Today

• Randomized algorithms: algorithms that flip coins
• Two examples:
  – Matrix product checker: is $AB = C$?
  – Quicksort:
    • Example of divide and conquer
    • Fast and practical sorting algorithm
    • Other applications on Wednesday
**Matrix Product Checker**

- Given: \( n \times n \) matrices \( A, B, C \)
- Goal: is \( A \times B = C \) ?
- We will see an \( O(n^2) \) algorithm that:
  - If answer = YES, then \( \Pr[output = YES] = 1 \)
  - If answer = NO, then \( \Pr[output = YES] \leq \frac{1}{2} \)

**The algorithm**

- Algorithm:
  - Choose a random binary vector \( x[1…n] \), such that \( \Pr[x_i=1] = \frac{1}{2}, i=1…n \)
  - Check if \( ABx = Cx \)
- Does it run in \( O(n^2) \) time?
  - YES, because \( ABx = A(Bx) \)

**Correctness**

- Let \( D = AB \), need to check if \( D = C \)
- What if \( D = C \) ?
  - Then \( Dx = Cx \), so the output is YES
- What if \( D \neq C \) ?
  - Presumably there exists \( x \) such that \( Dx \neq Cx \)
  - We need to show there are many such \( x \)

**D ≠ C**

- Vector product
  - Consider vectors \( d \neq c \) (say, \( d_i \neq c_i \))
  - Choose a random binary \( x \)
  - We have \( dx = cx \) iff \( (d-c)x = 0 \)
  - \( \Pr[(d-c)x = 0] = ? \)

\[
\begin{align*}
(d-c): & \quad d_1 - c_1 \quad \ldots \quad d_i - c_i \\
x: & \quad x_1 \quad x_2 \quad \ldots \quad x_i \quad = \sum_{j \neq i} (d_j - c_j) x_j + (d_i - c_i) x_i
\end{align*}
\]

**Analysis, ctd.**

- If \( x_i = 0 \), then \( (c-d)x = S_1 \)
- If \( x_i = 1 \), then \( (c-d)x = S_2 \neq S_1 \)
- So, \( \geq 1 \) of the choices gives \( (c-d)x \neq 0 \)
  - \( \rightarrow \Pr[cx = dx] \leq \frac{1}{2} \)
Matrix Product Checker

- Is $A \times B \times C$?
  - We have an algorithm that:
    - If answer= YES, then $\Pr[\text{output}=\text{YES}] = 1$
    - If answer= NO, then $\Pr[\text{output}=\text{YES}] \leq \frac{1}{2}$
  - What if we want to reduce $\frac{1}{2}$ to $\frac{1}{4}$?
    - Run the algorithm twice, using independent random numbers
    - Output YES only if both runs say YES
  - Analysis:
    - If answer= YES, then $\Pr[\text{output}_1=\text{YES}, \text{output}_2=\text{YES}] = 1$
    - If answer= NO, then $\Pr[\text{output}=\text{YES}] = \Pr[\text{output}_1=\text{YES}, \text{output}_2=\text{YES}] = \Pr[\text{output}_1=\text{YES}] \cdot \Pr[\text{output}_2=\text{YES}] \leq \frac{1}{4}$

Quicksort

- Divide-and-conquer algorithm.
- Sorts “in place” (like insertion sort, but not like merge sort).
- Very practical (with tuning).
- Can be viewed as a randomized Las Vegas algorithm

Divide and conquer

Quick sort an $n$-element array:

1. **Divide:** Partition the array into two subarrays around a pivot $x$ such that elements in lower subarray $\leq x \leq$ elements in upper subarray.

2. **Conquer:** Recursively sort the two subarrays.
3. **Combine:** Trivial.

**Key:** Linear-time partitioning subroutine.

Pseudocode for quicksort

```
QUICKSORT(A, p, r)
if p < r
    then q ← PARTITION(A, p, r)
    QUICKSORT(A, p, q-1)
    QUICKSORT(A, q+1, r)
```

Initial call: `QUICKSORT(A, 1, n)`

Partitioning subroutine

```
PARTITION(A, p, r) ▷ A[p . . r]
    i ← p
    for j ← p + 1 to r
        do if A[j] ≤ x
            then i ← i + 1
        return i
```

Invariant:

```
\text{\begin{tabular}{c}
\hline
\text{p} & \text{i} & \geq x \text{?} \\
\hline
\end{tabular}}
```

Example of partitioning

```
\begin{tabular}{ccccccccc}
6 & 10 & 13 & 5 & 8 & 3 & 2 & 11 \\
\hline
\end{tabular}
```

```
\begin{tabular}{cc}
i & j \\
\hline
\end{tabular}
```
Example of partitioning

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

\[
\begin{align*}
i & \quad j
\end{align*}
\]
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

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

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

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

Analysis of quicksort

- Assume all input elements are distinct.
- In practice, there are better partitioning algorithms for when duplicate input elements may exist.
- What is the worst case running time of Quicksort?

\[ x \leq x \geq x \]
Worst-case of quicksort

- Input sorted or reverse sorted.
- Partition around min or max element.
- One side of partition always has no elements.

\[ T(n) = T(0) + T(n-1) + \Theta(n) \]
\[ = \Theta(1) + T(n-1) + \Theta(n) \]
\[ = T(n-1) + \Theta(n) \]
\[ = \Theta(n^2) \quad \text{(arithmetic series)} \]
### Worst-case recursion tree

\[ T(n) = T(0) + T(n-1) + cn \]

- \[ T(0) \]
- \[ T(n-1) \]
- \[ cn \]
- \[ \Theta(1) \]
- \[ \Theta(n) \]

\[ \Theta \left( \sum_{k=1}^{n} k \right) = \Theta(n^2) \]

### Analysis of nice case

If we’re lucky, PARTITION splits the array evenly:

\[ T(n) = 2T(n/2) + \Theta(n) \]

\[ = \Theta(n \log n) \quad \text{(same as merge sort)} \]

What if the split is always \(\frac{9}{10}\)?

\[ T(n) = T(\frac{n}{10}) + T(\frac{n}{10}) + \Theta(n) \]
Analysis of nice case

\[ T(n) \leq cn \log_{10/9} n + \Theta(n) \]

More intuition

Suppose we alternate lucky, unlucky, lucky, unlucky, lucky, ...,

\[ L(n) = 2U(n/2) + \Theta(n) \quad \text{lucky} \]
\[ U(n) = L(n-1) + \Theta(n) \quad \text{unlucky} \]

Solving:

\[ L(n) = 2(L(n/2 - 1) + \Theta(n/2)) + \Theta(n) \]
\[ = 2L(n/2 - 1) + \Theta(n) \]
\[ = \Theta(n \log n) \quad \text{Lucky!} \]

How can we make sure we are usually lucky?

Randomized Algorithms

- Algorithms that make decisions based on random coin flips.
- Can “fool” the adversary.
- The running time (or even correctness) is a random variable; we measure the expected running time.
- We assume all random choices are independent.
- This is not the average case!

Randomized quicksort

- Partition around a random element. I.e., around \( A[t] \), where \( t \) chosen uniformly at random from \{p...r\}
- We will show that the expected time is \( \Theta(n \log n) \)

Analysis method #1: “Paranoid” quicksort

- Will modify the algorithm to make it easier to analyze:
  - Repeat:
    - Choose the pivot to be a random element of the array
    - Perform \textsc{Partition}
  - Until the resulting split is “lucky”, i.e., not worse than 1/10: 9/10
  - Recurse on both sub-arrays
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 the 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 until we get a lucky split, times $cn$

Expectations

- Therefore:
  $$T(n) \leq \max T(i) + T(n-i) + E[\text{\# partitions}] \cdot cn$$
  where maximum is taken over $i \in [n/10, 9n/10]$
- We will show that $E[\text{\# partitions}]$ is $\leq 10/8$
- Therefore:
  $$T(n) \leq \max T(i) + T(n-i) + 10/8 \cdot cn, i \in [n/10, 9n/10]$$

Final bound

- Can use the recursion tree argument:
  - Tree depth is $\Theta(\log n)$
  - Total cost at each level is at most $10/8 \cdot cn$
  - Overall $T(n) = \Theta(n \log n)$

Lucky partitions

- The probability that a random pivot induces a lucky partition is at least $8/10$
  (we are *not* lucky if the pivot happens to be among the smallest/largest $n/10$ elements)
- If we flip a coin, with heads prob. $p=8/10$, the expected waiting time for the first head is $1/p = 10/8$

Analysis method #2: Indicator variables

Let $T(n)$ = the random variable for the running time of randomized quicksort on an input of size $n$, assuming random numbers are independent.

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

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

$$X_0 \quad X_1 \quad X_2 \quad X_3 \quad X_{n-3} \quad X_{n-2} \quad X_{n-1}$$

$$\begin{array}{ccccccc}
0 & 0 & 0 & 0 & \cdots & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0
\end{array}$$

Great expectations: $E[X_k], E[T(n)]$

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

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

= \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

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

Take expectations of both sides.

Calculating expectation

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

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

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

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

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

Calculating expectation

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

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

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

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

= \sum_{k=0}^{n-1} \mathbb{E}[T(k)] + \Theta(n) \quad \text{Summations have identical terms.}
Hairy recurrence

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

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

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

• Choose \( a \) large enough so that \( an \lg n \) dominates \( E[T(n)] \) for sufficiently small \( n \geq 2 \).

**Use fact:** \( \sum_{k=2}^{n-1} k \lg k \leq \frac{1}{2} n^2 \lg n - \frac{n}{8} \) (exercise).

Substitution method

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

Substitute inductive hypothesis.

Substitution method

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

\[ \leq \frac{2a}{n} \left( \frac{1}{2} n^2 \lg n - \frac{n}{8} \right) + \Theta(n) \]

\[ = an \lg n - \left( \frac{an}{4} - \Theta(n) \right) \]

Express as desired – residual.

Substitution method

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

= \( \frac{2a}{n} \left( \frac{1}{2} n^2 \lg n - \frac{n}{8} \right) + \Theta(n) \)

= \( an \lg n - \left( \frac{an}{4} - \Theta(n) \right) \)

if \( a \) is chosen large enough so that \( an/4 \) dominates the \( \Theta(n) \).

Analysis method #2 summary:

• Defined indicator random variable \( X_i \), marking the partition point for \( kn-k-1 \) 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 \( E[X_i] \).

• Used algebra to prove \( E[T(n)] \leq an \log n \) by induction using the substitution method

> Quicksort expected running time \( \Theta(n \log n) \).
Quicksort in practice

- Quicksort is a great general-purpose sorting algorithm.
- Quicksort is typically over twice as fast as merge sort.
- Quicksort can benefit substantially from code tuning.
- Quicksort behaves well even with caching and virtual memory.
- Quicksort is great!