Randomized Algorithms II – High Probability

Lecture 10

September 25, 2014

Part I

Movie...

Part II

Understanding the binomial distribution

Binomial distribution

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

Claim

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

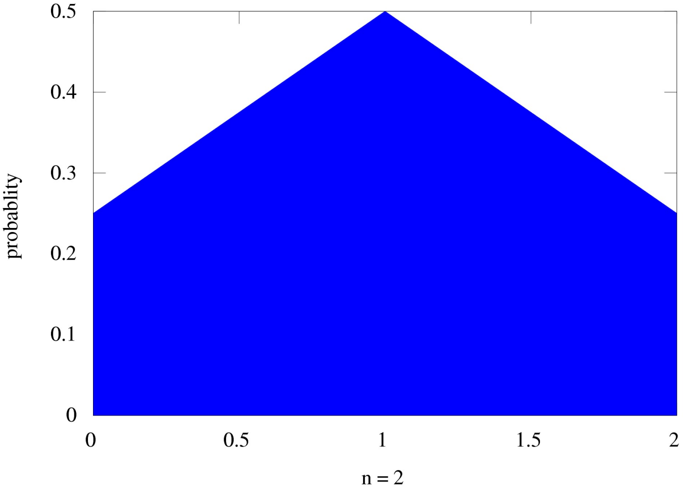
Where: $\binom{n}{k} = \frac{n!}{(n-k)!k!}$.

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 $0100010\ldots$) had probability $1/2^n$.

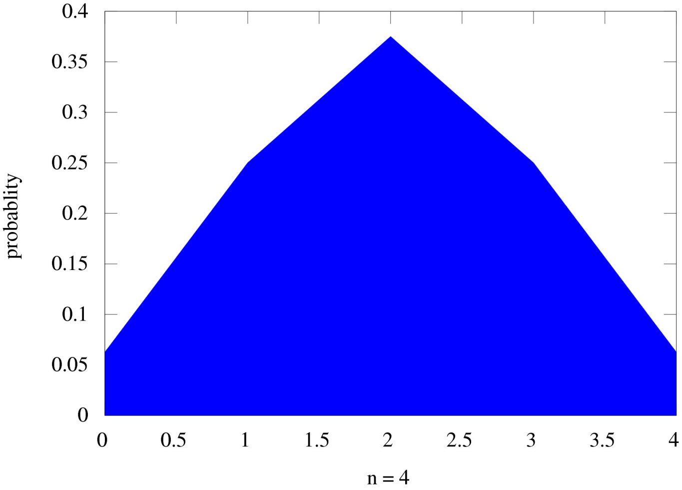
Massive randomness.. Is not that random.

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



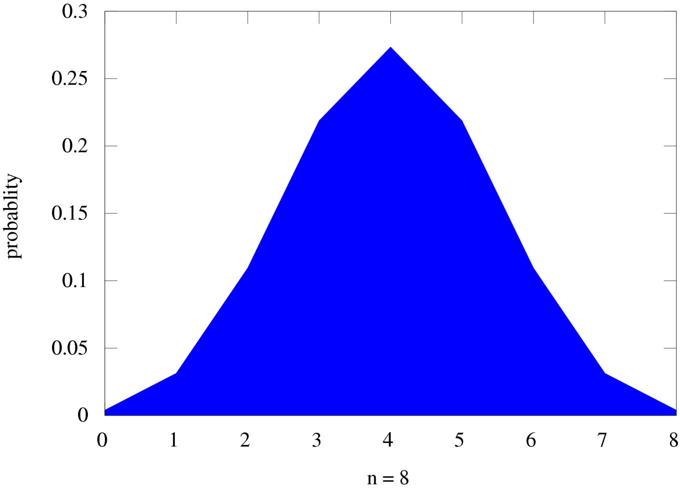
Massive randomness.. Is not that random.

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



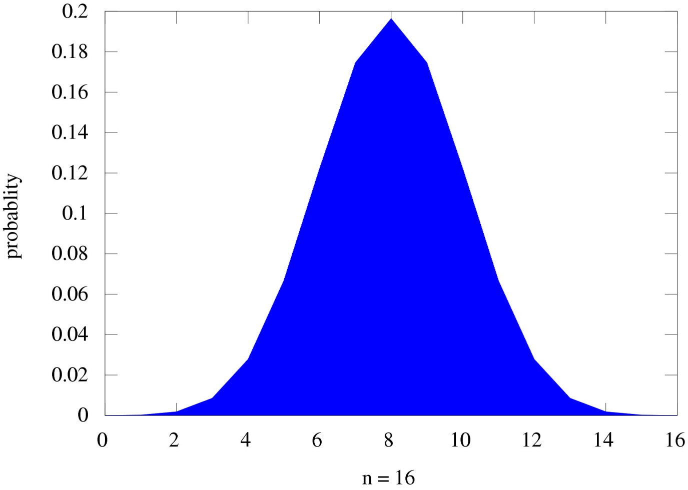
Massive randomness.. Is not that random.

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



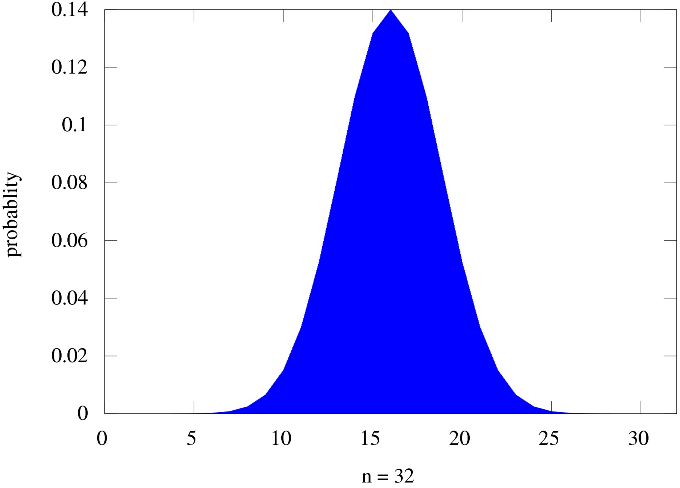
Massive randomness.. Is not that random.

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



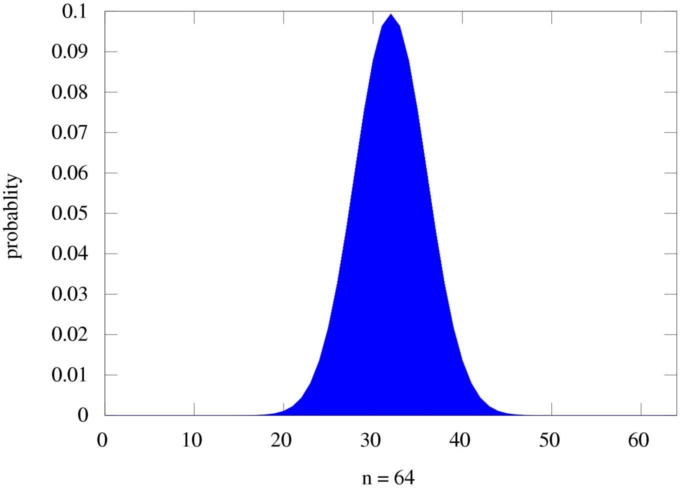
Massive randomness.. Is not that random.

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



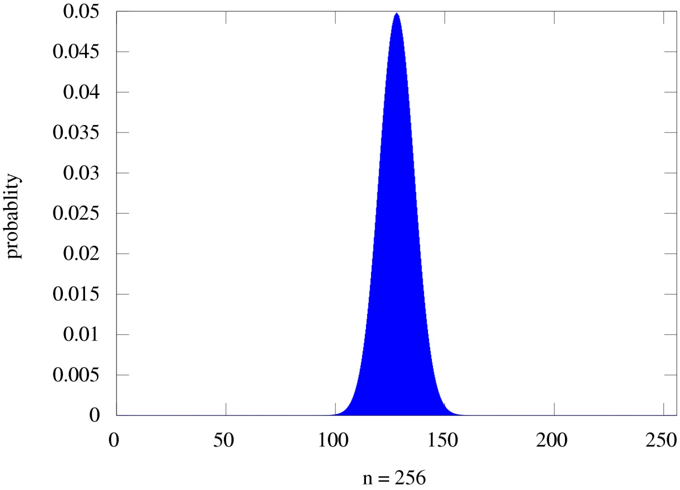
Massive randomness.. Is not that random.

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



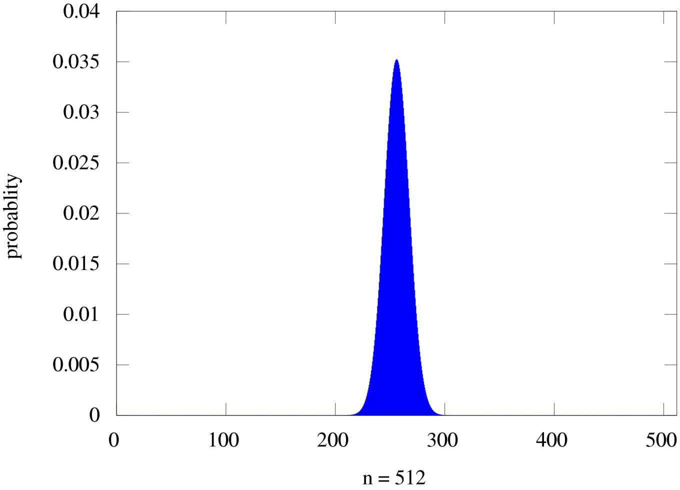
Massive randomness.. Is not that random.

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



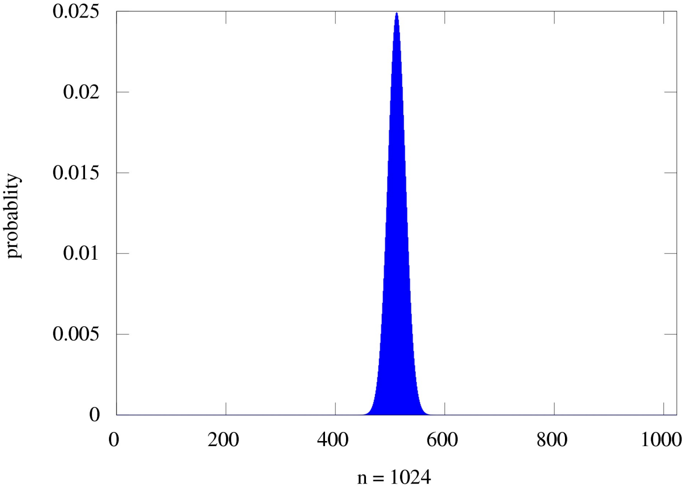
Massive randomness.. Is not that random.

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



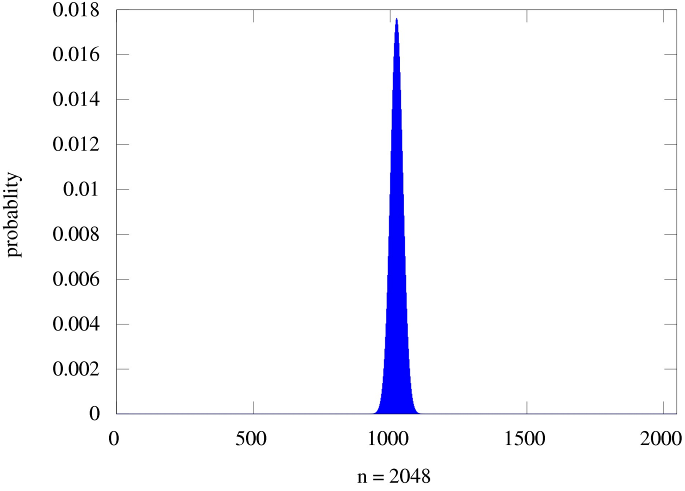
Massive randomness.. Is not that random.

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



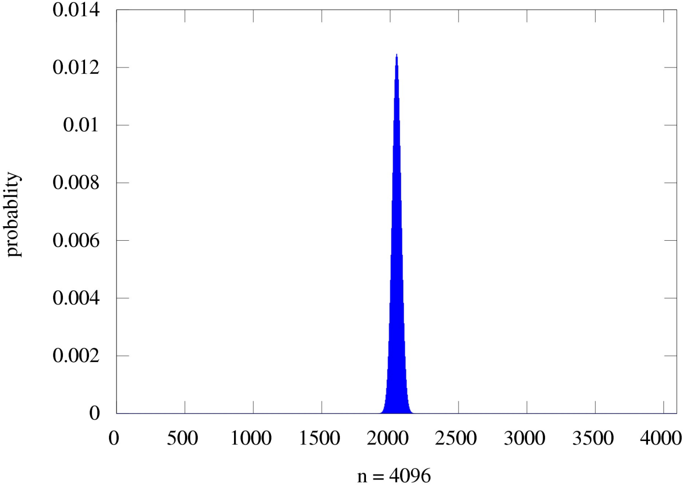
Massive randomness.. Is not that random.

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



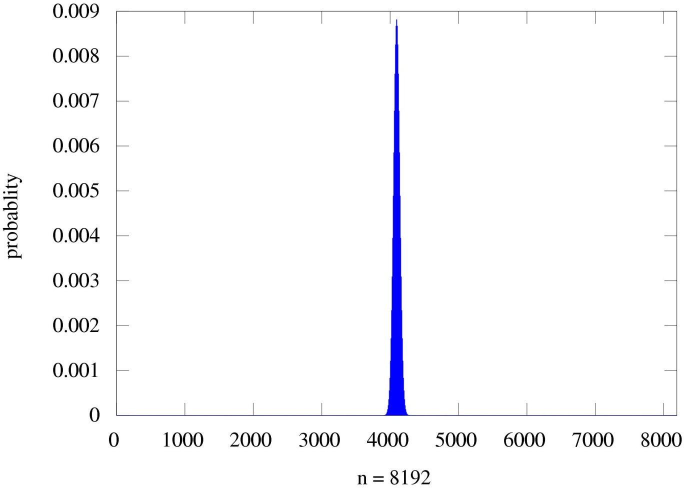
Massive randomness.. Is not that random.

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

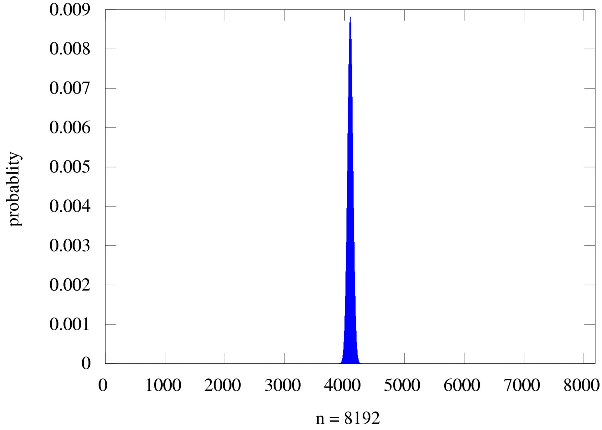


Massive randomness.. Is not that random.

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.



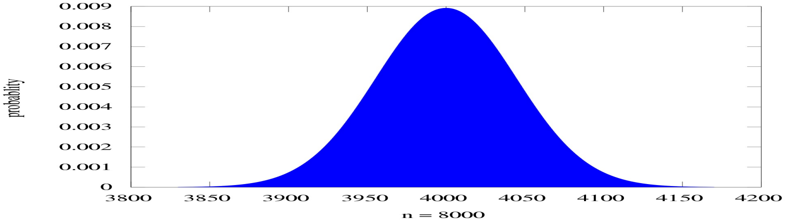
This is known as ***concentration of mass***.
This is a very special case of the ***law of large numbers***.

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.

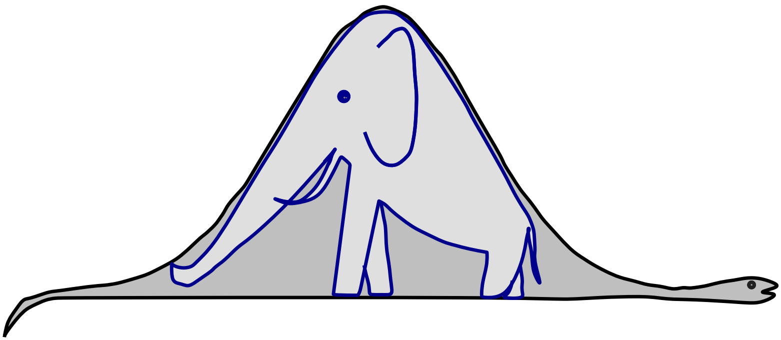


Massive randomness.. Is not that random.

Intuitive conclusion

Randomized algorithm are unpredictable in the tactical level, but very predictable in the strategic level.

What is really hiding below the Normal distribution?



Taken from **?**.

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 ith 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 $\alpha = \Pr\left[C_{QS} \geq 32n \ln n\right] \leq \Pr[\cup_i \mathcal{E}_i] \leq \sum_{i=1}^{n} \Pr\left[\mathcal{E}_i\right]$.
   ... by the union bound.

Show that **QuickSort** running time is $\mathbf{O(n \log n)}$

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 ith 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 $\alpha = \Pr\left[C_{QS} \geq 32 n \ln n\right] \leq \Pr[\cup_i \mathcal{E}_i] \leq \sum_{i=1}^{n} \Pr\left[\mathcal{E}_i\right]$.
   ... by the union bound.

Show that **QuickSort** running time is $\mathbf{O(n \log n)}$

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 ith 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 $\alpha = \Pr\left[C_{QS} \geq 32 n \ln n\right] \leq \Pr[\cup_i \mathcal{E}_i] \leq \sum_{i=1}^{n} \Pr\left[\mathcal{E}_i\right]$.
   ... by the union bound.

Show that **QuickSort** running time is $\mathbf{O(n \log n)}$

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 ith 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 $\alpha = \Pr\left[C_{QS} \geq 32 n \ln n\right] \leq \Pr[\cup_i \mathcal{E}_i] \leq \sum_{i=1}^{n} \Pr\left[\mathcal{E}_i\right]$.
   ... 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 ith 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 $\alpha = \Pr\left[C_{QS} \geq 32n \ln n\right] \leq \Pr[\cup_i \mathcal{E}_i] \leq \sum_{i=1}^{n} \Pr\left[\mathcal{E}_i\right]$.
... by the union bound.

Show that **QuickSort** running time is $\mathbf{O(n \log n)}$

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

   ... by the union bound.

Show that QuickSort running time is $\mathbf{O(n \log n)}$

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

Show that **QuickSort** running time is $\mathbf{O(n \log n)}$

1. Probability of failure is
$\alpha = \Pr\left[C_{QS} \geq 32n \ln n\right] \leq \Pr[\cup_i \mathcal{E}_i] \leq \sum_{i=1}^{n} \Pr\left[\mathcal{E}_i\right]$.
2. ***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... $\alpha \leq \sum_{i=1}^{n} \Pr\left[\mathcal{E}_i\right] \leq \sum_{i=1}^{n} \frac{1}{n^3} = \frac{1}{n^2}$.
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 **QuickSort** running time is $\mathbf{O(n \log n)}$

1. Probability of failure is
$\alpha = \Pr\left[C_{QS} \geq 32n \ln n\right] \leq \Pr[\bigcup_i \mathcal{E}_i] \leq \sum_{i=1}^{n} \Pr\left[\mathcal{E}_i\right]$.
2. ***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... $\alpha \leq \sum_{i=1}^{n} \Pr\left[\mathcal{E}_i\right] \leq \sum_{i=1}^{n} \frac{1}{n^3} = \frac{1}{n^2}$.
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 **QuickSort** running time is $\mathbf{O(n \log n)}$

1. Probability of failure is $\alpha = \Pr\left[C_{QS} \geq 32n \ln n\right] \leq \Pr[\bigcup_i \mathcal{E}_i] \leq \sum_{i=1}^{n} \Pr\left[\mathcal{E}_i\right]$.
2. ***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... $\alpha \leq \sum_{i=1}^{n} \Pr\left[\mathcal{E}_i\right] \leq \sum_{i=1}^{n} \frac{1}{n^3} = \frac{1}{n^2}$.
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 QuickSort running time is $\mathbf{O(n \log n)}$

1. Probability of failure is $\alpha = \Pr\left[C_{QS} \geq 32n \ln n\right] \leq \Pr[\cup_i \mathcal{E}_i] \leq \sum_{i=1}^{n} \Pr\left[\mathcal{E}_i\right]$.
2. ***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... $\alpha \leq \sum_{i=1}^{n} \Pr\left[\mathcal{E}_i\right] \leq \sum_{i=1}^{n} \frac{1}{n^3} = \frac{1}{n^2}$.
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 **QuickSort** running time is $\mathbf{O(n \log n)}$

1. Probability of failure is
$\alpha = \Pr\left[C_{QS} \geq 32n \ln n\right] \leq \Pr[\bigcup_i \mathcal{E}_i] \leq \sum_{i=1}^{n} \Pr\left[\mathcal{E}_i\right]$.
2. ***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... $\alpha \leq \sum_{i=1}^{n} \Pr\left[\mathcal{E}_i\right] \leq \sum_{i=1}^{n} \frac{1}{n^3} = \frac{1}{n^2}$.
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 **QuickSort**.
2. x: Arbitrary element x in input.
3. S_1: Input.
4. S_i: input to ith level recursive call that include x.
5. x ***lucky*** in jth iteration, if balanced split...
   $|S_{j+1}| \leq (3/4)|S_j|$ and $|S_j \setminus S_{j+1}| \leq (3/4)|S_j|$
6. $Y_j = 1 \iff x$ lucky in jth iteration.
7. $\Pr\left[Y_j\right] = \frac{1}{2}$.
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 $|S_j| \leq n(3/4)^{\#\text{ of lucky iteration in } 1\ldots j}$.
11. If ρ 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. n: number of elements in input for **QuickSort**.
2. x: Arbitrary element x in input.
3. S_1: Input.
4. S_i: input to ith level recursive call that include x.
5. x ***lucky*** in jth iteration, if balanced split...
   $|S_{j+1}| \leq (3/4)|S_j|$ and $|S_j \setminus S_{j+1}| \leq (3/4)|S_j|$
6. $Y_j = 1 \iff x$ lucky in jth iteration.
7. $\Pr\left[Y_j\right] = \frac{1}{2}$.
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 $|S_j| \leq n(3/4)^{\#\text{ of lucky iteration in} 1\ldots j}$.
11. If ρ 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. n: number of elements in input for **QuickSort**.
2. x: Arbitrary element x in input.
3. S_1: Input.
4. S_i: input to ith level recursive call that include x.
5. x ***lucky*** in jth iteration, if balanced split...
   $|S_{j+1}| \leq (3/4)\,|S_j|$ and $|S_j \setminus S_{j+1}| \leq (3/4)\,|S_j|$
6. $Y_j = 1 \iff x$ lucky in jth iteration.
7. $\Pr\left[Y_j \right] = \frac{1}{2}$.
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 $|S_j| \leq n(3/4)^{\#\text{ of lucky iteration in}1\ldots j}$.
11. If ρ 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. n: number of elements in input for **QuickSort**.
2. x: Arbitrary element x in input.
3. S_1: Input.
4. S_i: input to ith level recursive call that include x.
5. x ***lucky*** in jth iteration, if balanced split...
   $|S_{j+1}| \leq (3/4)|S_j|$ and $|S_j \setminus S_{j+1}| \leq (3/4)|S_j|$
6. $Y_j = 1 \iff x$ lucky in jth iteration.
7. $\Pr\left[Y_j\right] = \frac{1}{2}$.
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 $|S_j| \leq n(3/4)^{\#\text{ of lucky iteration in}1\ldots j}$.
11. If ρ 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 ρ heads?
3. A: In expectation, 2ρ 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 ρ heads?
3. A: In expectation, 2ρ 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 ρ heads?
3. A: In expectation, 2ρ times.

Proving that an element...

... participates in small number of rounds.

1. Assume the following:

Lemma

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

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** $\leq 1/n^3$.

Proving that an element...

... participates in small number of rounds.

1. Assume the following:

Lemma

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

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** $\leq 1/n^3$.

Proving that an element...

... participates in small number of rounds.

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

Proving that an element...

... participates in small number of rounds.

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

Proving that an element...

... participates in small number of rounds.

1. n input elements. Probability depth of recursion in **QuickSort** $> 32 \ln n$ is $\leq (1/n^3) * n = 1/n^2$.
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 ith level of recursion containing t.
4. $X_0 = n$, and $\mathbf{E}\left[X_i \,\middle|\, X_{i-1}\right] \leq \frac{1}{2}\frac{3}{4}X_{i-1} + \frac{1}{2}X_{i-1} \leq \frac{7}{8}X_{i-1}$.
5. $\forall$ random variables $\mathbf{E}\left[X\right] = \mathbf{E}_y\left[\mathbf{E}\left[X \,\middle|\, Y = y\right]\right]$.
6. $\mathbf{E}\left[X_i\right] = \mathbf{E}_y\left[\mathbf{E}\left[X_i \,\middle|\, X_{i-1} = y\right]\right] \leq \mathbf{E}_{X_{i-1}=y}\left[\frac{7}{8}y\right] = \frac{7}{8}\mathbf{E}\left[X_{i-1}\right] \leq \left(\frac{7}{8}\right)^i \mathbf{E}[X_0] = \left(\frac{7}{8}\right)^i n$.

Alternative proof of high probability of **QuickSort**

1. $M = 8\log_{8/7} n$: $\mu = \mathbf{E}\left[X_M\right] \leq \left(\frac{7}{8}\right)^M n \leq \frac{1}{n^8} n = \frac{1}{n^7}$.
2. Markov's Inequality: For a non-negative variable X, and $t > 0$, we have:

$$\Pr\left[X \geq t\right] \quad \leq \quad \frac{\mathbf{E}[X]}{t}.$$

3. By Markov's inequality:

$$\Pr\left[\begin{array}{c} t \text{ participates} \\ > M \text{ recursive calls} \end{array}\right] \leq \Pr\left[X_M \geq 1\right] \leq \frac{\mathbf{E}[X_M]}{1} \leq \frac{1}{n^7}.$$

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 $\forall x, y$:

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

2. The following is easy to prove:

Claim

If X and Y are independent
$\implies \mathbf{E}[XY] = \mathbf{E}[X]\,\mathbf{E}[Y]$.
$\implies Z = e^X$ *and* $W = e^Y$ *are independent.*

Chernoff inequality

Theorem (Chernoff inequality)

$X_1, \ldots, X_n$: 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*

$$\Pr\Big[Y \geq \Delta\Big] \leq \exp\Big(-\Delta^2/2n\Big).$$

Proof of Chernoff inequality

Fix arbitrary $t > 0$:

$$\mathbf{Pr}\Big[Y \geq \Delta \Big] = \mathbf{Pr}\Big[tY \geq t\Delta \Big]$$

Proof of Chernoff inequality

Fix arbitrary $t > 0$:

$$\mathbf{Pr}\Big[Y \geq \Delta\Big] = \mathbf{Pr}\Big[tY \geq t\Delta\Big] = \mathbf{Pr}\Big[\exp(tY) \geq \exp(t\Delta)\Big]$$

Proof of Chernoff inequality

Fix arbitrary $t > 0$:

$$\mathbf{Pr}\Big[Y \geq \Delta \Big] = \mathbf{Pr}\Big[tY \geq t\Delta \Big] = \mathbf{Pr}\Big[\exp(tY) \geq \exp(t\Delta) \Big]$$
$$\leq \frac{\mathbf{E}\Big[\exp(tY) \Big]}{\exp(t\Delta)},$$

Proof of Chernoff inequality

Continued...

$$\mathbf{E}\left[\exp(tX_i)\right] = \frac{1}{2}e^{t} + \frac{1}{2}e^{-t}.$$

Proof of Chernoff inequality

Continued...

$$\mathbf{E}\left[\exp(tX_i)\right] = \frac{1}{2}e^t + \frac{1}{2}e^{-t} = \frac{e^t + e^{-t}}{2}.$$

Proof of Chernoff inequality

Continued...

$$\mathbf{E}\Big[\exp(tX_i)\Big] = \frac{e^t + e^{-t}}{2}.$$

Proof of Chernoff inequality

Continued...

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

$$= \frac{1}{2}\left(1 + \frac{t}{1!} + \frac{t^2}{2!} + \frac{t^3}{3!} + \cdots\right) + \frac{1}{2}\left(1 - \frac{t}{1!} + \frac{t^2}{2!} - \frac{t^3}{3!} + \cdots\right).$$

Proof of Chernoff inequality
Continued...

$$\mathbf{E}\Big[\exp(tX_i)\Big] = \frac{1}{2}\left(1 + \frac{t}{1!} + \frac{t^2}{2!} + \frac{t^3}{3!} + \cdots\right) + \frac{1}{2}\left(1 - \frac{t}{1!} + \frac{t^2}{2!} - \frac{t^3}{3!} + \cdots\right).$$

Proof of Chernoff inequality

Continued...

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

Proof of Chernoff inequality

Continued...

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

Proof of Chernoff inequality

Continued...

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

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

Proof of Chernoff inequality

Continued...

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

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

$$\mathbf{E}\left[\exp(tX_i)\right] = \sum_{i=0}^{\infty} \frac{t^{2i}}{(2i)!}$$

Proof of Chernoff inequality

Continued...

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

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

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

Proof of Chernoff inequality

Continued...

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

Proof of Chernoff inequality

Continued...

$$\mathbf{E}\Big[\exp(tX_i)\Big] \leq \sum_{i=0}^{\infty} \frac{t^{2i}}{2^i(i!)} = \sum_{i=0}^{\infty} \frac{1}{i!}\left(\frac{t^2}{2}\right)^i$$

Proof of Chernoff inequality

Continued...

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

Proof of Chernoff inequality

Continued...

$$\mathbf{E}\Big[\exp(tX_i)\Big] \leq \sum_{i=0}^{\infty} \frac{1}{i!}\left(\frac{t^2}{2}\right)^i = \exp\left(\frac{t^2}{2}\right).$$

Proof of Chernoff inequality

Continued...

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

Proof of Chernoff inequality

Continued...

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

$$\mathbf{E}\Big[\exp(tY)\Big] = \mathbf{E}\left[\exp\left(\sum_i tX_i\right)\right]$$

Proof of Chernoff inequality

Continued...

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

$$\mathbf{E}\left[\exp(tY)\right] = \mathbf{E}\left[\exp\left(\sum_i tX_i\right)\right] = \mathbf{E}\left[\prod_i \exp(tX_i)\right]$$

Proof of Chernoff inequality

Continued...

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

$$\mathbf{E}\left[\exp(tY)\right] = \mathbf{E}\left[\prod_i \exp(tX_i)\right]$$

Proof of Chernoff inequality

Continued...

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

$$\mathbf{E}\left[\exp(tY)\right] = \mathbf{E}\left[\prod_i \exp(tX_i)\right] = \prod_{i=1}^{n} \mathbf{E}\left[\exp(tX_i)\right]$$

Proof of Chernoff inequality

Continued...

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

$$\mathbf{E}\Big[\exp(tY)\Big] = \prod_{i=1}^{n} \mathbf{E}\Big[\exp(tX_i)\Big]$$

Proof of Chernoff inequality

Continued...

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

$$\mathbf{E}\left[\exp(tY)\right] = \prod_{i=1}^{n} \mathbf{E}\left[\exp(tX_i)\right] \leq \prod_{i=1}^{n} \exp\left(\frac{t^2}{2}\right)$$

Proof of Chernoff inequality

Continued...

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

$$\mathbf{E}\Big[\exp(tY)\Big] \leq \prod_{i=1}^{n} \exp\left(\frac{t^2}{2}\right)$$

Proof of Chernoff inequality

Continued...

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

$$\mathbf{E}\Big[\exp(tY)\Big] \leq \prod_{i=1}^{n} \exp\left(\frac{t^2}{2}\right) = \exp\left(\frac{nt^2}{2}\right).$$

Proof of Chernoff inequality

Continued...

$$\mathbf{E}\Big[\exp(tY)\Big] \le \prod_{i=1}^{n} \exp\left(\frac{t^2}{2}\right) = \exp\left(\frac{nt^2}{2}\right).$$

Proof of Chernoff inequality

Continued...

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

Proof of Chernoff inequality

Continued...

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

$$\mathbf{Pr}\Big[\,Y \geq \Delta\Big]$$

Proof of Chernoff inequality

Continued...

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

$$\Pr\Big[Y \geq \Delta\Big] \leq \frac{\mathbf{E}\Big[\exp(tY)\Big]}{\exp(t\Delta)}$$

Proof of Chernoff inequality

Continued...

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

$$\mathbf{Pr}\Big[Y \geq \Delta\Big] \leq \frac{\mathbf{E}\Big[\exp(tY)\Big]}{\exp(t\Delta)} \leq \frac{\exp\left(\frac{nt^2}{2}\right)}{\exp(t\Delta)}$$

Proof of Chernoff inequality

Continued...

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

$$\mathbf{Pr}\left[Y \geq \Delta\right] \leq \frac{\exp\left(\frac{nt^2}{2}\right)}{\exp(t\Delta)}$$

Proof of Chernoff inequality

Continued...

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

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

Proof of Chernoff inequality

Continued...

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

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

Proof of Chernoff inequality

Continued...

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

Set $t = \Delta / n$:

Proof of Chernoff inequality

Continued...

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

Set $t = \Delta / n$:

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

Proof of Chernoff inequality

Continued...

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

Set $t = \Delta / n$:

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

■

Chernoff inequality...

...what it really says

By theorem:

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

Chernoff inequality...

symmetry

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*

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

Chernoff inequality for coin flips

$X_1, \ldots, X_n$ 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

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

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:

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

where $\Delta = M/2 - L \geq M/4$. Using the above Chernoff inequality, we get
$\Pr\Big[Y \leq L\Big] \leq \exp\Big(-\frac{2\Delta^2}{M}\Big) \leq \exp(-M/8)$. □

Part V

The Chernoff Bound — General Case

Problem

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

$$\Pr\Big[X_i = 1\Big] = p_i \qquad \textit{and} \qquad \Pr\Big[X_i = 0\Big] = 1 - p_i,$$

and let denote

$$Y = \sum_i X_i \qquad \mu = \mathbf{E}\big[Y\big].$$

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

Theorem (Chernoff inequality)

For any $\delta > 0$,

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

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

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

and

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

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

Theorem

Theorem

Under the same assumptions as the theorem above, we have

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

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.

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.

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.

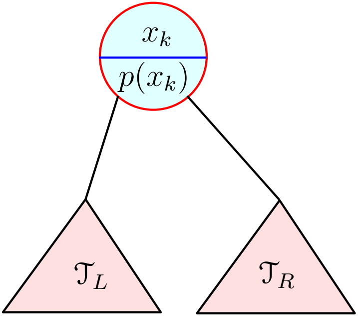
Balanced binary search trees...

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.

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.

Treaps

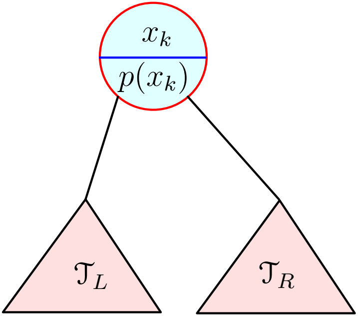


Continuing recursively, we have:

(A) $\boldsymbol{L}$: set of all the numbers smaller than $\boldsymbol{x_k}$ in $\boldsymbol{X}$, and
(B) $\boldsymbol{R}$: set of all the numbers larger than $\boldsymbol{x_k}$ in $\boldsymbol{X}$.

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 ***treap***.

Tree over the elements, and a heap over the priorities; that is, $\text{TREAP} = \text{TREE} + \text{HEAP}$.

Treaps continued

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... □

Treaps continued

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.

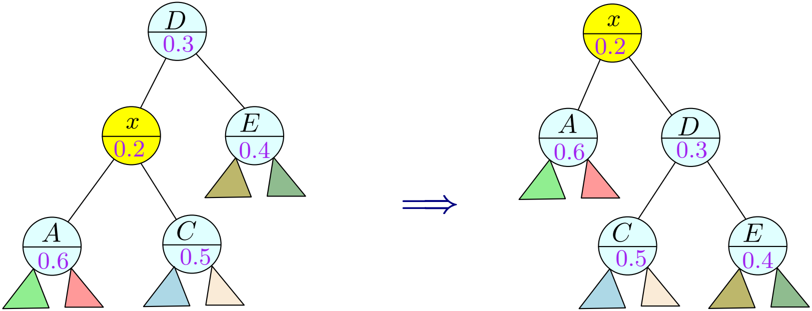
QuickSort... □

Treaps - implementation

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 $\mathbf{T}$ as a regular binary tree.
3. Takes $O(\text{height}(\mathbf{T}))$.
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 $\mathbf{T}$ as a regular binary tree.
3. Takes $O(\text{height}(\mathbf{T}))$.
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 $O(\text{height}(\mathbf{T}))$.
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 $\mathbf{T}$ as a regular binary tree.
3. Takes $O(\text{height}(\mathbf{T}))$.
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 $O(\text{height}(\mathbf{T}))$.
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 $\mathbf{T}$ as a regular binary tree.
3. Takes $O(\text{height}(\mathbf{T}))$.
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 $O(\text{height}(\mathbf{T}))$.
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...

```
RotateUp(x)
   y ← parent(x)
   while p(y) > p(x) do
       if y.left_child = x then
           RotateRight(y)
       else
           RotateLeft(y)
       y ← parent(x)
```

Insertion takes $O(\text{height}(\mathsf{T}))$.

Fix treap for a leaf x...

```
RotateUp(x)
    y ← parent(x)
    while p(y) > p(x) do
        if y.left_child = x then
            RotateRight(y)
        else
            RotateLeft(y)
        y ← parent(x)
```

Insertion takes $O(\text{height}(\mathsf{T}))$.

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...

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...

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...

Split

1. x: element stored in treap $\mathbf{T}$.
2. split $\mathbf{T}$ into two treaps – one treap $\mathbf{T}_{\leq x}$ and treap $\mathbf{T}_{>}$ 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 $\mathbf{T}_{\leq x}$ and treap $\mathbf{T}_{>}$ 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 $\mathbf{T}$.
2. split $\mathbf{T}$ into two treaps – one treap $\mathbf{T}_{\leq x}$ and treap $\mathbf{T}_{>}$ 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 $\mathbf{T}$.
2. split $\mathbf{T}$ into two treaps – one treap $\mathbf{T}_{\leq x}$ and treap $\mathbf{T}_{>}$ 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 $\mathbf{T}$.
2. split $\mathbf{T}$ into two treaps – one treap $\mathbf{T}_{\leq x}$ and treap $\mathbf{T}_{>}$ 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 $\mathbf{T}$.
2. split $\mathbf{T}$ into two treaps – one treap $\mathbf{T}_{\leq x}$ and treap $\mathbf{T}_{>}$ 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. $\mathbf{T}_L$ and $\mathbf{T}_R$: treaps.
2. all elements in $\mathbf{T}_L$ ¡ all elements in $\mathbf{T}_R$.
3. Want to merge them into a single treap...

Treap – summary

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

Proof.

1. $\mathsf{T}_1, \ldots, \mathsf{T}_m$: sequence of treaps.
2. T_i is treap after ith operation.
3. $\alpha_i = \Pr\Big[\mathrm{depth}(\mathsf{T}_i) > tc' \log n\Big] = \Pr\Big[\mathrm{depth}(\mathsf{T}_i) > c' t\Big(\frac{\log n}{\log|\mathsf{T}_i|}\Big) \cdot \log |\mathsf{T}_i|\Big] \leq \frac{1}{n^{O(1)}}$,
4. Use union bound...

Proof

Proof.

1. $\mathsf{T}_1, \ldots, \mathsf{T}_m$: sequence of treaps.
2. T_i is treap after ith operation.
3. $\alpha_i = \Pr\left[\mathrm{depth}(\mathsf{T}_i) > tc' \log n\right] = \Pr\left[\mathrm{depth}(\mathsf{T}_i) > c' t \left(\frac{\log n}{\log |\mathsf{T}_i|}\right) \cdot \log |\mathsf{T}_i|\right] \leq \frac{1}{n^{O(1)}}$,
4. Use union bound...

□

Proof

Proof.

1. $\mathsf{T}_1, \ldots, \mathsf{T}_m$: sequence of treaps.
2. T_i is treap after ith operation.
3. $\alpha_i = \Pr\left[\text{depth}(\mathsf{T}_i) > tc' \log n\right] = \Pr\left[\text{depth}(\mathsf{T}_i) > c' t\left(\frac{\log n}{\log |\mathsf{T}_i|}\right) \cdot \log |\mathsf{T}_i|\right] \leq \frac{1}{n^{O(1)}}$,
4. Use union bound...

Proof

Proof.

1. $\mathsf{T}_1, \ldots, \mathsf{T}_m$: sequence of treaps.
2. T_i is treap after ith operation.
3. $\alpha_i = \Pr\Big[\mathrm{depth}(\mathsf{T}_i) > tc' \log n\Big] =$
$\Pr\Big[\mathrm{depth}(\mathsf{T}_i) > c' t\Big(\frac{\log n}{\log|\mathsf{T}_i|}\Big) \cdot \log |\mathsf{T}_i|\Big] \leq \frac{1}{n^{O(1)}}$,
4. Use union bound...

□

Bibliographical Notes

1. Chernoff inequality was a rediscovery of Bernstein inequality.
2. ...published in 1924 by Sergei Bernstein.
3. Treaps were invented by Siedel and Aragon **?**.
4. Experimental evidence suggests that Treaps performs reasonably well in practice see **?**.
5. Old implementation of treaps I wrote in `C` is available here: http://valis.cs.uiuc.edu/blog/?p=6060.

Notes

Notes

Notes

Notes