Probabilistic analysis of algorithms: What’s it good for?

Conrado Martínez

Univ. Politècnica de Catalunya, Spain

University of Cape Town
February 2008
1. Introduction

2. Example #1: Generating random derangements

3. Example #2: Updating K-d trees

4. Example #3: Partial sorting

5. Concluding remarks
Introduction

Probabilistic analysis of algorithms is the right tool when

- We want to analyze "typical" behavior of algorithms
- We want to compare algorithms with asymptotically equivalent performances
- We want to analyze randomized algorithms (essential!)
- We want to have some mathematical fun :)

Probabilistic analysis of algorithms is the right tool when

- We want to analyze "typical" behavior of algorithms
- We want to compare algorithms with asymptotically equivalent performances
- We want to analyze randomized algorithms (essential!)
- We want to have some mathematical fun :)

Introduction
Introduction

Probabilistic analysis of algorithms is the right tool when

- We want to analyze "typical" behavior of algorithms
- We want to compare algorithms with asymptotically equivalent performances
- We want to analyze randomized algorithms (essential!)
- We want to have some mathematical fun :)}
Probabilistic analysis of algorithms is the right tool when

- We want to analyze "typical" behavior of algorithms
- We want to compare algorithms with asymptotically equivalent performances
- We want to analyze randomized algorithms (essential!)
- We want to have some mathematical fun :)

Introduction
Introduction

A few well known examples:

- Quicksort
- Find, a.k.a. Quickselect
- Hashing
- Simplex
- Randomized data structures
- and many more ...
Introduction

A few well known examples:

- Quicksort
- Find, a.k.a. Quickselect
- Hashing
- Simplex
- Randomized data structures
- and many more ...
A few well known examples:

- Quicksort
- Find, a.k.a. Quickselect
- Hashing
- Simplex
- Randomized data structures
- and many more …
A few well known examples:

- Quicksort
- Find, a.k.a. Quickselect
- Hashing
- Simplex
- Randomized data structures
- and many more …
Introduction

A few well known examples:

- Quicksort
- Find, a.k.a. Quickselect
- Hashing
- Simplex
- Randomized data structures

and many more ...
Introduction

A few well known examples:

- Quicksort
- Find, a.k.a. Quickselect
- Hashing
- Simplex
- Randomized data structures
- and many more …
Introduction

It was indeed very difficult for me to make a choice of examples ...
Introduction

...even if I restricted myself to those few that I’ve worked out myself!

- Randomized binary search trees
- Optimal sampling for quicksort and quickselect
- Adaptive sampling for quickselect
- Updates and associative queries in relaxed $K$-d trees
- Exhaustive and random generation of combinatorial objects
- Partial sorting
- Probabilistic analysis of binary search trees, skip lists, ...
1. Introduction

2. Example #1: Generating random derangements

3. Example #2: Updating $K$-d trees

4. Example #3: Partial sorting

5. Concluding remarks
Example #1: Generating random derangements

Le Problème des Derangements:
"A number of gentlemen, say \( n \), surrender their top hats in the cloakroom and proceed to the evening’s enjoyment. After wining and dining (and wining some more), they stumble back to the cloakroom and confusedly take the first top-hat they see. What is the probability that no gentleman gets his own hat?"
Derangements

- A derangement is a permutation without fixed points: $\pi(i) \neq i$ for any $i$, $1 \leq i \leq n$
- The number $D_n$ of derangements of size $n$ is

$$D_n = n! \cdot \left[ \frac{1}{0!} - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \cdots + \frac{(-1)^n}{n!} \right] = \left\lfloor \frac{n! + 1}{e} \right\rfloor.$$
- As $n \to \infty$, $D_n/n! \sim 1/e \approx 0.36788$. In fact, $e^{-1}$ is an extremely good approximation to the probability that a random permutation is a derangement for $n \geq 10$. 
A derangement is a permutation without fixed points: $\pi(i) \neq i$ for any $i$, $1 \leq i \leq n$

The number $D_n$ of derangements of size $n$ is

\[
D_n = n! \cdot \left[ \frac{1}{0!} - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \cdots + \frac{(-1)^n}{n!} \right] = \left\lfloor \frac{n! + 1}{e} \right\rfloor.
\]

As $n \to \infty$, $D_n/n! \sim 1/e \approx 0.36788$. In fact, $e^{-1}$ is a extremely good approximation to the probability that a random permutation is a derangement for $n \geq 10$. 

\[\]

Derangements
A derangement is a permutation without fixed points: \( \pi(i) \neq i \) for any \( i, 1 \leq i \leq n \).

The number \( D_n \) of derangements of size \( n \) is

\[
D_n = n! \cdot \left[ \frac{1}{0!} - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \cdots + \frac{(-1)^n}{n!} \right] = \left[ \frac{n! + 1}{e} \right].
\]

As \( n \to \infty \), \( D_n/n! \sim 1/e \approx 0.36788 \). In fact, \( e^{-1} \) is an extremely good approximation to the probability that a random permutation is a derangement for \( n \geq 10 \).
Excursion: Fisher-Yates’ shuffle

```plaintext
procedure RandomPermutation(n)
    for i ← 1 to n do A[i] ← i
    for i ← n downto 1 do
        j ← Uniform(1, i)
    return A
```
Excursion: Sattolo’s algorithm

```plaintext
procedure RandomCyclicPermutation(n)
    for i ← 1 to n do A[i] ← i
    for i ← n downto 1 do
        j ← Uniform(1, i − 1)
return A
```
Excursion: The rejection method

Require: \( n \neq 1 \)

procedure RandomDerangement(n)
  repeat
    \( A \leftarrow \text{RandomPermutation}(n) \)
  until Is-Derangement(\( A \)) return \( A \)

\( \mathbb{P}[A \text{ is a derangement}] \approx \frac{1}{e} \)

\( \mathbb{E}[\# \text{ of calls to Random}] = e \cdot n + O(1) \)
Excursion: The rejection method

Require: \( n \neq 1 \)

procedure RandomDerangement\((n)\)

repeat
  \( A \leftarrow \text{RandomPermutation} \( (n) \) \)
until Is-Derangement\((A)\) return \( A \)

\[
P[A \text{ is a derangement}] \approx \frac{1}{e}
\]

\[
\mathbb{E}[\text{\# of calls to Random}] = e \cdot n + O(1)
\]
A recurrence for the number of derangements

\[ D_0 = 1, \quad D_1 = 0 \]
\[ D_n = (n - 1)D_{n-1} + (n - 1)D_{n-2} \]
A recurrence for the number of derangements

\[ D_0 = 1, \quad D_1 = 0 \]
\[ D_n = (n - 1)D_{n-1} + (n - 1)D_{n-2} \]

Choice #1: \( n \) belongs to a cycle of length \( > 2 \).
The derangement of size \( n \) is built by constructing a derangement of size \( n - 1 \) and then \( n \) is inserted into any of the cycles (of length \( \geq 2 \)); there are \( (n - 1) \) possible ways to do that.
A recurrence for the number of derangements

\[ D_0 = 1, \quad D_1 = 0 \]
\[ D_n = (n - 1)D_{n-1} + (n - 1)D_{n-2} \]

Choice #2: \( n \) belongs to a cycle of length 2.
The derangement of size \( n \) is built by constructing a cycle of size 2 with \( n \) and some \( j, 1 \leq j \leq n - 1 \); then we build a derangement of size \( n - 2 \) with the remaining elements.
The recursive method

\[ C \leftarrow \{1, 2, \ldots, n\} \]
\[ \text{RandomDerangement-Rec}(n, C') \]

Require: \( n \neq 1 \)

procedure RandomDerangement-Rec\((n, C')\)
  if \( n \leq 1 \) then return
  \[ j \leftarrow \text{a random element from } C \]
  \[ p \leftarrow \text{Uniform}(0, 1) \]
  if \( p < (n - 1)D_{n-2}/D_n \) then
    RandomDerangement-Rec\((n - 2, C \setminus \{j, n\})\)
    \[ \pi(n) \leftarrow j; \pi(j) \leftarrow n \]
  else
    RandomDerangement-Rec\((n - 1, C \setminus \{n\})\)
    \[ \pi(n) \leftarrow \pi(j); \pi(j) \leftarrow n \]
Our algorithm

Require: \( n \neq 1 \)

procedure RandomDerangement(n)
    for \( i \leftarrow 1 \) to \( n \) do \( A[i] \leftarrow i; mark[i] \leftarrow \text{false} \)
    \( i \leftarrow n; u \leftarrow n \)
    while \( u \geq 2 \) do
        if \( \neg mark[i] \) then
            \( j \leftarrow \) a random unmarked element in \( A[1..i-1] \)
            \( A[i] \leftrightarrow A[j] \)
        if \( j \) has to close a cycle then
            \( mark[j] \leftarrow \text{true}; u \leftarrow u - 1 \)
            \( u \leftarrow u - 1 \)
        \( i \leftarrow i - 1 \)
    return \( A \)
Our algorithm

Require: \( n \neq 1 \)

procedure RandomDerangement\((n)\)

for \( i \leftarrow 1 \) to \( n \) do \( A[i] \leftarrow i; mark[i] \leftarrow false \)

\( i \leftarrow n; u \leftarrow n \)

while \( u \geq 2 \) do

if \( \neg mark[i] \) then

repeat \( j \leftarrow \text{Random}(1, i - 1) \)

until \( \neg mark[j] \)

\( A[i] \leftrightarrow A[j] \)

\( p \leftarrow \text{Uniform}(0, 1) \)

if \( p < (u - 1)D_{u-2}/D_u \) then \( mark[j] \leftarrow true; u \leftarrow u - 1 \)

\( u \leftarrow u - 1 \)

\( i \leftarrow i - 1 \)

return \( A \)
The analysis

- \# of marked elements = \# of cycles \((C_n)\)
- \# of iterations = \# of calls to Uniform = \(n - C_n\)
- \(G = \# \text{ of calls to Random}\)
- \(G_i = \# \text{ of calls to Random at iteration } i\)

\[
E[\text{cost}] = n - E[C_n] + E[G]
\]

\[
= n - E[C_n] + \sum_{1 < i \leq n} E[G_i]
\]
The analysis

- # of marked elements = # of cycles \((C_n)\)
- # of iterations = # of calls to Uniform = \(n - C_n\)
- \(G = \#\) of calls to Random
- \(G_i = \#\) of calls to Random at iteration \(i\)

\[
E[\text{cost}] = n - E[C_n] + E[G]
\]

\[
= n - E[C_n] + \sum_{1<i\leq n} E[G_i]
\]
The analysis

- # of marked elements = # of cycles ($C_n$)
- # of iterations = # of calls to Uniform = $n - C_n$
- $G = $ # of calls to Random
- $G_i = $ # of calls to Random at iteration $i$

$$E[\text{cost}] = n - E[C_n] + E[G]$$

$$= n - E[C_n] + \sum_{1<i\leq n} E[G_i]$$
The analysis

- # of marked elements = # of cycles \((C_n)\)
- # of iterations = # of calls to Uniform = \(n - C_n\)
- \(G = \) # of calls to Random
- \(G_i = \) # of calls to Random at iteration \(i\)

\[
\mathbb{E}[\text{cost}] = n - \mathbb{E}[C_n] + \mathbb{E}[G]
= n - \mathbb{E}[C_n] + \sum_{1<i\leq n} \mathbb{E}[G_i]
\]
The analysis

- \# of marked elements = \# of cycles \((C_n)\)
- \# of iterations = \# of calls to Uniform = \(n - C_n\)
- \(G = \# of calls to Random\)
- \(G_i = \# of calls to Random at iteration i\)

\[
\mathbb{E}[\text{cost}] = n - \mathbb{E}[C_n] + \mathbb{E}[G] \\
= n - \mathbb{E}[C_n] + \sum_{1<i\leq n} \mathbb{E}[G_i]
\]
The analysis

The computation of \( \mathbb{E}[C_n] \) can be done via standard generating function techniques:

\[
C(z, v) = \sum_{A \in \mathcal{D}} \frac{z^{|A|}}{|A|!} v^{\# \text{cycles}(A)} = \exp \left( v \left( \log \frac{1}{1 - z} - z \right) \right) = e^{-vz} \frac{1}{(1 - z)^v}
\]

\[
\mathbb{E} \left[ v^{C_n} \right] = \frac{n!}{D_n} [z^n] C(z, v) = \frac{e^{1-v}}{(v - 1)!} n^{v-1} (1 + O(n^{-1+\epsilon}))
\]

\[
\mathbb{E}[C_n] = \log n + O(1), \quad \forall[C_n] = \log n + O(1)
\]

\[
\frac{C_n - \log n}{\sqrt{\log n}} \to \mathcal{N}(0, 1)
\]
The analysis

The computation of $E[C_n]$ can be done via standard generating function techniques:

$$C(z, v) = \sum_{A \in \mathcal{D}} \frac{z^{|A|}}{|A|!} v^{|\text{cycles}(A)|}$$

$$= \exp \left( v \left( \log \frac{1}{1 - z} - z \right) \right) = e^{-vz} \frac{1}{(1 - z)^v}$$

$$E\left[v^{C_n}\right] = \frac{n!}{D_n} [z^n] C(z, v) = \frac{e^{1-v}}{(v - 1)!} n^{v-1} \left( 1 + O(n^{-1+\epsilon}) \right)$$

$$E[C_n] = \log n + O(1), \quad \forall[C_n] = \log n + O(1)$$

$$\frac{C_n - \log n}{\sqrt{\log n}} \to \mathcal{N}(0, 1)$$
The analysis

The computation of $\mathbb{E}[C_n]$ can be done via standard generating function techniques:

$$C(z, v) = \sum_{A \in D} \frac{z^{|A|}}{|A|!} v^{|\text{cycles}(A)|}$$

$$= \exp \left( v \left( \log \frac{1}{1-z} - z \right) \right) = e^{-vz} \frac{1}{(1-z)^v}$$

$$\mathbb{E} \left[ v^{C_n} \right] = \frac{n!}{D_n} [z^n] C(z, v) = \frac{e^{1-v}}{(v-1)!} n^{v-1} (1 + O(n^{-1+\epsilon}))$$

$$\mathbb{E}[C_n] = \log n + O(1), \quad \forall[C_n] = \log n + O(1)$$

$$\frac{C_n - \log n}{\sqrt{\log n}} \to \mathcal{N}(0, 1)$$
The analysis

The computation of $E[C_n]$ can be done via standard generating function techniques:

$$C(z, v) = \sum_{A \in \mathcal{D}} \frac{z^{|A|}}{|A|!} v^{\# \text{cycles}(A)} = \exp \left( v \left( \log \frac{1}{1 - z} - z \right) \right) = e^{-vz} \frac{1}{(1 - z)^v}$$

$$E\left[ v^{C_n} \right] = \frac{n!}{D_n} [z^n] C(z, v) = \frac{e^{1-v}}{(v-1)!} n^{v-1} (1 + O(n^{-1+\epsilon}))$$

$$E[C_n] = \log n + O(1), \quad \forall[C_n] = \log n + O(1)$$

$$\frac{C_n - \log n}{\sqrt{\log n}} \to \mathcal{N}(0, 1)$$
The analysis

The computation of $\mathbb{E}[C_n]$ can be done via standard generating function techniques:

$$C(z, v) = \sum_{A \in \mathcal{D}} \frac{z^{|A|}}{|A|!} v^{\# \text{cycles}(A)}$$

$$= \exp \left( v \left( \log \frac{1}{1 - z} - z \right) \right) = e^{-vz} \frac{1}{(1 - z)^v}$$

$$\mathbb{E} \left[ v^{C_n} \right] = \frac{n!}{D_n} [z^n] C(z, v) = \frac{e^{1-v}}{(v - 1)!} n^{v-1} (1 + O(n^{-1+\epsilon}))$$

$$\mathbb{E}[C_n] = \log n + O(1), \quad \mathbb{V}[C_n] = \log n + O(1)$$

$$\frac{C_n - \log n}{\sqrt{\log n}} \to \mathcal{N}(0, 1)$$
The analysis

- $M_i$ indicator variable for the event "$A[i]$ gets marked"
- $M_i = 1 \implies G_i = 0$

$$E[G] = \sum_{1<i\leq n} E[G_i | M_i = 0] \cdot P[M_i = 0]$$
The analysis

- $M_i$ indicator variable for the event "$A[i]$ gets marked"
- $M_i = 1 \implies G_i = 0$

\[
E[G] = \sum_{1 < i \leq n} E[G_i \mid M_i = 0] \cdot P[M_i = 0]
\]
The analysis

- $M_i$ indicator variable for the event "$A[i]$ gets marked"
- $M_i = 1 \implies G_i = 0$

$$\mathbb{E}[G] = \sum_{1 < i \leq n} \mathbb{E}[G_i \mid M_i = 0] \cdot \mathbb{P}[M_i = 0]$$
The analysis

- \( U_i = \# \text{ of unmarked elements in } A[1..i]; U_n = n \)
- \( B_{i+1} = \# \text{ of marked elements in } A[1..i]; B_{n+1} = 0 \)
- \( U_i + B_{i+1} = i \)
- If \( A[i] \) is not marked then \( G_i \) is geometrically distributed with probability of success 

\[
(U_i - 1)/(i - 1) = (i - 1 - B_{i+1})/(i - 1); \text{ hence}
\]

\[
\mathbb{E}[G_i \mid M_i = 0] = \mathbb{E}\left[ \frac{i - 1}{i - 1 - B_{i+1}} \mid M_i = 0 \right]
\]
The analysis

- $U_i = \# \text{ of unmarked elements in } A[1..i]; U_n = n$
- $B_{i+1} = \# \text{ of marked elements in } A[1..i]; B_{n+1} = 0$
- $U_i + B_{i+1} = i$
- If $A[i]$ is not marked then $G_i$ is geometrically distributed with probability of success
  
  \[
  (U_i - 1)/(i - 1) = (i - 1 - B_{i+1})/(i - 1); \text{ hence}
  \]

\[
\mathbb{E}[G_i \mid M_i = 0] = \mathbb{E}\left[\frac{i - 1}{i - 1 - B_{i+1}} \mid M_i = 0\right]
\]
The analysis

\[ U_i = \# \text{ of unmarked elements in } A[1..i]; \ U_n = n \]
\[ B_{i+1} = \# \text{ of marked elements in } A[1..i]; \ B_{n+1} = 0 \]
\[ U_i + B_{i+1} = i \]

If \( A[i] \) is not marked then \( G_i \) is geometrically distributed with probability of success

\[ \frac{(U_i - 1)/(i - 1)}{i - 1} = \frac{(i - 1 - B_{i+1})/(i - 1)}{i - 1 - B_{i+1}}; \text{ hence} \]

\[ \mathbb{E}[G_i \mid M_i = 0] = \mathbb{E}\left[\frac{i - 1}{i - 1 - B_{i+1}} \mid M_i = 0\right] \]
The analysis

- $U_i = \# \text{ of unmarked elements in } A[1..i]; U_n = n$
- $B_{i+1} = \# \text{ of marked elements in } A[1..i]; B_{n+1} = 0$
- $U_i + B_{i+1} = i$
- If $A[i]$ is not marked then $G_i$ is geometrically distributed with probability of success

\[
(U_i - 1)/(i - 1) = (i - 1 - B_{i+1})/(i - 1); \text{ hence}
\]

\[
\mathbb{E}[G_i \mid M_i = 0] = \mathbb{E}\left[\frac{i - 1}{i - 1 - B_{i+1}} \mid M_i = 0\right]
\]
The analysis

- $B_{i+1} \leq C_n$
- $0 \leq B_{i+1} \leq i$
- $U_i \neq 1$ and $B_{i+1} \neq i - 1$ for all $1 \leq i \leq n$
- If $M_i = 0$ then $B_{i+1} < i - 1$
The analysis

- $B_{i+1} \leq C_n$
- $0 \leq B_{i+1} \leq i$
- $U_i \neq 1$ and $B_{i+1} \neq i - 1$ for all $1 \leq i \leq n$
- If $M_i = 0$ then $B_{i+1} < i - 1$
The analysis

- $B_{i+1} \leq C_n$
- $0 \leq B_{i+1} \leq i$
- $U_i \neq 1$ and $B_{i+1} \neq i - 1$ for all $1 \leq i \leq n$
- If $M_i = 0$ then $B_{i+1} < i - 1$
The analysis

- $B_{i+1} \leq C_n$
- $0 \leq B_{i+1} \leq i$
- $U_i \neq 1$ and $B_{i+1} \neq i - 1$ for all $1 \leq i \leq n$
- If $M_i = 0$ then $B_{i+1} < i - 1$
The analysis

\[ \mathbb{E}[G] = \sum_{1 < i \leq n} \mathbb{E}\left[ \frac{i - 1}{i - 1 - B_{i+1}} \mid M_i = 0 \right] \cdot \mathbb{P}[M_i = 0] \]

\[ \leq \sum_{1 < i \leq n} \mathbb{E}\left[ \min\left\{ i - 1, \frac{i - 1}{i - 1 - C_n} \right\} \right] \]

\[ \leq \sum_{1 \leq k \leq \lfloor n/2 \rfloor} \mathbb{P}[C_n = k] \left( \sum_{i=1}^{k+1} (i - 1) + \sum_{i=k+2}^{\lfloor n/2 \rfloor} \frac{i - 1}{i - 1 - k} \right) \]

\[ = n - 1 - \mathbb{E}[C_n] + \frac{1}{2} \mathbb{E}\left[ C_n^2 \right] + O(\mathbb{E}[C_n \log(n - C_n)]) \]

\[ = n + O(\mathbb{E}\left[ C_n^2 \right]) + O(\log n \cdot \mathbb{E}[C_n]) = n + O(\log^2 n) \]
The analysis

\[
\mathbb{E}[G] = \sum_{1 < i \leq n} \mathbb{E}\left[ \frac{i - 1}{i - 1 - B_{i+1}} \mid M_i = 0 \right] \cdot \mathbb{P}[M_i = 0]
\]

\[
\leq \sum_{1 < i \leq n} \mathbb{E}\left[ \min \left\{ i - 1, \frac{i - 1}{i - 1 - C_n} \right\} \right]
\]

\[
\leq \sum_{1 \leq k \leq \lfloor n/2 \rfloor} \mathbb{P}[C_n = k] \left( \sum_{i=1}^{k+1} (i - 1) + \sum_{i=k+2}^{[n/2]} \frac{i - 1}{i - 1 - k} \right)
\]

\[
= n - 1 - \mathbb{E}[C_n] + \frac{1}{2} \mathbb{E}\left[ C_n^2 \right] + O(\mathbb{E}[C_n \log(n - C_n)])
\]

\[
= n + O(\mathbb{E}\left[ C_n^2 \right]) + O(\log n \cdot \mathbb{E}[C_n]) = n + O(\log^2 n)
\]
The analysis

\[
\mathbb{E}[G] = \sum_{1 < i \leq n} \mathbb{E}\left[ \frac{i - 1}{i - 1 - B_{i+1}} \mid M_i = 0 \right] \cdot \mathbb{P}[M_i = 0]
\]

\[
\leq \sum_{1 < i \leq n} \mathbb{E}\left[ \min \left\{ i - 1, \frac{i - 1}{i - 1 - C_n} \right\} \right]
\]

\[
\leq \sum_{1 \leq k \leq \lfloor n/2 \rfloor} \mathbb{P}[C_n = k] \left( \sum_{i=1}^{k+1} (i - 1) + \sum_{i=k+2}^{\lfloor n/2 \rfloor} \frac{i - 1}{i - 1 - k} \right)
\]

\[
= n - 1 - \mathbb{E}[C_n] + \frac{1}{2} \mathbb{E}\left[ C_n^2 \right] + O(\mathbb{E}[C_n \log(n - C_n)])
\]

\[
= n + O(\mathbb{E}\left[ C_n^2 \right]) + O(\log n \cdot \mathbb{E}[C_n]) = n + O(\log^2 n)
\]
\[
\mathbb{E}[G] = \sum_{1 < i \leq n} \mathbb{E}\left[\frac{i - 1}{i - 1 - B_{i+1}} \mid M_i = 0\right] \cdot \mathbb{P}[M_i = 0] \\
\leq \sum_{1 < i \leq n} \mathbb{E}\left[\min \left\{ i - 1, \frac{i - 1}{i - 1 - C_n} \right\} \right] \\
\leq \sum_{1 \leq k \leq \lfloor n/2 \rfloor} \mathbb{P}[C_n = k] \left( \sum_{i=1}^{k+1} (i - 1) + \sum_{i=k+2}^{\lfloor n/2 \rfloor} \frac{i - 1}{i - 1 - k} \right) \\
= n - 1 - \mathbb{E}[C_n] + \frac{1}{2} \mathbb{E}\left[C_n^2\right] + O(\mathbb{E}[C_n \log(n - C_n)]) \\
= n + O(\mathbb{E}\left[C_n^2\right]) + O(\log n \cdot \mathbb{E}[C_n]) = n + O(\log^2 n)
\]
The analysis

\[ \mathbb{E}[G] = \sum_{1 < i \leq n} \mathbb{E} \left[ \frac{i - 1}{i - 1 - B_{i+1}} \mid M_i = 0 \right] \cdot \mathbb{P}[M_i = 0] \]

\[ \leq \sum_{1 < i \leq n} \mathbb{E} \left[ \min \left\{ i - 1, \frac{i - 1}{i - 1 - C_n} \right\} \right] \]

\[ \leq \sum_{1 \leq k \leq \lfloor n/2 \rfloor} \mathbb{P}[C_n = k] \left( \sum_{i=1}^{k+1} (i - 1) + \sum_{i=k+2}^{\lfloor n/2 \rfloor} \frac{i - 1}{i - 1 - k} \right) \]

\[ = n - 1 - \mathbb{E}[C_n] + \frac{1}{2} \mathbb{E} \left[ C_n^2 \right] + O(\mathbb{E}[C_n \log(n - C_n)]) \]

\[ = n + O(\mathbb{E} \left[ C_n^2 \right]) + O(\log n \cdot \mathbb{E}[C_n]) = n + O(\log^2 n) \]
The analysis

Since we also have $E[G] \geq n - E[C_n]$, we have finally

$$E[\text{cost}] = n - E[C_n] + E[G] = 2n + O(\log^2 n)$$
1. Introduction

2. Example #1: Generating random derangements

3. Example #2: Updating $K$-d trees

4. Example #3: Partial sorting

5. Concluding remarks
Example #2: Updating $K$-d trees
Example #2: Updating $K$-d trees
Example #2: Updating K-d trees
Example #2: Updating $K$-d trees
Example #2: Updating $K$-d trees
Example #2: Updating K-d trees
Insertion in relaxed $K$-d trees

```cpp
rkdt insert(rkdt t, const Elem& x) {
    int n = size(t);
    int u = random(0, n);
    if (u == n)
        return insert_at_root(t, x);
    else { // t cannot be empty
        int i = t -> descr;
        if (x[i] < t -> key[i])
            t -> left = insert(t -> left, x);
        else
            t -> right = insert(t -> right, x);
        return t;
    }
}
```
Deletion in relaxed $K$-d trees

```cpp
rkdt delete(rkdt t, const Elem& x) {
    if (t == NULL) return NULL;
    if (t -> key == x)
        return join(t -> left, t -> right);

    int i = t -> descr;
    if (x -> key[i] < t -> key[i])
        t -> left = delete(t -> left, x);
    else
        t -> right = delete(t -> right, x);
    return t;
}
```
Split: Case #1
Split: Case #1
Split: Case #1
Split: Case #2
Split: Case #2
Split: Case #2
Split: Case #2
Analysis of split/join

- $s_n = \text{avg. number of visited nodes in a split}$
- $m_n = \text{avg. number of visited nodes in a join}$

$$s_n = 1 + \frac{2}{nK} \sum_{0 \leq j < n} \frac{j + 1}{n + 1}s_j + \frac{2(K - 1)}{nK} \sum_{0 \leq j < n} s_j$$

$$+ \frac{K - 1}{K} \sum_{0 \leq j < n} \pi_{n,j}m_j,$$

where $\pi_{n,j}$ is probability of joining two trees with total size $j$. 
Analysis of split/join

- $s_n = \text{avg. number of visited nodes in a split}$
- $m_n = \text{avg. number of visited nodes in a join}$

$$s_n = 1 + \frac{2}{nK} \sum_{0 \leq j < n} \frac{j + 1}{n + 1} s_j + \frac{2(K - 1)}{nK} \sum_{0 \leq j < n} s_j$$

$$+ \frac{K - 1}{K} \sum_{0 \leq j < n} \pi_{n,j} m_j,$$

where $\pi_{n,j}$ is probability of joining two trees with total size $j$. 
Analysis of split/join

- $s_n =$ avg. number of visited nodes in a split
- $m_n =$ avg. number of visited nodes in a join

$$s_n = 1 + \frac{2}{nK} \sum_{0 \leq j < n} \frac{j + 1}{n + 1} s_j + \frac{2(K - 1)}{nK} \sum_{0 \leq j < n} s_j$$

$$+ \frac{K - 1}{K} \sum_{0 \leq j < n} \pi_{n,j} m_j,$$

where $\pi_{n,j}$ is probability of joining two trees with total size $j$. 

Analysis of split/join

- The recurrence for $s_n$ is

$$s_n = 1 + \frac{2}{nK} \sum_{0 \leq j < n} \frac{j + 1}{n + 1} s_j + \frac{2(K - 1)}{nK} \sum_{0 \leq j < n} s_j$$

$$+ \frac{2(K - 1)}{nK} \sum_{0 \leq j < n} \frac{n - j}{n + 1} m_j,$$

with $s_0 = 0$.

- The recurrence for $m_n$ has exactly the same shape with the rôles of $s_n$ and $m_n$ interchanged; it easily follows that $s_n = m_n$. 
Analysis of split/join

- Define

\[ S(z) = \sum_{n \geq 0} s_n z^n \]

- The recurrence for \( s_n \) translates to

\[ z \frac{d^2 S}{dz^2} + 2 \frac{1 - 2z}{1 - z} \frac{dS}{dz} \]

\[ - 2 \left( \frac{3K - 2}{K} - z \right) \frac{S(z)}{(1 - z)^2} = \frac{2}{(1 - z)^3} \]

with initial conditions \( S(0) = 0 \) and \( S'(0) = 1 \).
The homogeneous second order linear ODE is of hypergeometric type.

An easy particular solution of the ODE is

\[-\frac{1}{2} \left( \frac{K}{K - 1} \right) \frac{1}{1 - z}\]
Analysis of split/join

**Theorem**

The generating function $S(z)$ of the expected cost of split is, for any $K \geq 2$,

$$S(z) = \frac{1}{2} \frac{1}{1 - \frac{1}{K}} \left[ (1 - z)^{-\alpha} \cdot _2F_1 \left( \frac{1 - \alpha, 2 - \alpha}{2} \right| z \right) - \frac{1}{1 - z} \right],$$

where $\alpha = \alpha(K) = \frac{1}{2} \left( 1 + \sqrt{17 - \frac{16}{K}} \right)$. 
Analysis of split/join

Theorem

The expected cost $s_n$ of splitting a relaxed $K$-d tree of size $n$ is

$$s_n = \eta(K) n^{\phi(K)} + o(n),$$

with

$$\eta = \frac{1}{2} \frac{1}{1 - \frac{1}{K}} \frac{\Gamma(2\alpha - 1)}{\alpha \Gamma^3(\alpha)},$$

$$\phi = \alpha - 1 = \frac{1}{2} \left( \sqrt{17 - \frac{16}{K}} - 1 \right).$$
Analysis of split/join

Plot of $\phi(K)$
The cost of insertions and deletions

- The recurrence for the expected cost of an insertion is

\[
I_n = \frac{I_n}{n+1} + \left(1 - \frac{1}{n+1}\right) \left(1 + \frac{2}{n} \sum_{0 \leq j < n} \frac{j+1}{n+1} I_j \right)
\]

\[
= \frac{I_n}{n+1} + 1 + O \left(\frac{1}{n}\right) + \frac{2}{n+1} \sum_{0 \leq j < n} \frac{j+1}{n+1} I_j.
\]

with \(I_n\) the average cost of an insertion at root

- The expected cost of deletions \(D_n\) satisfies a similar recurrence; it is asymptotically equivalent to the average cost of insertions

- We substitute \(I_n\) by the costs obtained previously \((s_n)\)
The cost of insertions and deletions

**Theorem**

Let \( I_n \) and \( D_n \) denote the average cost of a randomized insertion and randomized deletion in a random relaxed \( K \)-d tree of size \( n \) using split and join. Then

1. If \( K = 2 \) then \( I_n \sim D_n = 4 \ln n + O(1) \).
2. If \( K > 2 \) then

\[
I_n \sim D_n = \eta \frac{\phi - 1}{\phi + 1} n^{\phi - 1} + O(\log n),
\]

where \( I_n = \eta n^\phi + O(1) \).
The cost of insertions and deletions

**Theorem**

Let $I_n$ and $D_n$ denote the average cost of a randomized insertion and randomized deletion in a random relaxed $K$-d tree of size $n$ using split and join. Then

1. if $K = 2$ then $I_n \sim D_n = 4 \ln n + \mathcal{O}(1)$.
2. if $K > 2$ then

$$I_n \sim D_n = \eta \frac{\phi - 1}{\phi + 1} n^{\phi - 1} + \mathcal{O}(\log n),$$

where $I_n = \eta n^\phi + \mathcal{O}(1)$.

Note that for $K > 2$, $\phi(K) > 1!$
Copy-based insertions
Copy-based insertions

Diagram illustrating copy-based insertions with nodes and edges representing the structure.
Copy-based insertions
Copy-based insertions
Copy-based insertions
Excursion: Partial match

Given a query $q = (q_0, \ldots, q_{K_1})$ where each $q_i \in [0, 1]$ or $q_i = \star$, find all elements $x$ in the $K$-d tree such that $x_i = q_i$ whenever $q_i \neq \star$.

Partial match

```c
void partial_match(rkdt t, query q) {
    if (t == NULL) return;
    if (matches(t -> key, q))
        report(t-> key);
    int i = t -> descr;
    if (q[i] == '*') {
        partial_match(t -> left, q);
        partial_match(t -> right, q);
    } else if (q[i] < t -> key) {
        partial_match(t -> left, q);
    } else {
        partial_match(t -> left, q);
    }
}
```
Analysis of copy-based updates

The cost of building $T$ using copy-based insertion of a key $x$:

$$C(T) = P(T) + \frac{1}{K} \frac{|L| + 1}{|T| + 1} C(L) + \frac{1}{K} \frac{|R| + 1}{|T| + 1} C(R) + \frac{K - 1}{K} (C(L) + C(R)),$$

where $P(T)$ denotes the number of nodes visited by a partial match in $T - \{x\}$ with query $q = (x_0, \ldots, x_{i-1}, *, x_{i+1}, \ldots, x_{K-1})$. 
Analysis of copy-based updates

The cost of making an insertion at root into a tree of size $n$:

$$C_n = P_n + \frac{2}{nK} \sum_{0 \leq k < n} \frac{k + 1}{n + 1} C_k + \frac{2(K - 1)}{nK} \sum_{0 \leq k < n} C_k.$$  

with $P_n$ the expected cost of a partial match in a random relaxed $K$-d tree of size $n$ with only one specified coordinate out of $K$ coordinates.
Theorem (Duch et al. 1998, Martínez et al. 2001)

The expected cost $P_n$ (measured as the number of key comparisons) of a partial match query with $s$ out of $K$ attributes specified, $0 < s < K$, in a randomly built relaxed $K$-d tree of size $n$ is

$$P_n = \beta(s/K) \cdot n^{\rho(s/K)} + O(1),$$

where

$$\rho = \rho(x) = \left(\sqrt{9 - 8x} - 1\right)/2,$$

$$\beta(x) = \frac{\Gamma(2\rho + 1)}{(1 - x)(\rho + 1)\Gamma^3(\rho + 1)},$$

and $\Gamma(x)$ is Euler's Gamma function.
Analysis of copy-based updates

We will use Roura’s Continuous Master Theorem to solve recurrences of the form:

\[ F_n = t_n + \sum_{0 \leq j < n} w_{n,j} F_j, \quad n \geq n_0, \]

where \( t_n \) is the so-called toll function and the quantities \( w_{n,j} \geq 0 \) are called weights.
Excursion: Roura’s Continuous Master Theorem

**Theorem (Roura 2001)**

Let \( t_n \sim C n^a \log^b n \) for some constants \( C, a \geq 0 \) and \( b > -1 \), and let \( \omega(z) \) be a real function over \([0, 1]\) such that

\[
\sum_{0 \leq j < n} \left| w_{n,j} - \int_{j/n}^{(j+1)/n} \omega(z) \, dz \right| = O(n^{-d})
\]

for some constant \( d > 0 \). Let \( \phi(x) = \int_0^1 z^x \omega(z) \, dz \), and define \( H = 1 - \phi(a) \). Then

1. If \( H > 0 \) then \( F_n \sim t_n / H \).
2. If \( H = 0 \) then \( F_n \sim t_n \ln n / H' \), where \( H' = -(b + 1) \int_0^1 z^a \ln z \omega(z) \, dz \).
3. If \( H < 0 \) then \( F_n = \Theta(n^\alpha) \), where \( \alpha \) is the unique real solution of \( \phi(x) = 1 \).
Analysis of copy-based updates

Applying the CMT to our recurrence we have

- \( \omega(z) = \frac{2z}{K} + \frac{2(K-1)}{K} \)
- \( t_n = P_n \implies a = \rho = \rho(1/K) = (\sqrt{9 - 8/K} - 1)/2 \)

Thus \( H = 0 \)
Analysis of copy-based updates

Applying the CMT to our recurrence we have

- \( \omega(z) = \frac{2z}{K} + \frac{2(K-1)}{K} \)
- \( t_n = P_n \implies a = \varrho = \rho(1/K) = (\sqrt{9 - 8/K} - 1)/2 \)

Thus \( \mathcal{H} = 0 \)

We have to compute \( \mathcal{H}' \) with \( b = 0 \)

\[
\mathcal{H}' = -(b + 1) \int_0^1 z^a \omega(z) \ln z \, dz
\]

and get

\[
\mathcal{H}' = 2 \frac{K \varrho^2 + (4K - 2) \varrho + 4K - 3}{K(\varrho + 2)^2(\varrho + 1)^2}.
\]
Theorem

The average cost $C_n$ of copy-based insertion at root of a random relaxed $K$-d tree is

$$C_n = \gamma \cdot n^\varrho \ln n + o(n \ln n),$$

where

$$\varrho = \varrho(K) = \rho(1/K) = \left(\sqrt{9 - 8/K} - 1\right)/2,$$

$$\gamma = \frac{\beta(1/K)}{\mathcal{H}'} = \frac{\Gamma(2\varrho + 1)K(\varrho + 2)^2(\varrho + 1)}{2(1 - 1/K)^2\Gamma^3(\varrho + 1)(K\varrho^2 + (4K - 2)\varrho + (4K - 3))}.$$
The cost of insertions and deletions (2)

**Theorem**

For any fixed dimension $K \geq 2$, the average cost of a randomized insertion or deletion in random relaxed $K$-d tree of size $n$ using copy-based updates is

$$I_n \sim D_n = 2 \ln n + \Theta(1).$$
The cost of insertions and deletions (2)

**Theorem**

For any fixed dimension $K \geq 2$, the average cost of a randomized insertion or deletion in random relaxed $K$-d tree of size $n$ using copy-based updates is

$$I_n \sim D_n = 2 \ln n + \Theta(1).$$

The "reconstruction" phase has constant cost on the average!
1. Introduction

2. Example #1: Generating random derangements

3. Example #2: Updating K-d trees

4. Example #3: Partial sorting

5. Concluding remarks
Example #3: Partial sorting

- **Partial sorting**: Given an array $A$ of $n$ elements and a value $1 \leq m \leq n$, rearrange $A$ so that its first $m$ positions contain the $m$ smallest elements in ascending order.

- For $m = \Theta(n)$ it might be OK to sort the array; otherwise, we are doing too much work.
Example #3: Partial sorting

- **Partial sorting**: Given an array $A$ of $n$ elements and a value $1 \leq m \leq n$, rearrange $A$ so that its first $m$ positions contain the $m$ smallest elements in ascending order.

- For $m = \Theta(n)$ it might be OK to sort the array; otherwise, we are doing too much work.
A few common solutions

Idea #1: Partial heapsort

- Build a heap with the $n$ elements and perform $m$ extractions of the heap’s minimum
- The worst-case cost is $\Theta(n + m \log n)$
- This the “traditional” implementation of C++ STL’s `partial_sort`
A few common solutions

- **Idea #1: Partial heapsort**
  - Build a heap with the \( n \) elements and perform \( m \) extractions of the heap’s minimum
  - The worst-case cost is \( \Theta(n + m \log n) \)
  - This the "traditional" implementation of C++ STL’s `partial_sort`
A few common solutions

- **Idea #1: Partial heapsort**
  - Build a heap with the $n$ elements and perform $m$ extractions of the heap’s minimum
  - The worst-case cost is $\Theta(n + m \log n)$
  - This the “traditional” implementation of C++ STL’s `partial_sort`
A few common solutions

- Idea #2: On-line selection
  - Build a heap with the $m$ first elements; then scan the remaining $n - m$ elements and update the heap as needed; finally extract the $m$ elements from the heap
  - The worst-case cost is $\Theta(n \log m)$
  - Not very attractive unless $m$ is very small or if used in on-line settings
A few common solutions

- **Idea #2: On-line selection**
  - Build a heap with the $m$ first elements; then scan the remaining $n-m$ elements and update the heap as needed; finally extract the $m$ elements from the heap.
  - The worst-case cost is $\Theta(n \log m)$.
  - Not very attractive unless $m$ is very small or if used in on-line settings.
A few common solutions

- **Idea #2: On-line selection**
  - Build a heap with the $m$ first elements; then scan the remaining $n - m$ elements and update the heap as needed; finally extract the $m$ elements from the heap.
  - The worst-case cost is $\Theta(n \log m)$.
  - Not very attractive unless $m$ is very small or if used in on-line settings.
Idea #3: “Quickselsort”

- Find the $m$th smallest element with quickselect, then quicksort the preceding $m - 1$ elements
- The average cost is $\Theta(n + m \log m)$
- Uses two basic algorithms widely available (and highly tuned for performance in standard libraries)
Idea #3: "Quickselsort"

- Find the $m$th smallest element with quickselect, then quicksort the preceding $m - 1$ elements
- The average cost is $\Theta(n + m \log m)$
- Uses two basic algorithms widely available (and highly tuned for performance in standard libraries)
Idea #3: "QuickselSort"

- Find the $m$th smallest element with quickselect, then quicksort the preceding $m - 1$ elements.
- The average cost is $\Theta(n + m \log m)$.
- Uses two basic algorithms widely available (and highly tuned for performance in standard libraries).
void partial_quicksort(vector<Elem>& A,
                       int i, int j, int m) {
    if (i < j) {
        int p = get_pivot(A, i, j);
        swap(A[p], A[1]);
        int k;
        partition(A, i, j, k);
        partial_quicksort(A, i, k - 1, m);
        if (k < m - 1)
            partial_quicksort(A, k + 1, j, m);
    }
}
The analysis

- Probability that the selected pivot is the $k$-th of $n$ elements: $\pi_{n,k}$

- Average number of comparisons $P_{n,m}$ to sort the $m$ smallest elements out of $n$:

$$P_{n,m} = n - 1 + \sum_{k=m+1}^{n} \pi_{n,k} \cdot P_{k-1,m}$$

$$+ \sum_{k=1}^{m} \pi_{n,k} \cdot (P_{k-1,k-1} + P_{n-k,m-k})$$
The analysis

- Probability that the selected pivot is the $k$-th of $n$ elements: $\pi_{n,k}$
- Average number of comparisons $P_{n,m}$ to sort the $m$ smallest elements out of $n$:

$$P_{n,m} = n - 1 + \sum_{k=m+1}^{n} \pi_{n,k} \cdot P_{k-1,m}$$

$$+ \sum_{k=1}^{m} \pi_{n,k} \cdot (P_{k-1,k-1} + P_{n-k,m-k})$$
The analysis

For \( m = n \), partial quicksort \( \equiv \) quicksort; let \( q_n \) denote the average number of comparisons used by quicksort.

Hence,

\[
P_{n,m} = n - 1 + \sum_{0 \leq k < m} \pi_{n,k+1} \cdot q_k + \sum_{k=m+1}^{n} \pi_{n,k} \cdot P_{k-1,m} + \sum_{k=1}^{m} \pi_{n,k} \cdot P_{n-k,m-k} \quad (1)
\]
The analysis

- For \( m = n \), partial quicksort \( \equiv \) quicksort; let \( q_n \) denote the average number of comparisons used by quicksort.

- Hence,

\[
P_{n,m} = n - 1 + \sum_{0 \leq k < m} \pi_{n,k+1} \cdot q_k \\
+ \sum_{k=m+1}^{n} \pi_{n,k} \cdot P_{k-1,m} + \sum_{k=1}^{m} \pi_{n,k} \cdot P_{n-k,m-k}
\]  

(1)
The analysis

- The recurrence for $P_{n,m}$ is the same as for quickselect but the toll function is

$$t_{n,m} = n - 1 + \sum_{0 \leq k < m} \pi_{n,k+1} \cdot q_k$$

- Up to now, everything holds no matter which pivot selection scheme do we use; for the standard variant we must take $\pi_{n,k} = 1/n$, for all $1 \leq k \leq n$
The analysis

- The recurrence for $P_{n,m}$ is the same as for quickselect but the toll function is

\[ t_{n,m} = n - 1 + \sum_{0 \leq k < m} \pi_{n,k+1} \cdot q_k \]

- Up to now, everything holds no matter which pivot selection scheme do we use; for the standard variant we must take $\pi_{n,k} = 1/n$, for all $1 \leq k \leq n$
Define the two BGFs

\[ P(z, u) = \sum_{n \geq 0} \sum_{1 \leq m \leq n} P_{n,m} z^n u^m \]

\[ T(z, u) = \sum_{n \geq 0} \sum_{1 \leq m \leq n} t_{n,m} z^n u^m \]

Then the recurrence (1) translates to

\[ \frac{\partial P}{\partial z} = \frac{P(z, u)}{1 - z} + \frac{u P(z, u)}{1 - uz} + \frac{\partial T}{\partial z} \]
The analysis: Generating functions

- Define the two BGFs

\[
P(z, u) = \sum_{n \geq 0} \sum_{1 \leq m \leq n} P_{n,m} z^n u^m
\]

\[
T(z, u) = \sum_{n \geq 0} \sum_{1 \leq m \leq n} t_{n,m} z^n u^m
\]

- Then the recurrence (1) translates to

\[
\frac{\partial P}{\partial z} = \frac{P(z, u)}{1 - z} + \frac{u P(z, u)}{1 - uz} + \frac{\partial T}{\partial z}
\]  \(2\)
The analysis: Generating functions

Let $P(z, u) = F(z, u) + S(z, u)$, where $F(z, u)$ corresponds to the selection part of the toll function $(n - 1)$ and $S(z, u)$ to the sorting part $(\sum_k q_k/n)$.

Let

$$T_F(z, u) = \sum_{n \geq 0} \sum_{1 \leq m \leq n} (n - 1) z^n u^m$$

$$T_S(z, u) = \sum_{n \geq 0} \sum_{1 \leq m \leq n} \frac{1}{n} \left( \sum_{0 \leq k < m} q_k \right) z^n u^m$$
The analysis: Generating functions

- Let \( P(z, u) = F(z, u) + S(z, u) \), where \( F(z, u) \) corresponds to the selection part of the toll function \((n - 1)\) and \( S(z, u) \) to the sorting part \((\sum_k q_k/n)\)

- Let

\[
T_F(z, u) = \sum_{n \geq 0} \sum_{1 \leq m \leq n} (n - 1)z^n u^m
\]

\[
T_S(z, u) = \sum_{n \geq 0} \sum_{1 \leq m \leq n} \frac{1}{n} \left( \sum_{0 \leq k < m} q_k \right) z^n u^m
\]
Then, each of $F(z, u)$ and $S(z, u)$ satisfies a differential equation like (2) and

$$F(z, u) = \frac{1}{(1 - z)(1 - zu)} \times \left\{ \int (1 - z)(1 - zu) \frac{\partial T_F}{\partial z} \, dz + K_F \right\}$$

$$S(z, u) = \frac{1}{(1 - z)(1 - zu)} \times \left\{ \int (1 - z)(1 - zu) \frac{\partial T_S}{\partial z} \, dz + K_S \right\}$$
The analysis: Generating functions

- $F(z, u)$ satisfies exactly the same differential equation as standard quickselect; it is well known (Knuth, 1971) that for $1 \leq m \leq n$,

$$F_{n,m} = [z^n u^m] F(z, u) = 2(n + 3 + (n + 1)H_n$$

$$- (m + 2)H_m - (n + 3 - m)H_{n+1-m}$$
The analysis: Generating functions

To compute $S(z, u)$, we need first to determine $T_S(z, u)$

$$\frac{\partial T_S}{\partial z} = \frac{u}{1 - z} \frac{Q(uz)}{1 - uz}$$

where $Q(z) = \sum_{n \geq 0} q_n z^n$.

With the toll function $n - 1$, we solve the recurrence for quicksort to get

$$Q(z) = \frac{2}{(1 - z)^2} \left( \ln \frac{1}{1 - z} - z \right)$$
The analysis: Generating functions

To compute $S(z, u)$, we need first to determine $T_S(z, u)$

$$\frac{\partial T_S}{\partial z} = \frac{u}{1 - z} \frac{Q(uz)}{1 - uz}$$

where $Q(z) = \sum_{n \geq 0} q_n z^n$.

With the toll function $n - 1$, we solve the recurrence for quicksort to get

$$Q(z) = \frac{2}{(1 - z)^2} \left( \ln \frac{1}{1 - z} - z \right)$$
The analysis: Generating functions

Hence,

\[ S(z, u) = \frac{1}{(1 - z)(1 - uz)} \left\{ \int u Q(uz) \, dz + K_S \right\} \]

\[ = \frac{2}{(1 - uz)^2(1 - z)} \ln \frac{1}{1 - uz} \]

\[ + \frac{2}{(1 - z)(1 - uz)} \ln \frac{1}{1 - uz} \]

\[ - 4 \frac{uz}{(1 - uz)^2(1 - z)} \]
The analysis: Generating functions

- Extracting coefficients \( S_{n,m} = [z^n u^m] S(z, u) \)

\[
S_{n,m} = 2(m + 1)H_m - 6m + 2H_m
\]

- And finally

\[
P_{n,m} = 2n + 2(n + 1)H_n - 2(n + 3 - m)H_{n+1-m} - 6m + 6
\]
The analysis: Generating functions

- Extracting coefficients $S_{n,m} = [z^n u^m] S(z,u)$

$$S_{n,m} = 2(m + 1)H_m - 6m + 2H_m$$

- And finally

$$P_{n,m} = 2n + 2(n + 1)H_n - 2(n + 3 - m)H_{n+1-m} - 6m + 6$$
Partial quicksort vs. quickselsort

- The average number of comparisons made by quickselsort is

\[ Q_{n,m} = F_{n,m} + q_{m-1} \]

- Using partial quicksort we save

\[ Q_{n,m} - P_{n,m} = 2m - 4H_m + 2 \]

comparisons on the average
Partial quicksort vs. quickselsort

- The average number of comparisons made by quickselsort is

\[ Q_{n,m} = F_{n,m} + q_{m-1} \]

- Using partial quicksort we save

\[ Q_{n,m} - P_{n,m} = 2m - 4H_m + 2 \]

comparisons on the average.
Partial quicksort avoids some of the redundant comparisons, exchanges, ... made by quickselect.

- It is easily implemented.
- It benefits from standard optimization techniques: sampling, recursion removal, recursion cutoff on small subfiles, improved partitioning schemes, etc.
- The same idea can be applied to similar algorithms like radix sorting and quicksort for strings.
Final remarks on partial quicksort

- Partial quicksort avoids some of the redundant comparisons, exchanges, ... made by quickselect.
- It is easily implemented.
- It benefits from standard optimization techniques: sampling, recursion removal, recursion cutoff on small subfiles, improved partitioning schemes, etc.
- The same idea can be applied to similar algorithms like radix sorting and quicksort for strings.
Final remarks on partial quicksort

- Partial quicksort avoids some of the redundant comparisons, exchanges, ... made by quicksort.
- It is easily implemented.
- It benefits from standard optimization techniques: sampling, recursion removal, recursion cutoff on small subfiles, improved partitioning schemes, etc.
- The same idea can be applied to similar algorithms like radix sorting and quicksort for strings.
Final remarks on partial quicksort

- Partial quicksort avoids some of the redundant comparisons, exchanges, ... made by quickselects.
- It is easily implemented.
- It benefits from standard optimization techniques: sampling, recursion removal, recursion cutoff on small subfiles, improved partitioning schemes, etc.
- The same idea can be applied to similar algorithms like radix sorting and quicksort for strings.
1 Introduction

2 Example #1: Generating random derangements

3 Example #2: Updating K-d trees

4 Example #3: Partial sorting

5 Concluding remarks
Concluding remarks

I hope I have convinced you about the usefulness of probabilistic analysis

- Provides useful information about typical behavior
- Necessary when analyzing randomized algorithms
- Allows meaningful comparisons between competitors of equivalent performance
- A source of beautiful and challenging mathematical problems!
Concluding remarks

I hope I have convinced you about the usefulness of probabilistic analysis

- Provides useful information about typical behavior
- Necessary when analyzing randomized algorithms
- Allows meaningful comparisons between competitors of equivalent performance
- A source of beautiful and challenging mathematical problems!
Concluding remarks

I hope I have convinced you about the usefulness of probabilistic analysis

- Provides useful information about typical behavior
- Necessary when analyzing randomized algorithms
- Allows meaningful comparisons between competitors of equivalent performance
- A source of beautiful and challenging mathematical problems!
I hope I have convinced you about the usefulness of probabilistic analysis:

- Provides useful information about typical behavior
- Necessary when analyzing randomized algorithms
- Allows meaningful comparisons between competitors of equivalent performance
- A source of beautiful and challenging mathematical problems!
Credits

- Alois Panholzer and Helmut Prodinger: Generating random derangements
- Amalia Duch: Updating K-d trees
THANKS!!