Introduction to Algorithms

6.046J/18.410J

Prof. Shafi Goldwasser

LECTURE 5
Randomized Algorithms
Monte Carlo Algorithms (MC): Last Time

Let $n$ be the length of the input.

For every input:

- MC runs in worst case polynomial Time
- MC is correct with high probability
  - Constant probability > $\frac{1}{2}$
Revisit Example:

Distinguishing Prime and Composite Numbers
Let $N$ positive integer.

$\mathbb{Z}_N = \{0,1,2\ldots N-1\}$ under $+ \mod N$, identity = 0

$\mathbb{Z}_N^* = \{ A \mid 1 \leq A < N \text{ and } \gcd(A, N) = 1 \}$

under multiplication mod $N$, identity 1.

The inverse of $A$ is $x$ s.t. $Ax=1 \mod N$.

**EX:** $\mathbb{Z}_6^* = \{1,5\}$,  **EX:** $\mathbb{Z}_7^* = \{1,\ldots,6\}$

**Computing inverses:** $O(n^2)$.

Given $A$ in $\mathbb{Z}_N^*$, by definition $\gcd(A,N)=1$,

so extended-$\text{GCD}(A,N) = (1,x,y)$ s.t.

$xA+yN=1 \implies xA+yN=1 \mod N$. But, $yN=0 \mod N$,

so $xA=1 \mod N$ and $x=A^{-1} \mod N$. 

**Our Basic Groups**
Lagrange

Theorem [Lagrange]: Let $G$ is a group, $H$ a subgroup of $G$. Then $|H|$ divides $|G|$.
(orders divide)
If \( N \) is prime, then \( \forall \) integer \( a \) s.t. \( 1 \leq a \leq N-1 \), 
\[ a^{N-1} = 1 \pmod{N} \]

**Fermat’s Little Theorem**

**Idea for primality test:** On input \( N \), **look for** an \( 1 \leq a \leq N-1 \), such that \( a^{N-1} \neq 1 \pmod{N} \). If find it, then \( N \) is NOT prime.

**For example:** let \( a=2 \) and test if 
\[ 2^{14} \pmod{15} = ? 1 \]
\[ 2^{14} = 4 \pmod{15} \Rightarrow 15 \text{ is composite} \]
Unfortunately, Not an “If and only if”

$$\forall a, \ 1 \leq a \leq N-1 \ , \ a^{N-1} = 1 \pmod{N} \nRightarrow N \text{ is prime}$$

There are composite $N$ (e.g. 561) for which $\forall a, \ a^{N-1} = 1 \pmod{N}$.
These are called **Carmichael Numbers**
and there are infinitely many of them.

(although they are rare – only 255 of them appear in the first $10^8$ integers)
Fermat’s Probabilistic Primality Test (for non-Charmichael numbers N)

Fermat’s Primality Test (N>2 non-Charmichael)
Pick \( a \in \{1\ldots N-1\} \) at random
If \( \gcd(a,N) > 1 \), output “N is composite”
  if \( a^{N-1} \neq 1 \pmod{N} \),
    output “N is composite”
  else output “N probably prime”

Call such a, “witness of compositeness”

Easy Observations
Analysis: Runs in polynomial time
Correctness on N prime: always says ‘N probably prime’
Prob[error] = 0
Correctness on N composite (not Carmichael)

Theorem. If N is composite (not Carmichael) the probₐ(output ‘N probably prime’) ≤ 1/2.

Proof: Define B = \{ a ∈ Zₙ* | a^{N-1} = 1 (mod B) \}
B contains all the inputs which cause Fermat’s test to produce an error. It is a subgroup of Zₙ*

Closure: a^{N-1} = 1 (mod N), b^{N-1} = 1 (mod N) ⇒ (ab)^{N-1} = 1 (mod N)
Identity: 1^{N-1} = 1 (mod N)
Associativity: multiplication is associative
Inverses: a^{N-1} = 1 ⇒ (a^{N-1})^{-1} = 1 ⇒ (a^{-1})^{N-1} = 1 (mod N)

1/2 |Zₙ*| ≥ 1/2 (N-1).
Correctness on N composite (not Carmichael)

Theorem. If N is composite (not Carmichael) the \( \text{prob}_a(\text{output } `N \text{ probably prime'}) \leq 1/2. \)

Proof: Define \( B = \{ a \in \mathbb{Z}_N^* \mid a^{N-1} = 1 \text{ (mod } B) \} \)
B contains all the inputs which cause Fermat’s test to produce an error. It is a subgroup of \( \mathbb{Z}_N^* \)

Since \( N \) is neither prime nor Carmichael, \( B \neq \mathbb{Z}_N^* \) which means \( |B| < \text{(strictly smaller)} \ |\mathbb{Z}_N^*| \)

Finally: \( |B| \text{ divides } |\mathbb{Z}_n^*| \) \ (Lagrange’s theorem) \( \Rightarrow |B| \leq 1/2 \ |\mathbb{Z}_n^*| \leq 1/2 \ (N-1). \)
Boosting the Probability by repeating

**Fermat’s primality test (N > 2 non Charmichael):**

repeat \( k \) times

pick \( a \in \{1\ldots N - 1\} \) at random

if \( \gcd(a,N) > 1 \) output “N is composite”

if \( a^{N-1} \neq 1 \pmod{N} \) then

output “N is composite”

else output “N probably prime”,

**Claim:** if N is composite,

\[
\text{prob}[\text{output `N probably prime’ } ] \leq (1/2)^k
\]
Nice warm up, but ultimately unsatisfactory – does not give guarantees for all inputs N.

We next see the “Miller-Rabin” algorithm Which will in addition detect **Carmichael** numbers

In your homework, you will design another general purpose primality test for all composite numbers.
Modular Square Roots Theorem

Theorem: If $N$ is prime, then the equation $X^2 = 1 \mod N$ has only two solutions in $\mathbb{Z}_N$: $x = 1 \mod N$ and $x = -1 \mod N$.

Proof: Suppose $a$ is such that $a^2 = 1 \mod N$. Then, $(a+1)(a-1) = a^2 - 1 = 0 \mod N$.

Since $(a+1)(a-1)$ is a multiple of $N$, and $N$ is a prime, then either $(a+1)$ or $(a-1)$ is a multiple of $N$, which means that either $a = 1 \mod N$ or $a = -1 \mod N$.
Using the Theorem to get a Primality Test

New Idea for a Primality test:
On input $N$, look for $a$ such that $a^2 \equiv 1 \text{ mod } N$ but $a \neq 1 \text{ (mod } N)$ and $a \neq -1 \text{ (mod } N)$.
If found it, then $a$ is witness that $N$ is composite

This will lead to the Miller-Rabin primality test which will work for all inputs $N$
Different Witnesses that $N$ is non prime

An $1 < a < N$ such that either

1. $\gcd(a, N) > 1$ implies $N$ has non trivial factors.

2. $a^{N-1} \neq 1 \mod N$ implies violation of Fermat’s theorem.

3. $a^2 = 1 \mod N$ but $a \neq 1, -1$ implies $1$ has more than two square roots in $Z_N^*$. 
How to Find Square Roots of 1 different from 1 and -1 mod N

• **Idea:** Take any \( a \) such that \( a^{N-1} \mod N = 1 \) (always when \( N \) is Carmichael numbers), and use this equation to find roots of 1 as follows

Write \( N-1 = 2^s t \) where \( t \) is odd. Compute successive square roots of 1

\[ a^{N-1} \mod N, \ a^{(N-1)/2} \mod N, \ a^{(N-1)/4} \mod N, \ldots \ a^{(N-1)/2^s} \mod N \]

\[ 1' \quad 1' \quad 1' \quad 1 \quad \text{or} \quad x \neq 1, -1 \]

**Must Prove:** \( N \) Carmichael \( \Rightarrow \) \( \text{Prob}_a[\text{hit non } 1, -1 \text{ root}] > 1/2 \)
Miller Rabin Primality Test

- **Input**: \( N > 2 \) odd s.t. \( N - 1 = 2^t \) such that \( t \) is odd.

- **Output**: “probably prime” or “composite”
  - if \( N = a^b \), \( a, b > 1 \), output “composite” (found witness)
  - Choose random integer \( a, 1 < a \leq N - 1 \)
    - if \( \gcd(a, N) \neq 1 \), output `composite’ (found witness)
    - if \( a^{N-1} \neq 1 \) output `composite’ (found witness)
    - else compute the sequence \( a^{(N-1)/2}, \ldots, a^{(N-1)/2^S} \mod N \)
      - if \( \forall y \) in the sequence \( y = 1 \) or \(-1\), output “\( N \) probably prime”,
      - else output “\( N \) composite” (found witness)

Works unless \( N \) is a perfect power, test for perfect power.
Correctness

• If \( n \) is a prime, no matter how we choose \( a \), algorithm always says `probably prime’ since can never find \( a \) s.t. \( a^{N-1} \neq 1 \mod N \) or a non-trivial root of 1.

• If \( n \) is composite, need to prove there are many choices of \( a \) for which algorithm output `composite’

Show: \( \text{Prob}_a [\text{Output “composite”}] \geq 1/2 \)
Error Analysis: Fix composite N

• Consider the set B of bad choices of a such that Miller-Rabin say that N is prime when the random choice a is made

• We want to prove $B \leq (N-1)/2$. We do it by proving that B is always contained in a proper sub-group of $\mathbb{Z}_N^*$ and therefore $|B| \leq 1/2|\mathbb{Z}_N^*|

$$B = \{ a: a^{N-1/2J} = +1 \mod N \ \forall J=1,...,s, \ OR \ a^{N-1} = +1 \mod N \ \& \ \exists J \leq s: a^{N-1/2J} = -1 \mod N \}$$
Size of B, first case: N is not a Charmichael number
(same analysis as for Fermat’s Test)

- There is some \( a \in \mathbb{Z}_N^* \) such that \( a^{N-1} \neq 1 \mod N \)

- Define the set \( G = \{ a \text{ s.t. } a^{N-1} = 1 \mod N \} \)
  This is a subgroup of \( \mathbb{Z}_N^* \) (check!) and it is a proper subgroup (N not Charmichael case).

As \(|G|\) divides \(|\mathbb{Z}_N^*|\) (Lagrange) and smaller than it, \(|G| \leq (N-1)/2\) and \( B \subseteq G \).
Size of B, second case: N is a Charmichael number (Harder)

Let \( N-1 = t 2^s \).

Consider the equations:
\[
x^{N-1} \mod N, \ x^{(N-1)/2} \mod N, \ x^{(N-1)/4} \mod N \ldots x^{(N-1)/2^s} \mod N
\]
Let \( u \) be the first exponent for which \( \exists x \ \text{s.t.} \ x^u \mod N = -1 \).

Claim 1: \( u \) exists (since \( t \) is odd and \((-1)^{(N-1)/2^s} = (-1)^t = -1\))

Claim 2: \( S^u = \{ a: a^u \mod N = -1 \text{ or } +1 \} \) is a proper subgroup of \( Z_N^* \) and \( B \subseteq S^u \)

Proof: \( S^u \) is a sub-group (check), \( B \subseteq S^u \) (check).

Need to show \( \exists w \notin S^u \) for it to be proper and thus at
\[
|S^u| \text{ most } 1/2 \text{ of } Z_N^*.
\]
\Rightarrow \text{Prob}( \ a \text{ chosen by algorithm in } B) \leq 1/2 \quad \text{QED}
Breaking News 2005: Primes is in P

- Manindra Agrawal, Neeraj Kayal, Nitin Saxena, India Institute of Technology, Kanpur:

  Actually, a De-randomization of a probabilistic Algorithm!
Las Vegas Randomized Algorithms

• Can generate uniformly at random an
  \( r \in \{0, \ldots, R\} \) (i.e flip coins)

• The running time on input \( x \), becomes a
  random variable \( Time(x, r) \) depending on \( r \)

• **Expected Polynomial Time** :
  \[
  T(n) = E_r[Time(x, r)] \text{ for any } x \text{ of length } n \text{ is bounded by a polynomial function in } n
  \]

• **Always correct** (for all input, prob (error)=0)
Ex 1: Quicksort

- Divide and conquer algorithm but the work is in divide rather than in combine.
- Sorts “in place” (like insertion sort, but not like merge sort).
- Very practical (with tuning).
Main Idea: Partition Around a Pivot

Quicksort an $n$-element array $A$:

**Divide:**

1. Pick a pivot element $x$ in $A$
2. Partition the array into sub-arrays $L(\text{elements}<x)$, $E(\text{elements}=x)$, $G(\text{elements}>x)$

**Conquer:** Recursively sort sub-arrays $L$ and $G$

**Combine:** Trivial.
How do you Choose a Pivot $x$?

**Basic Quick Sort:**

$x = A[1]$

Can Show: worst case $O(n^2)$ time

**Randomized Quick Sort:**

$x$ is chosen at random from the array $A$

(recursively each time a random choice)

Show: Expected $O(n \log n)$ for all inputs.

Randomness gives us more control over runtime.
How to Partition: Idea

• **Partition an input sequence as follows:**
  – Remove, in turn, each element $y$ from $A$ and
  – Insert $y$ into $L$, $E$ or $G$, depending on the result of the comparison with the pivot $x$

• Each insertion and removal takes $O(1)$ time

• Thus, the partition step takes $O(n)$

• To do this in place: tricky code (in slides/book, skip in lecture)
Partitioning subroutine

\textbf{Partition}(A, p, r) \triangleright A[p \ldots r]

\texttt{x} \leftarrow A[p] \quad \triangleright \text{pivot} = A[p]

i \leftarrow p

\textbf{for} j \leftarrow p + 1 \textbf{to} r \\
\textbf{do if} A[j] \leq x \\
\quad \textbf{then} \quad i \leftarrow i + 1 \\
\quad \text{exchange} \ A[i] \leftrightarrow A[j]

exchange \ A[p] \leftrightarrow A[i]

\textbf{return} \ i

\textbf{Invariant:} \quad \begin{array}{|c|c|c|c|}
\hline
x & \leq x & \geq x & ? \\
\hline
p & i & j & r \\
\hline
\end{array}

Already looked at \quad \text{Not looked at yet}
Example of partitioning

```
6 10 13 5 8 3 2 11
```

\[ i \quad j \]
Example of partitioning

\[ \begin{array}{cccccccc}
6 & 10 & 13 & 5 & 8 & 3 & 2 & 11 \\
\end{array} \]

\[ i \quad \rightarrow \quad j \]
Example of partitioning
Example of partitioning

\[
\begin{array}{cccccccc}
6 & 10 & 13 & 5 & 8 & 3 & 2 & 11 \\
6 & 5 & 13 & 10 & 8 & 3 & 2 & 11 \\
\end{array}
\]
Example of partitioning
Example of partitioning

<table>
<thead>
<tr>
<th>6</th>
<th>10</th>
<th>13</th>
<th>5</th>
<th>8</th>
<th>3</th>
<th>2</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>5</td>
<td>13</td>
<td>10</td>
<td>8</td>
<td>3</td>
<td>2</td>
<td>11</td>
</tr>
</tbody>
</table>

\[ i \quad \rightarrow \quad j \]
Example of partitioning

6 10 13 5 8 3 2 11

6 5 13 10 8 3 2 11

6 5 3 10 8 13 2 11

\[ i \quad j \]
Example of partitioning

\[
\begin{array}{cccccccc}
6 & 10 & 13 & 5 & 8 & 3 & 2 & 11 \\
6 & 5 & 13 & 10 & 8 & 3 & 2 & 11 \\
6 & 5 & 3 & 10 & 8 & 13 & 2 & 11 \\
\end{array}
\]
Example of partitioning
Example of partitioning

<table>
<thead>
<tr>
<th>6</th>
<th>10</th>
<th>13</th>
<th>5</th>
<th>8</th>
<th>3</th>
<th>2</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>5</td>
<td>13</td>
<td>10</td>
<td>8</td>
<td>3</td>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>3</td>
<td>10</td>
<td>8</td>
<td>13</td>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>8</td>
<td>13</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>

$i$ \rightarrow $j$
Example of partitioning

\[
\begin{array}{ccccccccc}
6 & 10 & 13 & 5 & 8 & 3 & 2 & 11 \\
6 & 5 & 13 & 10 & 8 & 3 & 2 & 11 \\
6 & 5 & 3 & 10 & 8 & 13 & 2 & 11 \\
6 & 5 & 3 & 2 & 8 & 13 & 10 & 11 \\
\end{array}
\]
Example of partitioning

6  10  13  5  8  3  2  11

6  5  13  10  8  3  2  11

6  5  3  10  8  13  2  11

6  5  3  2  8  13  10  11

2  5  3  6  8  13  10  11

i
Pseudocode for basic quicksort

QUICKSORT(A, i, j)
  if i < j
    then q ← PARTITION(A, i, j)
    QUICKSORT(A, i, q–1)
    QUICKSORT(A, q+1, j)

Initial call: QUICKSORT(A, 1, n)
Analysis of quicksort

• Assume all input elements are distinct.
• In practice, there are better partitioning algorithms for when duplicate input elements may exist.
Worst-Case Running Time

- The worst case for deterministic quick-sort occurs when the array is sorted or reverse sorted and the pivot is the unique minimum or maximum element.
- One of $L$ and $G$ has size $n - 1$ and the other has size 0.
- The running time is proportional to the sum $n + (n - 1) + \ldots + 2 + 1$.
- Thus, the worst-case running time of quick-sort is $O(n^2)$. 
Randomized Quicksort

• Partition around a *random* element as a pivot. I.e., around $A[t]$, where $t$ chosen uniformly at random from $\{p \ldots r\}$

• Every recursive call choose a pivot at random independently.

• We will show that the *expected* time is $O(n \log n)$ for all input arrays $A$
Probability Review

• **Sample Spaces:** universe of outcomes $\Omega$ which is finite sets, with a probability function $p(i) \geq 0$ s.t. $\sum_{i \in \Omega} p(i) = 1$.

Ex: choosing a random pivot. $\text{pr}[\text{pivot}=i] = \frac{1}{n}$

• **Events:** Subset of sample space.

$\text{prob}[\text{event}] = \sum p(i) = 1$.

Ex: prob [a random pivot in $\Omega$ gives 1/4:3/4 split] = $\frac{1}{2}$
Probability Review

• **Random Variables**: a function $X: \Omega \rightarrow \mathbb{R}$
  - **Ex**: $T(n)$ running time of Quick Sort algorithm
  - **Ex**: $S =$ size of sub-array passed in recursive call.

• **Expectation of Random Variable**: average value of $X$ weighted by the probability
  $$E[X] = \sum_{i \in \Omega} X(i)p(i)$$
  - **Ex**: how big on average is the first sub-array passed in randomized quick sort with random inputs?
Probability Review

• **Linearity of Expectations:** Let $X_1,..X_k$ be random variables defined over $\Omega$. Then, $E[\Sigma X_i]=\Sigma[E[X_i]]$

True for all random variables

**Independent**

• **Events:** We say that $A$ and $B$ are independent events $\subseteq \Omega$ if $\text{prob}[A \text{ and } B]=\text{prob}[A] \times \text{prob}[B]$.

Check $\Rightarrow \text{prob}[A|B]=\text{prob}[A]$ and $\text{prob}[B|A]=\text{prob}[A]$.

• **Random Variables:** $X$ and $Y$ are independent if $\text{prob}[X=a \text{ and } Y=b]=\text{prob}[X=a] \times \text{prob}[X=b]$

**Claim:** $E[X \times Y]=E[X] \times E[Y]$ if $X$ and $Y$ are independent.
Analysis of Quick Sort method #1: Indicator random variables

Sample Space = all possible sequence-of-pivots that Quick Sort could choose in its execution.

Let $T(n) =$ the random variable denoting the running time of randomized Quicksort on an input list of size $n \approx$ number of comparisons that Quick Sort makes.

Recall: $T(n): \Omega \Rightarrow R$. A choice for a sequence – of-pivots, yields a running time.

Goal: compute $E[T(n)]$
Analysis of Quick Sort method #1: Indicator random variables

For $k = 0, 1, \ldots, n-1$, define the *indicator random variable* $X_k$ as:

$$X_k = \begin{cases} 
1 & \text{if PARTITION generates a } k : n-k-1 \text{ split}, \\
0 & \text{otherwise.} 
\end{cases}$$

<table>
<thead>
<tr>
<th>$X_0$</th>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$X_3$</th>
<th>$X_{k-1}$</th>
<th>$X_k$</th>
<th>$X_{k+1}$</th>
<th>$X_{n-3}$</th>
<th>$X_{n-2}$</th>
<th>$X_{n-1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>..</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

$k$ and $n-k-1$
Great expectations: \( E[X_k], E[T(n)] \).

- Can use random variables to calculate expectations
- Expected value of indicator random variable:
  \[
  E[X_k] = 1 \cdot \Pr\{X_k=1\} + 0 \cdot \Pr\{X_k=0\} \\
  = 1 \cdot \left(\frac{1}{n}\right) + 0 \cdot \left(\frac{n-1}{n}\right) \\
  = \frac{1}{n}
  \]
- Since all splits are equally likely, assuming elements are distinct.
- Can use \( E[X_k] \) to calculate \( E[T(n)] \)
The power of indicator random variable

$$T(n) = \begin{cases} 
T(0) + T(n-1) + \Theta(n) & \text{if } 0:n-1 \text{ split}, \\
T(1) + T(n-2) + \Theta(n) & \text{if } 1:n-2 \text{ split}, \\
& \cdots \\
T(n-1) + T(0) + \Theta(n) & \text{if } n-1:0 \text{ split}, 
\end{cases}$$

$$= \sum_{k=0}^{n-1} X_k (T(k) + T(n - k - 1) + \Theta(n)).$$

- Summarize all \(n\) cases in a single expression using \(X_k\).
- Sum selects the \(X_k\) where the split happens (\(X_k=1\))
Calculating expectation

\[ E[T(n)] = E \left[ \sum_{k=0}^{n-1} X_k (T(k) + T(n - k - 1) + \Theta(n)) \right] \]

Take expectations of both sides.
Calculating expectation

\[ E[T(n)] = E\left[ \sum_{k=0}^{n-1} X_k (T(k) + T(n - k - 1) + \Theta(n)) \right] \]

\[ = \sum_{k=0}^{n-1} E[X_k (T(k) + T(n - k - 1) + \Theta(n))] \]

Linearity of expectation.
Calculating expectation

\[
E[T(n)] = E\left[ \sum_{k=0}^{n-1} X_k (T(k) + T(n - k - 1) + \Theta(n)) \right]
\]

\[
= \sum_{k=0}^{n-1} E[X_k (T(k) + T(n - k - 1) + \Theta(n))]
\]

\[
= \sum_{k=0}^{n-1} E[X_k] \cdot E[T(k) + T(n - k - 1) + \Theta(n)]
\]

Independence of \(X_k\) from other random choices in the recursive calls.
Calculating expectation

\[ E[T(n)] = E\left[ \sum_{k=0}^{n-1} X_k (T(k) + T(n - k - 1) + \Theta(n)) \right] \]

\[ = \sum_{k=0}^{n-1} E[X_k] \cdot E[T(k) + T(n - k - 1) + \Theta(n)] \]

\[ = \frac{1}{n} \sum_{k=0}^{n-1} E[T(k)] + \frac{1}{n} \sum_{k=0}^{n-1} E[T(n - k - 1)] + \frac{1}{n} \sum_{k=0}^{n-1} \Theta(n) \]

Linearity of expectation; \( E[X_k] = 1/n \).
Calculating expectation

\[
E[T(n)] = E\left[ \sum_{k=0}^{n-1} X_k (T(k) + T(n - k - 1) + \Theta(n)) \right]
\]

\[
= \sum_{k=0}^{n-1} E[X_k] \cdot E[T(k) + T(n - k - 1) + \Theta(n)]
\]

\[
= \frac{1}{n} \sum_{k=0}^{n-1} E[T(k)] + \frac{1}{n} \sum_{k=0}^{n-1} E[T(n - k - 1)] + \frac{1}{n} \sum_{k=0}^{n-1} \Theta(n)
\]

\[
= \frac{2}{n} \sum_{k=1}^{n-1} E[T(k)] + \Theta(n)
\]

Summations have identical terms.
Hairy recurrence

\[ E[T(n)] = 2 \sum_{k=2}^{n-1} E[T(k)] + \Theta(n) \]

(The \( k = 0, 1 \) terms can be absorbed in the \( \Theta(n) \).)

Prove: \( E[T(n)] \leq a n \lg n \) for constant \( a > 0 \)

By the substitution method.

Substitute Induction hypothesis

\[ E[T(n)] \leq 2 \sum_{k=2}^{n-1} ak \lg k + \Theta(n) \]
Analysis method #1 summary

• Defined indicator random variable $X_k$, marking the partition point for $k:n-k-1$ split.

• Expressed running time $T(n)$ (rand. var.) as a function of this indicator random variable.

• Calculated the expected value of $E[T(n)]$ using properties of $E[X_k]$.

• Prove $E[T(n)] \leq an \log n$ by induction using the substitution method

$\Rightarrow$ Quicksort expected running time $O(n \log n)$. 
Analysis Method #2: What do we want from the Pivot

• Hope: Always the median? Too much to hope

• But, say we get say 1/4:3/4 split, that would yield $O(n \log n)$ runtime.

• Since half the elements will give a split of (at worst) 1/4:3/4, the probability of picking such an element as a pivot is $>1/2$

• This intuition leads to another analysis
“Paranoid” quicksort: Different Analysis method

• Let’s modify the quick sort algorithm to make it easier to analyze by another method:
  
  • Repeat:
    • Choose the pivot to be a random element of the array
    • Perform \textsc{Partition}
    • Until the resulting partition will cause a “good” recursive call: the sizes of both L and G are no larger than $\frac{3}{4}$ (size of the array)
    • Recurse on both sub-arrays
Expected Running Time Analysis for paranoid quick sort

- Consider a recursive call of quick-sort on a sequence of size $n$.

- Define
  - **Good call:** the sizes of $L$ and $G$ are each less than $3n/4$  
  - **Bad call:** one of $L$ and $G$ has size greater than $3n/4$

- **Claim:** A call is good with probability $1/2$
- **Proof:** 1/2 of the possible pivots cause good calls:
Analysis

- Let $T(n)$ be an upper bound on the expected running time on any array of $n$ elements.
- Consider any input of size $n$.
- The time needed to sort the input is bounded from above by a sum of
  - The time needed to sort the left subarray
  - The time needed to sort the right subarray
  - The number of iterations till we get a good call * $cn$ (Cost of partition)
Expectations

Therefore:

\[ T(n) \leq \max_i (T(i) + T(n - i)) + E[\# \text{ iterations}] \cdot cn \]

where maximum is taken over \( i \in [n/4, 3n/4] \)

• We showed \( \Pr[\text{good call}] > 1/2 \Rightarrow E[\#\text{iterations}] \) is \( \leq 2 \)

• So:

\[ T(n) \leq \max_i (T(i) + T(n - i)) + 2cn \quad , \quad i \in [n/4, 3n/4] \]
Final bound

• Use a recursion tree argument:
  • Tree depth is $\Theta(\log n)$
  • Total cost at each level is at most $2cn$
  • Overall $T(n) = O(n \log n)$

• Show (exercise): if Paranoid Quick Sort is Expected to run in $O(n \log n)$, so is Randomized Quick Sort.