CS 124 Section 0

Biased Coins, Math Review, Sorting/Searching

Lavanya Singh

2021

Goals

- CALCULATIONS
  - Sums and Series
  - Probability and Counting
- PROOFS
  - Induction
  - Contradiction
- BIG-O
- SORTING/SEARCHING
  - Binary Search
  - Mergesort

Calculations

Here are some important sums:

- $\sum_1^n i = n(n+1)/2$
- $\sum_1^n i^2 = n(n+1)(2n+1)/6$

Let's derive this formula:

$$\sum_{i=0}^{n} s^i = \frac{1 - s^{n+1}}{1 - s}$$

Useful Facts:

- $P(A) = 1 - P(A^c) = 1 - P(\bar{A})$
- If A and B are independent events, $P(A \text{ and } B) = P(A)P(B)$
  - independent events = knowing A gives you no information about B
- Expectation $E(X) = \sum xP(X = x)$
- Linearity of expectation: $E(X_1 + X_2) = E(X_1) + E(X_2)$

Recall the coin tossing algorithm from Lecture 0. We have a biased coin that is H with probability p and T with probability q. To convert it to an unbiased coin, we leverage symmetry. We consider each set of two flips, and we count HT as a fair heads, TH as a fair tails, and toss the rest.

Probability and Counting

Recall the coin tossing algorithm from Lecture 0. We have a biased coin that is H with probability p and T with probability q. To convert it to an unbiased coin we leverage symmetry. We consider each set of two flips, and we count HT as a fair heads, TH as a fair tails, and toss the rest.

What is the probability of getting a bit in the first 2 flips? Of not getting a bit in the first 2 flips?

Probability and Counting

Recall the coin tossing algorithm from Lecture 0. We have a biased coin that is H with probability p and T with probability q. To convert it to an unbiased coin, we leverage symmetry. We consider each set of two flips, and we count HT as a fair heads, TH as a fair tails, and toss the rest.

What is the probability of getting k bits from $2k$ flips?

Recall the coin tossing algorithm from Lecture 0. We have a biased coin that is H with probability p and T with probability q. To convert it to an unbiased coin we leverage symmetry. We consider each set of two flips, and we count HT as a fair heads, TH as a fair tails, and toss the rest.

What is the expected value of the number of bits retrieved in the first $2k$ flips?

Proofs

What is a Proof?

A proof is a mathematical argument. It should start with what the reader already knows and proceed in clear logical steps from there.

Proof Strategy: Induction

Use this if you want to show that a statement is true for all natural numbers n.

- Base case: Show that the statement is true for $n = 0$ or $n = 1$
- Inductive hypothesis: Assume that the statement is true for $n = k$.
- Inductive step: Show that the statement is true for $n = k + 1$

Why does this work?

Example Inductive Proof

The sum of the first n natural numbers is $n(n+1)/2$.

Example Inductive Proof

The sum of the first n natural numbers is $n(n+1)/2$.

Proof.

- Base case: $n = 0$. $0 = 0(0+1)/2$ so the statement holds.

□

Example Inductive Proof

The sum of the first n natural numbers is $n(n+1)/2$.

Proof.

- Base case: $n = 0$. $0 = 0(0+1)/2$ so the statement holds.
- Inductive hypothesis: Assume that the sum of the first $n = k$ numbers is $k(k+1)/2$.

□

Example Inductive Proof

The sum of the first n natural numbers is $n(n+1)/2$.

Proof.

- Base case: $n = 0$. $0 = 0(0+1)/2$ so the statement holds.
- Inductive hypothesis: Assume that the sum of the first $n = k$ numbers is $k(k+1)/2$.
- Inductive step: The sum of the first $n = k+1$ numbers is $k+1+k(k+1)/2$ by the inductive hypothesis. Doing some algebra, $k+1+k(k+1)/2 = (k^2+3k+2)/2 = (k+1)(k+2)/2$.

□

Proof by Contradiction

If you want to prove a statement is true, you can assume it is false and then show that its falsity leads to a contradiction.

Proof by Contradiction

If you want to prove a statement is true, you can assume it is false and then show that its falsity leads to a contradiction.

Consider the statement "there are infinitely many primes." It's much easier to think about a world with finitely many primes, than one with infinitely many.

Example Proof by Contradiction

There are infinitely many primes.

Proof.

Assume towards a contradiction that there are finitely many primes. Number the primes $p_1, p_2, \ldots ...p_n$. Now consider the number $N = 1 + p_1 * p_2 * \ldots * p_n$. $N > p_n$ so N must not be prime, so it must be divisible by some prime (since every number can be factored into primes). All the p_i divide $N - 1$ so none of them can divide N so there must be some different prime that divides N. This contradicts the earlier statement that the p_i were all the primes, so the assumption must be false and there must be infinitely many primes. □

Advice

DO:

Give us enough details to indicate that you haven't made any plausible errors

DON'T:

- Give an example instead of a proof
- Skip steps, assume too much
- Use inconsistent definitions

Big-O

Sometimes we don't care about the exact running time of our algorithms. Instead, we just care about their asymptotic behavior: what happens as the input sizes become arbitrarily large?

Consider two functions F and G. $F = O(G)$ if there exist two positive real numbers a and N such that, $\forall n > N, f(n) \leq ag(n)$.

Ex. $f(n) = n, g(n) = n^2 \longrightarrow f = O(g)$[1]

[1] $f = O(g)$ and $f \in O(g)$ may be used interchangeably

Big-Ω and Big-Θ

Consider two functions F and G. $F = \Omega(G)$ if there exist two positive real numbers a and N such that $\forall n > N \; f(n) \geq ag(n)$.

Ex. $f(n) = n, g(n) = n^2 \longrightarrow g = \Omega(f)$

$f = \Theta(g) \iff f = O(g)$ and $f = \Omega(g)$

Little-o

Consider two functions F and G. $F = o(G)$ if for every positive real number c, there exists a real number N such that $\forall n > N$, $|f(n)| \leq c|g(n)|$.

Equivalent definition: Consider two functions F and G. $F = o(G)$ if $\lim_{n \to \infty} f(n)/g(n) = 0$.

Sorting and Searching

Say I have a sorted array of integers A. How can I determine if A contains the integer 6?

Binary Search

Say I have a sorted array of integers *A*. How can I determine if *A* contains the integer 6?

Start at the middle. If the middle number is greater than 6 (maybe it's 8) then we're too high, and we should be looking at the smaller half of *A*, and vice versa if it is lower.

Repeat this procedure until you find 6, or realize that it's not in the array.

Binary Search

Let $T(n)$ be the number of comparisons needed to search for an element in an array of n elements.

$$T(n) = T(n/2) + c$$

$$T(n) = T(n/4) + 2c$$

$$\ldots$$

$$T(n) = T(n/2^{\log_2(n)}) + \log_2(n)c$$

$$T(n) = c \log_2 n$$

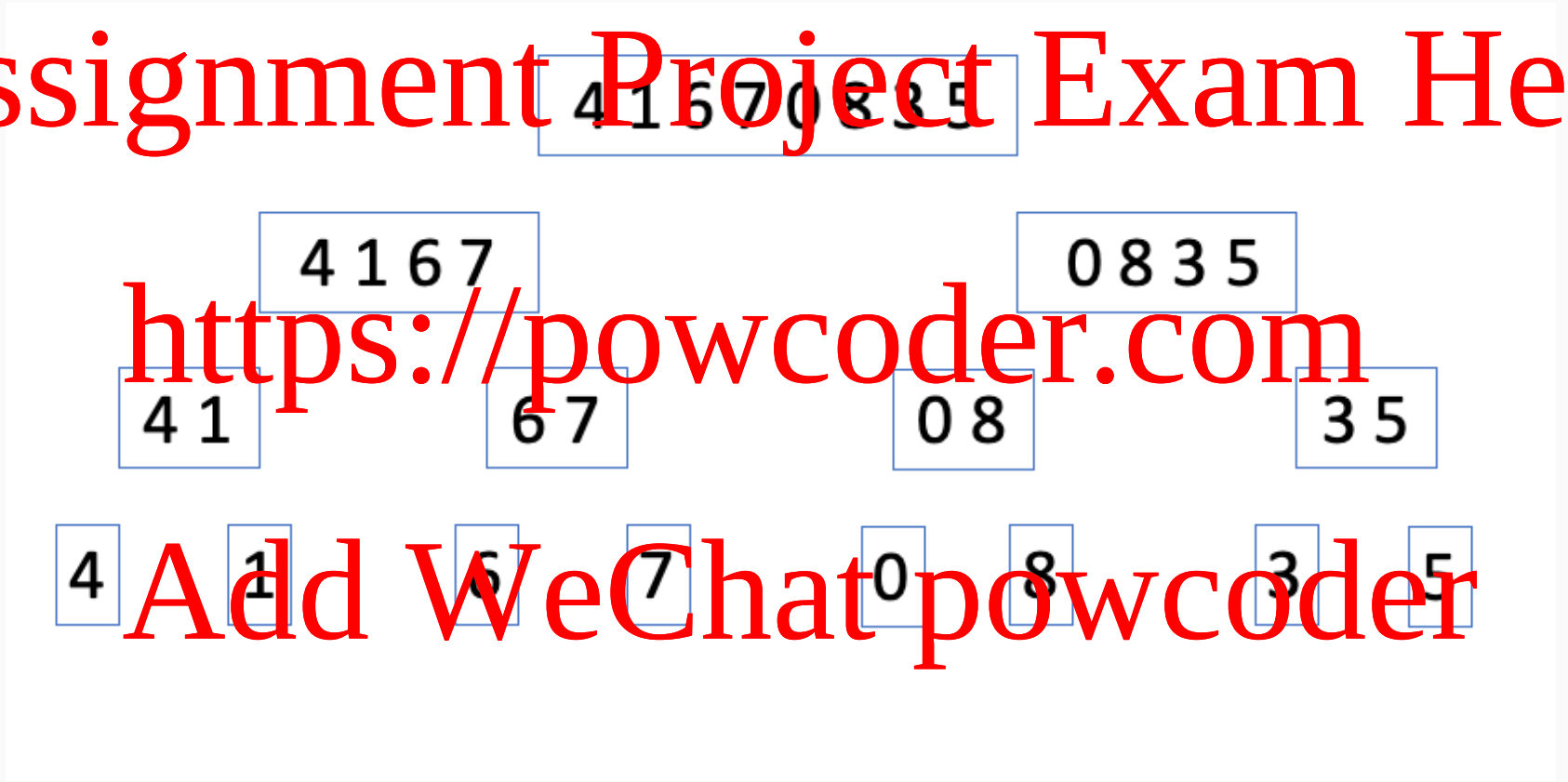
So binary search runs in time $O(\log_2 n)$

Mergesort

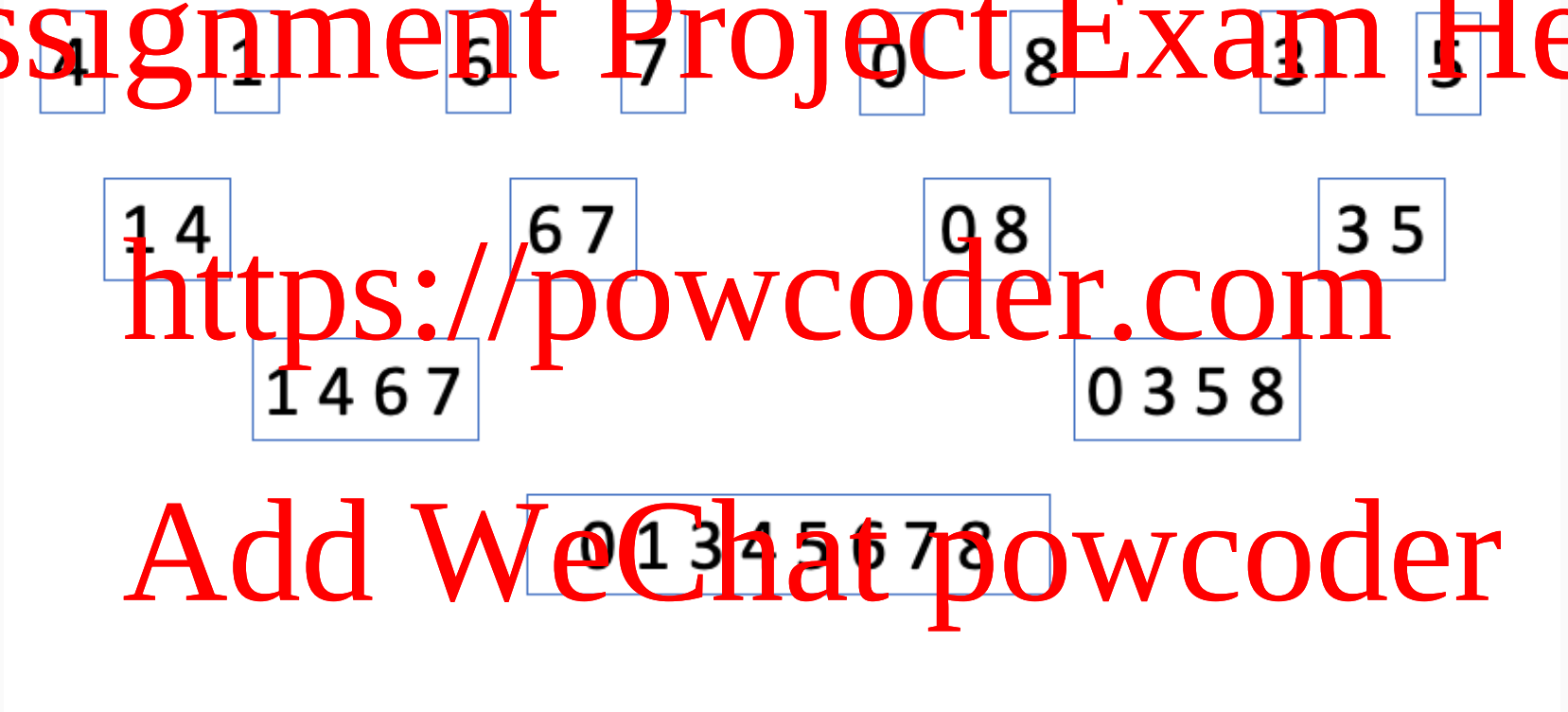
Sorting a list:

- Divide: Repeatedly divide the list in half
- Conquer: Combine each sublist, sorting as you go

Mergesort: Divide



Mergesort: Conquer



3-way Mergesort

Suppose instead of splitting the array in two 2 pieces during the divide steps, we split it in 3. Let's analyze the time complexity of this algorithm.

3-way Mergesort

Suppose instead of splitting the array in two 2 pieces during the divide steps, we split it in 3. Let's analyze the time complexity of this algorithm.

$$T(n) = 3T(n/3) + cn$$

3-way Mergesort

$$T(n) = 3T(n/3) + cn$$

$$T(n) = 3(3T(n/9) + cn/3) + cn = 9T(n/9) + 2cn$$

$$\ldots$$

$$T(n) = 3^{\log_3 n} T(n/3^{\log_3 n}) + cn \log_3 n$$

$$T(n) = O(n \log_3 n)$$

[2]

[2] Note that $\log_2$ is only a constant factor different from $\log_3$ so this is the same big-O bound as for the original mergesort algorithm.

Consider $T(n) = aT(n/b) + cn^k$.

$$\begin{cases} O(n^{\log_b a}) & a > b^k \\ O(n^k \log n) & a = b^k \\ O(n^k) & a < b^k \end{cases}$$

Section Problems

Problem 1

Assume arrays never have repeating elements. Here is some pseudocode for the sorting algorithm ***bogosort***:

```
def bogosort(A):
    while A is not sorted: shuffle A
```

- Say A has n elements. What is the probability that bogosort finds the correct solution on the 4th shuffle?
- Consider ***bogomergesort***. This algorithm is like ***mergesort***, except the ***merge*** function combines the two sublists by running ***bogosort*** until you have a correctly sorted, combined sublist. Write a recurrence for the expected number of comparisons required in ***bogomergesort***, solve it, and prove its correctness. *Hint: the expected number of comparisons for bogosort is n*n!*

Problem 2

Earlier, I said that the recurrence the number of comparison operations required for mergesort is $T(n) = 2T(n/2) + cn$. This was a lie!

- Play along with my lie. Assume n is a power of 2. Prove that mergesort takes $O(n \log n)$ comparisons by induction.
- Write a recurrence for the number of comparison operations when n is *not* a power of 2. As a bonus, try to solve it. *Hint: you may find the following concept useful: what is* $2^{\lceil \log_2 n \rceil}$*?*

Problem 3

For all of the problems below, when asked to give an example, you should give a function mapping positive integers to positive integers. (No cheating with zeroes!)

- Find (with proof) a function f_1 such that $f_1(2n)$ is $O(f_1(n))$.
- Find (with proof) a function f_2 such that $f_2(2n)$ is not $O(f_2(n))$.
- Prove that if $f(n)$ is $O(g(n))$, and $g(n)$ is $O(h(n))$, then $f(n)$ is $O(h(n))$.

- Give a proof or a counterexample: if f is not $O(g)$, then g is $O(f)$.