_m \) are independent variables.
9. \( x \) can participate \( \leq \rho = \log_{4/3} n \leq 3.5 \ln n \) rounds.
10. ...since \( |S_j| \leq n(3/4)^j \) # of lucky iteration in \( 1 \ldots j \).
11. If \( \rho \) lucky rounds in first \( k \) rounds \( \implies |S_k| \leq (3/4)^\rho n \leq 1 \).
Proving that an element...

... participates in small number of rounds.

1 Brain reset!

2 Q: How many rounds $x$ participates in = how many coin flips till one gets $\rho$ heads?

3 A: In expectation, $2\rho$ times.
Proving that an element...  
... participates in small number of rounds.

1. Brain reset!

2. Q: How many rounds $x$ participates in $=$ how many coin flips till one gets $\rho$ heads?

3. A: In expectation, $2^\rho$ times.
Proving that an element...

... participates in small number of rounds.

1. Brain reset!

2. Q: How many rounds $x$ participates in $= \text{how many coin flips till one gets } \rho \text{ heads}$?

3. A: In expectation, $2\rho$ times.
Proving that an element...

... participates in small number of rounds.

1. Assume the following:

Lemma

In $M$ coin flips: $\Pr[\#\ heads \leq M/4] \leq \exp(-M/8)$.

2. Set $M = 32 \ln n \geq 8\rho$.

3. $\Pr[Y_j = 0] = \Pr[Y_j = 1] = 1/2$.

4. $Y_1, Y_2, \ldots, Y_M$ are independent.

5. $\implies$ probability $\leq \rho \leq M/4$ ones in $Y_1, \ldots, Y_M$ is

\[
\leq \exp\left(-\frac{M}{8}\right) \leq \exp(-\rho) \leq \frac{1}{n^3}.
\]

6. $\implies$ probability $x$ participates in $M$ recursive calls of $\text{QuickSort}$ $\leq 1/n^3$. 
Proving that an element...  
... participates in small number of rounds.

1. Assume the following:

**Lemma**

In $M$ coin flips: $\Pr[\# \text{ heads } \leq M/4] \leq \exp(-M/8)$.

2. Set $M = 32 \ln n \geq 8\rho$.

3. $\Pr[Y_j = 0] = \Pr[Y_j = 1] = 1/2$.

4. $Y_1, Y_2, \ldots, Y_M$ are independent.

5. $\implies$ probability $\leq \rho \leq M/4$ ones in $Y_1, \ldots, Y_M$ is

\[ \leq \exp\left(-\frac{M}{8}\right) \leq \exp(-\rho) \leq \frac{1}{n^3}. \]

6. $\implies$ probability $x$ participates in $M$ recursive calls of QuickSort $\leq 1/n^3$. 

Sariel (UIUC)  
CS573  
Fall 2013  

14 / 44
Proving that an element... 
... participates in small number of rounds.

1. $n$ input elements. Probability depth of recursion in \texttt{QuickSort} $> 32 \ln n$ is $\leq (1/n^3) \times n = 1/n^2$.

2. Result:

\begin{itemize}
  \item \textbf{Theorem}
  
  With high probability (i.e., $1 - 1/n^2$) the depth of the recursion of \texttt{QuickSort} is $\leq 32 \ln n$. Thus, with high probability, the running time of \texttt{QuickSort} is $O(n \log n)$.
\end{itemize}

3. Same result holds for \texttt{MatchNutsAndBolts}.
1. $n$ input elements. Probability depth of recursion in \texttt{QuickSort} $> 32 \ln n$ is $\leq (1/n^3) \times n = 1/n^2$.

2. **Result:**

**Theorem**

*With high probability (i.e., $1 - 1/n^2$) the depth of the recursion of \texttt{QuickSort} is $\leq 32 \ln n$. Thus, with high probability, the running time of \texttt{QuickSort} is $O(n \log n)$.*

3. Same result holds for \texttt{MatchNutsAndBolts}. 
Proving that an element...

...participates in small number of rounds.

1. \( n \) input elements. Probability depth of recursion in QuickSort

\[ > 32 \ln n \leq (1/n^3) \ast n = 1/n^2. \]

2. Result:

**Theorem**

*With high probability (i.e., \( 1 - 1/n^2 \)) the depth of the recursion of QuickSort is \( \leq 32 \ln n \). Thus, with high probability, the running time of QuickSort is \( O(n \log n) \).*

3. Same result holds for MatchNutsAndBolts.
Alternative proof of high probability of QuickSort

1. \( T \): \( n \) items to be sorted.

2. \( t \in T \): element.

3. \( X_i \): the size of subproblem in \( i \)th level of recursion containing \( t \).

4. \( X_0 = n \), and \( \mathbb{E}[X_i \mid X_{i-1}] \leq \frac{3}{4} X_{i-1} + \frac{1}{2} X_{i-1} \leq \frac{7}{8} X_{i-1} \).

5. \( \forall \) random variables \( \mathbb{E}[X] = \mathbb{E}_y \mathbb{E}[X \mid Y = y] \).

6. \( \mathbb{E}[X_i] = \mathbb{E}_y \left[ \mathbb{E}[X_i \mid X_{i-1} = y] \right] \leq \mathbb{E}_{X_{i-1}=y} \left[ \frac{7}{8} y \right] = \frac{7}{8} \mathbb{E}[X_{i-1}] \leq \left( \frac{7}{8} \right)^i \mathbb{E}[X_0] = \left( \frac{7}{8} \right)^i n. \)
1. \( M = 8 \log_{8/7} n \): \( \mu = \mathbb{E}[X_M] \leq \left( \frac{7}{8} \right)^M n \leq \frac{1}{n^8} n = \frac{1}{n^7} \).

2. **Markov’s Inequality**: For a non-negative variable \( X \), and \( t > 0 \), we have:

\[
\Pr[X \geq t] \leq \frac{\mathbb{E}[X]}{t}.
\]

3. By Markov’s inequality:

\[
\Pr[t \text{ participates } > M \text{ recursive calls}] \leq \Pr[X_M \geq 1] \leq \frac{\mathbb{E}[X_M]}{1} \leq \frac{1}{n^7}.
\]

4. Probability any element of input participates \( > M \) recursive calls \( \leq n(1/n^7) \leq 1/n^6 \).
Part IV

Chernoff inequality
**Preliminaries**

1. **$X, Y$:** Random variables are *independent* if $\forall x, y$:

   $$\Pr[(X = x) \cap (Y = y)] = \Pr[X = x] \cdot \Pr[Y = y].$$

2. The following is easy to prove:

   **Claim**

   *If $X$ and $Y$ are independent*

   $$\implies E[XY] = E[X] E[Y].$$

   $$\implies Z = e^X \text{ and } W = e^Y \text{ are independent.}$$
**Theorem (Chernoff inequality)**

For independent random variables $X_1, \ldots, X_n$, such that $\Pr[X_i = 1] = \Pr[X_i = -1] = \frac{1}{2}$, for $i = 1, \ldots, n$. Let $Y = \sum_{i=1}^{n} X_i$. Then, for any $\Delta > 0$, we have

$$\Pr[Y \geq \Delta] \leq \exp\left(-\frac{\Delta^2}{2n}\right).$$
Proof of Chernoff inequality

Fix arbitrary $t > 0$:

$$\Pr[Y \geq \Delta] = \Pr[tY \geq t\Delta]$$
Proof of Chernoff inequality

Fix arbitrary $t > 0$:

$$\Pr[Y \geq \Delta] = \Pr[tY \geq t\Delta] = \Pr[\exp(tY) \geq \exp(t\Delta)]$$
Proof of Chernoff inequality

Fix arbitrary $t > 0$:

$$\Pr[Y \geq \Delta] = \Pr[tY \geq t\Delta] = \Pr[\exp(tY) \geq \exp(t\Delta)] \leq \frac{E[\exp(tY)]}{\exp(t\Delta)},$$
Proof of Chernoff inequality

Continued...

\[ E[\exp(tX_i)] = \frac{1}{2}e^t + \frac{1}{2}e^{-t}. \]
Proof of Chernoff inequality

Continued...

\[ E\left[ \exp(tX_i) \right] = \frac{1}{2} e^t + \frac{1}{2} e^{-t} = \frac{e^t + e^{-t}}{2}. \]
Proof of Chernoff inequality

Continued...

\[ E \left[ \exp(tX_i) \right] = \frac{e^t + e^{-t}}{2}. \]
Proof of Chernoff inequality

Continued...

\[
E \left[ \exp(tX_i) \right] = \frac{e^t + e^{-t}}{2}
\]

\[
= \frac{1}{2} \left( 1 + \frac{t}{1!} + \frac{t^2}{2!} + \frac{t^3}{3!} + \cdots \right)
\]

\[
+ \frac{1}{2} \left( 1 - \frac{t}{1!} + \frac{t^2}{2!} - \frac{t^3}{3!} + \cdots \right).
\]
Proof of Chernoff inequality

Continued...

\[
E\left[ \exp(tX_i) \right] = \frac{1}{2} \left( 1 + \frac{t}{1!} + \frac{t^2}{2!} + \frac{t^3}{3!} + \cdots \right) \\
+ \frac{1}{2} \left( 1 - \frac{t}{1!} + \frac{t^2}{2!} - \frac{t^3}{3!} + \cdots \right).
\]
Proof of Chernoff inequality

Continued...

\[
E\left[ \exp(tX_i) \right] = \frac{1}{2} \left( 1 + \frac{t}{1!} + \frac{t^2}{2!} + \frac{t^3}{3!} + \cdots \right) \\
+ \frac{1}{2} \left( 1 - \frac{t}{1!} + \frac{t^2}{2!} - \frac{t^3}{3!} + \cdots \right) \\
= 1 + \frac{t^2}{2!} + + \cdots + \frac{t^{2k}}{(2k)!} + \cdots .
\]
Proof of Chernoff inequality

Continued...

\[ E \left[ \exp(tX_i) \right] = 1 + \frac{t^2}{2!} + \cdots + \frac{t^{2k}}{(2k)!} + \cdots. \]
Proof of Chernoff inequality

Continued...

\[ E \left[ \exp(tX_i) \right] = 1 + \frac{t^2}{2!} + \cdots + \frac{t^{2k}}{(2k)!} + \cdots. \]

However: \((2k)! = k!(k + 1)(k + 2) \cdots 2k \geq k!2^k.\)
Proof of Chernoff inequality
Continued...

\[
E\left[ \exp(tX_i) \right] = 1 + \frac{t^2}{2!} + \cdots + \frac{t^{2k}}{(2k)!} + \cdots.
\]

However:

\[
(2k)! = k!(k + 1)(k + 2) \cdots 2k \geq k!2^k.
\]

\[
E\left[ \exp(tX_i) \right] = \sum_{i=0}^{\infty} \frac{t^{2i}}{(2i)!}
\]
Proof of Chernoff inequality

Continued...

\[
E\left[ \exp(tX_i) \right] = 1 + \frac{t^2}{2!} + + \cdots + \frac{t^{2k}}{(2k)!} + \cdots .
\]

However: 
\[
(2k)! = k!(k + 1)(k + 2) \cdots 2k \geq k!2^k.
\]

\[
E\left[ \exp(tX_i) \right] = \sum_{i=0}^{\infty} \frac{t^{2i}}{(2i)!} \leq \sum_{i=0}^{\infty} \frac{t^{2i}}{2^i(i!)}
\]
$\mathbb{E}[\exp(tX_i)] \leq \sum_{i=0}^{\infty} \frac{t^{2i}}{2^i (i!)}$
Proof of Chernoff inequality

Continued...

\[ E[\exp(tX_i)] \leq \sum_{i=0}^{\infty} \frac{t^{2i}}{2^i(i!)} = \sum_{i=0}^{\infty} \frac{1}{i!} \left( \frac{t^2}{2} \right)^i \]
Proof of Chernoff inequality

Continued...

\[
E\left[ \exp(tX_i) \right] \leq \sum_{i=0}^{\infty} \frac{1}{i!} \left( \frac{t^2}{2} \right)^i
\]
Proof of Chernoff inequality

Continued...

\[
\mathbb{E}[\exp(tX_i)] \leq \sum_{i=0}^{\infty} \frac{1}{i!} \left( \frac{t^2}{2} \right)^i = \exp\left(\frac{t^2}{2}\right).
\]
Proof of Chernoff inequality

Continued...

\[ \mathbb{E}[\exp(tX_i)] \leq \exp\left(\frac{t^2}{2}\right). \]
Proof of Chernoff inequality

Continued...

\[
\mathbb{E}[\exp(tX_i)] \leq \exp\left(\frac{t^2}{2}\right).
\]

\[
\mathbb{E}[\exp(t Y)] = \mathbb{E}\left[\exp\left(\sum_i tX_i\right)\right]
\]
Proof of Chernoff inequality

Continued...

$$E[\exp(tX_i)] \leq \exp\left(\frac{t^2}{2}\right).$$

$$E[\exp(tY)] = E\left[\exp\left(\sum_i tX_i\right)\right] = E\left[\prod_i \exp(tX_i)\right]$$
Proof of Chernoff inequality

Continued...

\[ E[\exp(tX_i)] \leq \exp\left(\frac{t^2}{2}\right). \]

\[ E[\exp(tY)] = E\left[\prod_i \exp(tX_i)\right] \]
Proof of Chernoff inequality

Continued...

\[ E[\exp(tX_i)] \leq \exp\left(\frac{t^2}{2}\right). \]

\[ E[\exp(tY)] = E\left[ \prod_i \exp(tX_i) \right] = \prod_{i=1}^{n} E[\exp(tX_i)] \]
Proof of Chernoff inequality

Continued...

\[ \mathbb{E}\left[ \exp(tX_i) \right] \leq \exp\left( \frac{t^2}{2} \right). \]

\[ \mathbb{E}\left[ \exp(tY) \right] = \prod_{i=1}^{n} \mathbb{E}\left[ \exp(tX_i) \right] \]
Proof of Chernoff inequality
Continued...

\[ E[\exp(tX_i)] \leq \exp\left(\frac{t^2}{2}\right). \]

\[ E[\exp(tY)] = \prod_{i=1}^{n} E[\exp(tX_i)] \leq \prod_{i=1}^{n} \exp\left(\frac{t^2}{2}\right) \]
Proof of Chernoff inequality

Continued...

\[ E[\exp(tX_i)] \leq \exp\left(\frac{t^2}{2}\right). \]

\[ E[\exp(tY)] \leq \prod_{i=1}^{n} \exp\left(\frac{t^2}{2}\right) \]
Proof of Chernoff inequality

Continued...

\[ \mathbb{E}[\exp(tX_i)] \leq \exp\left(\frac{t^2}{2}\right). \]

\[ \mathbb{E}[\exp(tY)] \leq \prod_{i=1}^{n} \exp\left(\frac{t^2}{2}\right) = \exp\left(\frac{nt^2}{2}\right). \]
Proof of Chernoff inequality

Continued...

\[
E\left[\exp(tY)\right] \leq \prod_{i=1}^{n} \exp\left(\frac{t^2}{2}\right) = \exp\left(\frac{nt^2}{2}\right).
\]
Proof of Chernoff inequality

Continued...

\[ E[\exp(tY)] \leq \exp\left(\frac{nt^2}{2}\right). \]
Proof of Chernoff inequality

Continued...

\[ E \left[ \exp(tY) \right] \leq \exp\left( \frac{nt^2}{2} \right). \]

\[ \Pr \left[ Y \geq \Delta \right] \]
Proof of Chernoff inequality

Continued...

\[ E\left[ \exp(tY) \right] \leq \exp\left( \frac{nt^2}{2} \right). \]

\[ \Pr\left[ Y \geq \Delta \right] \leq \frac{E\left[ \exp(tY) \right]}{\exp(t\Delta)} \]
Proof of Chernoff inequality

Continued...

\[ E[\exp(tY)] \leq \exp\left(\frac{nt^2}{2}\right). \]

\[ \Pr[Y \geq \Delta] \leq \frac{E[\exp(tY)]}{\exp(t\Delta)} \leq \frac{\exp\left(\frac{nt^2}{2}\right)}{\exp(t\Delta)} \]
Proof of Chernoff inequality

Continued...

\[ E\left[ \exp(tY) \right] \leq \exp\left( \frac{nt^2}{2} \right). \]

\[ \Pr\left[ Y \geq \Delta \right] \leq \frac{\exp\left( \frac{nt^2}{2} \right)}{\exp(t\Delta)}. \]
Proof of Chernoff inequality

Continued...

\[ E[\exp(tY)] \leq \exp\left(\frac{nt^2}{2}\right). \]

\[ \Pr[Y \geq \Delta] \leq \frac{\exp\left(\frac{nt^2}{2}\right)}{\exp(t\Delta)} = \exp\left(\frac{nt^2}{2} - t\Delta\right). \]
Proof of Chernoff inequality

Continued...

\[ E[\exp(tY)] \leq \exp\left(\frac{nt^2}{2}\right). \]

\[ \Pr[Y \geq \Delta] = \exp\left(\frac{nt^2}{2} - t\Delta\right). \]
Proof of Chernoff inequality

Continued...

\[
\Pr[Y \geq \Delta] = \exp \left( \frac{nt^2}{2} - t\Delta \right).
\]

Set \( t = \Delta/n \):
Proof of Chernoff inequality

Continued...

\[ Pr[Y \geq \Delta] = \exp\left(\frac{nt^2}{2} - t\Delta\right). \]

Set \( t = \Delta/n \):

\[ Pr[Y \geq \Delta] \leq \exp\left(\frac{n}{2} \left(\frac{\Delta}{n}\right)^2 - \frac{\Delta}{n}\Delta\right) = \exp\left(-\frac{\Delta^2}{2n}\right). \]
Proof of Chernoff inequality
Continued...

\[
\Pr[ Y \geq \Delta ] = \exp\left( \frac{nt^2}{2} - t\Delta \right).
\]

Set \( t = \Delta/n \):

\[
\Pr[ Y \geq \Delta ] \leq \exp\left( \frac{n}{2} \left( \frac{\Delta}{n} \right)^2 - \frac{\Delta}{n} \Delta \right) = \exp\left( -\frac{\Delta^2}{2n} \right).
\]
Chernoff inequality...

...what it really says

By theorem:

$$\Pr[Y \geq \Delta] = \sum_{i=\Delta}^{n} \Pr[Y = i] = \sum_{i=n/2+\Delta/2}^{n} \frac{n!}{i!(n-i)!} \leq \exp\left(-\frac{\Delta^2}{2n}\right),$$
Corollary

Let $X_1, \ldots, X_n$ be $n$ independent random variables, such that $\Pr[X_i = 1] = \Pr[X_i = -1] = \frac{1}{2}$, for $i = 1, \ldots, n$. Let $Y = \sum_{i=1}^{n} X_i$. Then, for any $\Delta > 0$, we have

$$\Pr[|Y| \geq \Delta] \leq 2 \exp\left(-\frac{\Delta^2}{2n}\right).$$
$X_1, \ldots, X_n$ be $n$ independent coin flips, such that $\Pr[X_i = 1] = \Pr[X_i = 0] = \frac{1}{2}$, for $i = 1, \ldots, n$. Let $Y = \sum_{i=1}^{n} X_i$. Then, for any $\Delta > 0$, we have

$$\Pr\left[ \frac{n}{2} - Y \geq \Delta \right] \leq \exp\left(-\frac{2\Delta^2}{n}\right) \quad \text{and} \quad \Pr\left[ Y - \frac{n}{2} \geq \Delta \right] \leq \exp\left(-\frac{2\Delta^2}{n}\right).$$

In particular, we have $\Pr\left[ \left| Y - \frac{n}{2} \right| \geq \Delta \right] \leq 2 \exp\left(-\frac{2\Delta^2}{n}\right)$. 
The special case we needed

Lemma

In a sequence of $M$ coin flips, the probability that the number of ones is smaller than $L \leq M/4$ is at most $\exp(-M/8)$.

Proof.

Let $Y = \sum_{i=1}^{m} X_i$ the sum of the $M$ coin flips. By the above corollary, we have:

$$\Pr[Y \leq L] = \Pr\left[\frac{M}{2} - Y \geq \frac{M}{2} - L\right] = \Pr\left[\frac{M}{2} - Y \geq \Delta\right],$$

where $\Delta = M/2 - L \geq M/4$. Using the above Chernoff inequality, we get

$$\Pr[Y \leq L] \leq \exp\left(-\frac{2\Delta^2}{M}\right) \leq \exp(-M/8).$$
Part V

The Chernoff Bound — General Case
The Chernoff Bound

The general problem

Problem

Let $X_1, \ldots, X_n$ be $n$ independent Bernoulli trials, where

$$\Pr[X_i = 1] = p_i \quad \text{and} \quad \Pr[X_i = 0] = 1 - p_i,$$

and let denote

$$Y = \sum_{i} X_i \quad \mu = \mathbb{E}[Y].$$

**Question:** what is the probability that $Y \geq (1 + \delta)\mu.$
The Chernoff Bound

The general case

Theorem (Chernoff inequality)

For any $\delta > 0$,

$$\Pr[Y > (1 + \delta)\mu] < \left( \frac{e^\delta}{(1 + \delta)^{1+\delta}} \right)^\mu.$$

Or in a more simplified form, for any $\delta \leq 2e - 1$,

$$\Pr[Y > (1 + \delta)\mu] < \exp\left( -\mu\delta^2 / 4 \right),$$

and

$$\Pr[Y > (1 + \delta)\mu] < 2^{-\mu(1+\delta)},$$

for $\delta \geq 2e - 1$. 

Sariel (UIUC)
CS573
Fall 2013
Under the same assumptions as the theorem above, we have

\[
\Pr\left[ Y < (1 - \delta) \mu \right] \leq \exp\left( -\mu \frac{\delta^2}{2} \right).
\]
Part VI

Treaps
Balanced binary search trees...

1. Work usually by storing additional information.

2. Idea: For every element $x$ inserted randomly choose priority $p(x) \in [0, 1]$.

3. $X = \{x_1, \ldots, x_n\}$
   priorities: $p(x_1), \ldots, p(x_n)$.

4. $x_k$: lowest priority in $X$.

5. Make $x_k$ the root.

6. partition $X$ in the natural way:
   (A) $L$: set of all the numbers smaller than $x_k$ in $X$, and
   (B) $R$: set of all the numbers larger than $x_k$ in $X$. 

Sariel (UIUC)  |  CS573 | Fall 2013
--- | --- | ---
32 | 32 | 44
Balanced binary search trees...

1. Work usually by storing additional information.

2. Idea: For every element $x$ inserted randomly choose **priority** $p(x) \in [0,1]$.

3. $X = \{x_1, \ldots, x_n\}$
   
   priorities: $p(x_1), \ldots, p(x_n)$.

4. $x_k$: lowest priority in $X$.

5. Make $x_k$ the root.

6. partition $X$ in the natural way:
   
   (A) $L$: set of all the numbers smaller than $x_k$ in $X$, and
   
   (B) $R$: set of all the numbers larger than $x_k$ in $X$. 

Sariel (UIUC)
Balanced binary search trees...

1. Work usually by storing additional information.

2. Idea: For every element $x$ inserted randomly choose priority $p(x) \in [0, 1]$. 

3. $X = \{x_1, \ldots, x_n\}$
   priorities: $p(x_1), \ldots, p(x_n)$.

4. $x_k$: lowest priority in $X$.

5. Make $x_k$ the root.

6. Partition $X$ in the natural way:
   (A) $L$: set of all the numbers smaller than $x_k$ in $X$, and
   (B) $R$: set of all the numbers larger than $x_k$ in $X$. 
Balanced binary search trees...

1. Work usually by storing additional information.
2. Idea: For every element $x$ inserted randomly choose \textit{priority} $p(x) \in [0, 1]$.
3. $X = \{x_1, \ldots, x_n\}$
   priorities: $p(x_1), \ldots, p(x_n)$.
4. $x_k$: lowest priority in $X$.
5. Make $x_k$ the root.
6. partition $X$ in the natural way:
   (A) $L$: set of all the numbers smaller than $x_k$ in $X$, and
   (B) $R$: set of all the numbers larger than $x_k$ in $X$. 
Balanced binary search trees...

1. Work usually by storing additional information.

2. Idea: For every element $x$ inserted randomly choose priority $p(x) \in [0, 1]$.

3. $X = \{x_1, \ldots, x_n\}$ priorities: $p(x_1), \ldots, p(x_n)$.

4. $x_k$: lowest priority in $X$.

5. Make $x_k$ the root.

6. partition $X$ in the natural way:
   (A) $L$: set of all the numbers smaller than $x_k$ in $X$, and
   (B) $R$: set of all the numbers larger than $x_k$ in $X$. 
Balanced binary search trees...

1. Work usually by storing additional information.

2. Idea: For every element $x$ inserted randomly choose priority $p(x) \in [0, 1]$.

3. $X = \{x_1, \ldots, x_n\}$
   priorities: $p(x_1), \ldots, p(x_n)$.

4. $x_k$: lowest priority in $X$.

5. Make $x_k$ the root.

6. Partition $X$ in the natural way:
   (A) $L$: set of all the numbers smaller than $x_k$ in $X$, and
   (B) $R$: set of all the numbers larger than $x_k$ in $X$. 
Continuing recursively, we have:

(A) \( L \): set of all the numbers smaller than \( x_k \) in \( X \), and

(B) \( R \): set of all the numbers larger than \( x_k \) in \( X \).

**Definition**

Resulting tree a *treap*. Tree over the elements, and a heap over the priorities; that is, TREAP = TREE + HEAP.
Continuing recursively, we have:

(A) \( L \): set of all the numbers smaller than \( x_k \) in \( X \), and

(B) \( R \): set of all the numbers larger than \( x_k \) in \( X \).

Definition

Resulting tree a \textit{treap}.

Tree over the elements, and a heap over the priorities; that is,

\[
\text{TREAP} = \text{TREE} + \text{HEAP}.
\]
Lemma

$S$: $n$ elements.

Expected depth of treap $T$ for $S$ is $O(\log(n))$.

Depth of treap $T$ for $S$ is $O(\log(n))$ w.h.p.

Proof.

QuickSort...
**Lemma**

\( S: n \) elements.

*Expected depth of treap \( T \) for \( S \) is \( O(\log(n)) \).*

*Depth of treap \( T \) for \( S \) is \( O(\log(n)) \) w.h.p.*

**Proof.**

*QuickSort...*
Observation

Given $n$ distinct elements, and their (distinct) priorities, the treap storing them is uniquely defined.
Rotate right...
**Treaps – insertion**

1. \(x\): an element \(x\) to insert.
2. Insert it into \(T\) as a regular binary tree.
3. Takes \(O(\text{height}(T))\).
4. \(x\) is a leaf in the treap.
5. Pick priority \(p(x) \in [0, 1]\).
6. Valid search tree,.. but priority heap is broken at \(x\).
7. Fix priority heap around \(x\).
Treaps – insertion

1. $x$: an element $x$ to insert.
2. Insert it into $T$ as a regular binary tree.
3. Takes $O(\text{height}(T))$.
4. $x$ is a leaf in the treap.
5. Pick priority $p(x) \in [0, 1]$.
6. Valid search tree, but priority heap is broken at $x$.
7. Fix priority heap around $x$. 
Treaps – insertion

1. $x$: an element $x$ to insert.
2. Insert it into $T$ as a regular binary tree.
3. Takes $O(\text{height}(T))$.
4. $x$ is a leaf in the treap.
5. Pick priority $p(x) \in [0, 1]$.
6. Valid search tree,.. but priority heap is broken at $x$.
7. Fix priority heap around $x$. 
**Treaps – insertion**

1. $x$: an element $x$ to insert.
2. Insert it into $T$ as a regular binary tree.
3. Takes $O(\text{height}(T))$.
4. $x$ is a leaf in the treap.
5. Pick priority $p(x) \in [0, 1]$.
6. Valid search tree, but priority heap is broken at $x$.
7. Fix priority heap around $x$. 

**Treaps – insertion**

1. \( x \): an element \( x \) to insert.
2. Insert it into \( T \) as a regular binary tree.
3. Takes \( O(\text{height}(T)) \).
4. \( x \) is a leaf in the treap.
5. Pick priority \( p(x) \in [0, 1] \).
6. Valid search tree,.. but priority heap is broken at \( x \).
7. Fix priority heap around \( x \).
Treaps – insertion

1. $x$: an element $x$ to insert.
2. Insert it into $T$ as a regular binary tree.
3. Takes $O(\text{height}(T))$.
4. $x$ is a leaf in the treap.
5. Pick priority $p(x) \in [0, 1]$.
6. Valid search tree, but priority heap is broken at $x$.
7. Fix priority heap around $x$. 
Treaps – insertion

1. \( x \): an element \( x \) to insert.
2. Insert it into \( T \) as a regular binary tree.
3. Takes \( O(\text{height}(T)) \).
4. \( x \) is a leaf in the treap.
5. Pick priority \( p(x) \in [0, 1] \).
6. Valid search tree,.. but priority heap is broken at \( x \).
7. Fix priority heap around \( x \).
Fix treap for a leaf $x$...

\textbf{RotateUp}(x)

\begin{align*}
y & \leftarrow \text{parent}(x) \\
\text{while } p(y) > p(x) & \text{ do} \\
& \quad \text{if } y.\text{left\_child} = x \text{ then} \\
& \qquad \text{RotateRight}(y) \\
& \quad \text{else} \\
& \qquad \text{RotateLeft}(y) \\
& \quad y \leftarrow \text{parent}(x)
\end{align*}

Insertion takes $O(\text{height}(T))$. 
Fix treap for a leaf \( x \)...

\[
\textbf{RotateUp}(x) \\
y \leftarrow \text{parent}(x) \\
\text{while } p(y) > p(x) \text{ do} \\
\quad \text{if } y.\text{left}_\text{child} = x \text{ then} \\
\quad \quad \textbf{RotateRight}(y) \\
\quad \text{else} \\
\quad \quad \textbf{RotateLeft}(y) \\
\quad y \leftarrow \text{parent}(x)
\]

Insertion takes \( O(\text{height}(T)) \).
Treaps – deletion

1. Deletion is just an insertion done in reverse.
2. \( x \): element to delete.
3. Set \( p(x) \leftarrow +\infty \),
4. rotate \( x \) down till its a leaf.
5. Rotate so that child with lower priority becomes new parent.
6. \( x \) is now leaf – deleting is easy...
Treaps – deletion

1. Deletion is just an insertion done in reverse.
2. \( x \): element to delete.
3. Set \( p(x) \leftarrow +\infty \),
4. rotate \( x \) down till its a leaf.
5. Rotate so that child with lower priority becomes new parent.
6. \( x \) is now leaf – deleting is easy...
Deletion is just an insertion done in reverse.

$x$: element to delete.

Set $p(x) \leftarrow +\infty$.

Rotate $x$ down till its a leaf.

Rotate so that child with lower priority becomes new parent.

$x$ is now leaf – deleting is easy...
Treaps – deletion

1. Deletion is just an insertion done in reverse.
2. $x$: element to delete.
3. Set $p(x) \leftarrow +\infty$,
4. rotate $x$ down till its a leaf.
5. Rotate so that child with lower priority becomes new parent.
6. $x$ is now leaf – deleting is easy...
Deletion is just an insertion done in reverse.

\( x \): element to delete.

Set \( p(x) \leftarrow +\infty \),

rotate \( x \) down till its a leaf.

Rotate so that child with lower priority becomes new parent.

\( x \) is now leaf – deleting is easy...
Split

1. $x$: element stored in treap $T$.
2. split $T$ into two treaps – one treap $T_{\leq x}$ and treap $T_{> x}$ for all the elements larger than $x$.
3. Set $p(x) \leftarrow -\infty$,
4. fix priorities by rotation.
5. $x$ item is now the root.
6. Splitting is now easy....
7. Restore $x$ to its original priority. Fix by rotations.
Split

1. $x$: element stored in treap $T$.
2. split $T$ into two treaps – one treap $T_{\leq x}$ and treap $T_{>x}$ for all the elements larger than $x$.
3. Set $p(x) \leftarrow -\infty$,
4. fix priorities by rotation.
5. $x$ item is now the root.
6. Splitting is now easy....
7. Restore $x$ to its original priority. Fix by rotations.
Split

1. \( x \): element stored in treap \( T \).
2. split \( T \) into two treaps – one treap \( T_{\leq x} \) and treap \( T_{>x} \) for all the elements larger than \( x \).
3. Set \( p(x) \leftarrow -\infty \),
4. fix priorities by rotation.
5. \( x \) item is now the root.
6. Splitting is now easy....
7. Restore \( x \) to its original priority. Fix by rotations.
Split

1. $x$: element stored in treap $T$.
2. split $T$ into two treaps – one treap $T_{\leq x}$ and treap $T_{>}$ for all the elements larger than $x$.
3. Set $p(x) \leftarrow -\infty$,
4. fix priorities by rotation.
5. $x$ item is now the root.
6. Splitting is now easy....
7. Restore $x$ to its original priority. Fix by rotations.
**Split**

1. $x$: element stored in treap $T$.
2. split $T$ into two treaps – one treap $T_{\leq x}$ and treap $T_{>}$ for all the elements larger than $x$.
3. Set $p(x) \leftarrow -\infty$,
4. fix priorities by rotation.
5. $x$ item is now the root.
6. Splitting is now easy....
7. Restore $x$ to its original priority. Fix by rotations.
Split

1. $x$: element stored in treap $T$.
2. split $T$ into two treaps – one treap $T_{\leq x}$ and treap $T_{>}$ for all the elements larger than $x$.
3. Set $p(x) \leftarrow -\infty$,
4. fix priorities by rotation.
5. $x$ item is now the root.
6. Splitting is now easy....
7. Restore $x$ to its original priority. Fix by rotations.
Meld

1. $T_L$ and $T_R$: treaps.
2. all elements in $T_L \cup$ all elements in $T_R$.
3. Want to merge them into a single treap...
Theorem

Let $T$ be an empty treap, after a sequence of $m = n^c$ insertions, where $c$ is some constant.

d: arbitrary constant.

The probability depth $T$ ever exceed $d \log n$ is $\leq 1/n^{O(1)}$.

A treap can handle insertion/deletion in $O(\log n)$ time with high probability.
Proof.

1. $T_1, \ldots, T_m$: sequence of treaps.
2. $T_i$ is treap after $i$th operation.
3. $\alpha_i = \Pr[\text{depth}(T_i) > tc' \log n] = \Pr[\text{depth}(T_i) > c't\left(\frac{\log n}{\log |T_i|}\right) \cdot \log |T_i|] \leq \frac{1}{n^{o(1)}}$,
4. Use union bound...
Proof

Proof.

1. $T_1, \ldots, T_m$: sequence of treaps.
2. $T_i$ is treap after $i$th operation.
3. $\alpha_i = \Pr[\text{depth}(T_i) > tc' \log n] = \Pr[\text{depth}(T_i) > c't \left(\frac{\log n}{\log |T_i|}\right) \cdot \log |T_i|] \leq \frac{1}{n^{o(1)}}$,
4. Use union bound...
Proof.

1. $T_1, \ldots, T_m$: sequence of treaps.
2. $T_i$ is treap after $i$th operation.
3. $\alpha_i = \Pr[\text{depth}(T_i) > tc' \log n] = \Pr[\text{depth}(T_i) > c't\left(\frac{\log n}{\log |T_i|}\right) \cdot \log |T_i|] \leq \frac{1}{n^{O(1)}},$
4. Use union bound...
Proof

Proof.

1. $T_1, \ldots, T_m$: sequence of treaps.
2. $T_i$ is treap after $i$th operation.
3. $\alpha_i = \Pr[\text{depth}(T_i) > tc' \log n] = \Pr[\text{depth}(T_i) > c't\left(\frac{\log n}{\log |T_i|}\right) \cdot \log |T_i|] \leq \frac{1}{n^{O(1)}}$.
4. Use union bound...
Chernoff inequality was a rediscovery of Bernstein inequality.

...published in 1924 by Sergei Bernstein.

Treaps were invented by Siedel and Aragon [1996].

Experimental evidence suggests that Treaps performs reasonably well in practice see Cho and Sahni [2000].

Old implementation of treaps I wrote in C is available here:
http://valis.cs.uiuc.edu/blog/?p=6060.