LECTURE 2

ASYMPTOTIC ANALYSIS:

Date _____

* Big O, small o, Omega Ω, Theta Θ notations.
* Recurrences. (focus on divide and conquer style recurrences).
* Solving recurrences by:
  - unrolling
  - guess and inductive proof
  - recursion tree
  - master theorem.

ASYMPTOTIC ANALYSIS: How does the running time scale with the size of the input.

→ ignore low order terms and constant factors.

→ focus on the shape of the runtime curve.

→ n denotes size of input.

→ T(n) denotes the runtime of algo on input size n.

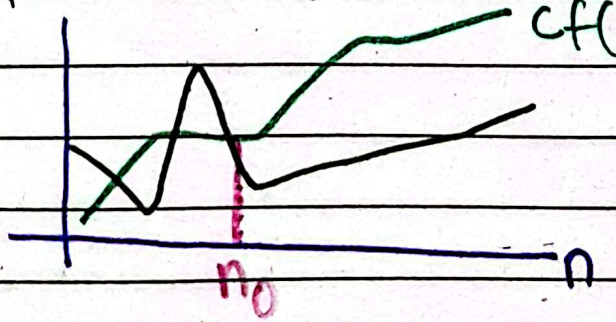
DEFINITION 1:

$T(n) \in O(f(n))$ if there exists constants $c, n_0 > 0$ such that $T(n) \leq cf(n)$ for all $n > n_0$.

EXPLANATIONS:

→ T(n) is proportional to f(n) or better as n gets large.

→ upper bound / worst case complexity / max time req.



Date _____

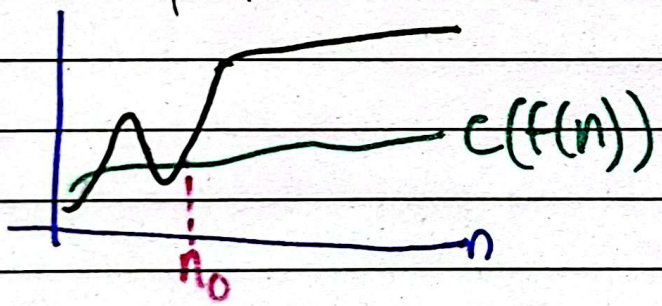
for eg: $3n^2+17 \in O(n^2)$ and $3n^2+17 \in O(n^3)$.
Multiple upper bounds can apply, but tighter ones are usually prefered. $O(n^2)$ is a tighter bound meaning it is closer to the actual growth rate. $O(n^3)$ is a valid but loose bound meaning that it is technically correct but not very infomative as it is far above the actual rate.

DEFINITION 2:

$T(n) \in \Omega(f(n))$ if there exists constants $c, n_0 > 0$ such that $T(n) \geq cf(n)$ for all $n > n_0$.

EXPLANATION:

→ lower bound / best case complexity
→ $T(n)$ is proportional to $f(n)$ or worse as n gets large.



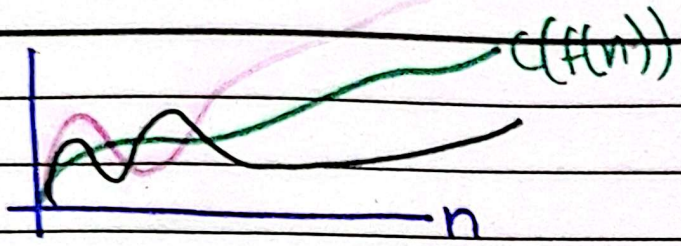
DEFINITION 3:

$T(n) \in \Theta(f(n))$ if $T(n) \in O(f(n))$ and $T(n) \in \Omega(f(n))$.

EXPLANATION:

→ $T(n)$ is proportional to $f(n)$ as n gets large.
→ encloses the function from above and below.
→ avg case complexity.
→ intuitively runtime is $f(n)$ upto some constant factors.

Date



DEFINITION 4:

$T(n) \in o(f(n))$ if for all constants $c > 0$, there exists $n_0 > 0$ such that $T(n) < cf(n)$ for all $n > n_0$.

$$O \leq ; \Omega \geq , \Theta = ; o <$$

In terms of computing whether or not $T(n)$ belongs to one of these sets with respect to $f(n)$, a convenient way is to use limits.

$$\lim_{n \to \infty} \frac{T(n)}{f(n)}$$

If the limit exists, we can make the following statements.

→ if limit is 0, then $T(n) = o(f(n))$ and $T(n) = O(f(n))$.

→ if limit is ∞, Then $T(n) = \omega(f(n))$ and $T(n) = \Omega(f(n))$.

→ if limit is a number greater than 0; then

Date _____

$T(n) = \Theta(f(n))$ ($T(n) = O(f(n))$ and $T(n) = \Omega(f(n))$).

e.g: $T(n) = 2n^3 + 100n^2 \log_2 n + 17$, $f(n) = n^3$.

$$\frac{T(n)}{f(n)} = \frac{2n^3}{n^3} + \frac{100n^2 \log_2 n}{n^3} + \frac{17}{n^3}$$

$$\frac{T(n)}{f(n)} = 2 + \frac{100 \log_2 n}{n} + \frac{17}{n^3}$$

$$\lim_{n \to \infty} \left(2 + \frac{100 \log_2 n}{n} + \frac{17}{n^3} \right)$$

limit goes to 2. $\rightarrow$ $2 + 0 + 0 = 2$.

$\frac{1}{n^k} = 0$ Therefore $T(n) = \Theta f(n)$.

Agar $n \to \infty$, and
n in denominator toh 0.

eg $\lim_{x \to \infty} \frac{2x^2 - 3x + 7}{x^2 + 47x + 1} \Rightarrow \frac{\frac{2x^2}{x^2} - \frac{3x}{x^2} + \frac{7}{x^2}}{\frac{x^2}{x^2} + \frac{47x}{x^2} + \frac{1}{x^2}}$

$\Rightarrow \frac{2 - \frac{3}{x} + \frac{7}{x^2}}{1 + \frac{47}{x} + \frac{1}{x^2}} \Rightarrow \frac{2 - 0 + 0}{1 + 0 + 0} \Rightarrow \frac{2}{1} \Rightarrow 2.$

(only work at leading terms of numerator & denominator). $\lim_{x \to \infty} \frac{2x^2}{x^2} = \frac{2}{1} = 2.$

Date

if limit does not exist eg $T(n) = n(2+\sin n)$

$f(n) = n$.

$\Rightarrow \frac{T(n)}{f(n)} = \frac{n(2+\sin n)}{n} = 2+\sin(n)$

This oscillates b/w 1 & 3. so limit approach fails.

In this case, prove by applying definition

$T(n) = \Theta f(n)$.

① Definition of Θ

$T(n) \in \Theta(f(n))$ if:

$\exists\ c_1, c_2, n_0 > 0$ s.t $c_1 \cdot f(n) \le T(n) \le c_2 \cdot f(n)$ $\forall n \ge n_0$

meaning that $T(n)$ must be sandwiched b/w two constant multiples of $f(n)$ for sufficiently large n.

② Substitute $T(n)$ & $f(n)$

$c_1 \cdot n \le n(2+\sin n) \le c_2 \cdot n$

③ Simplify

$T(n)$ can be simplified to $2n + n\sin n$.

④ Bounds on $\sin(n)$:

since $-1 \le \sin n \le 1$

This gives $-1 \times n \le n \times \sin n \le 1 \times n$

$-n \le n\sin n \le n$

and $-n+2n \le 2n + n\sin n \le n+2n$

$2n - n \le T(n) \le 2n+n$

$1n \le T(n) \le 3n$

⑤ $\hookrightarrow c_1 = 1$ & $c_2 = 3$

find constants c_1 & c_2.

Date

⑥ **conclusion**. since constants $c_1 = 1$ & $c_2 = 3$ exists, we have $T_n \in O(n) \Rightarrow T_n \in \Theta(f(n))$.

NOTE:

→ for any constant k

$$\lim_{n\to\infty} \frac{(\log n)^k}{n} \Rightarrow 0.$$

eg: $n\log n = O(n^{1.5})$

$T(n) = n\log n$

$f(n) = n^{1.5}$

$\lim_{n\to\infty} \frac{T_n}{f(n)}$.

$$\lim_{n\to\infty} \frac{n \log n}{n^{1.5}}$$

$$\lim_{n\to\infty} \frac{\log n}{n^{1/2}}$$

$$\lim_{n\to\infty} \frac{\log n}{\sqrt{n}}$$

$$\lim_{n\to\infty} \frac{\sqrt{(\log n)^2}}{(\sqrt{n})^2}$$

$$\lim_{n\to\infty} \sqrt{\frac{\log n^2}{n}}$$

$$\lim_{n\to\infty} \left(\frac{(\log n)^2}{n}\right)^{1/2}$$

$$\lim_{n\to\infty} = 0.$$

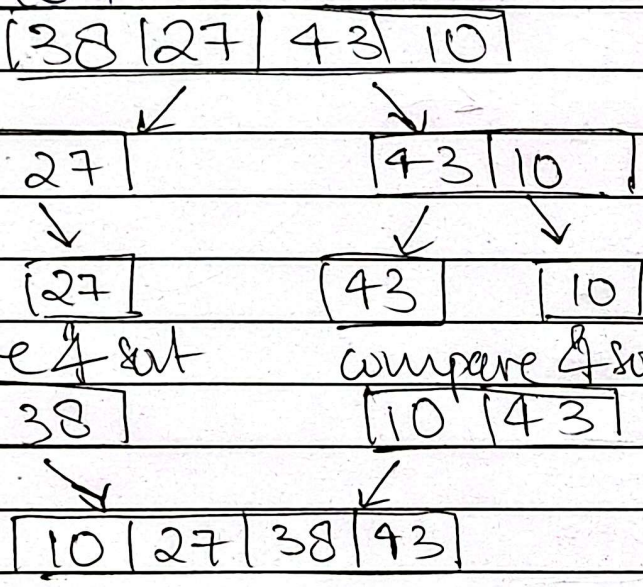
Date _____

RECURRENCES:

① MERGE SORT

To sort an array of size n, sort the left half, sort the right half. and finally merge the two results.

EXAMPLE:



We can do merge in linear time. So if T(n) denotes running time on input of size n.

recurrence : $T(n) = 2T\left(\frac{n}{2}\right) + cn.$

→ O(n lg n)

best avg worst case for merge sort.

Date _____

SELECTION SORT

We run through the array to find the smallest element. We put this in the leftmost position, and then recursively sort the remainder of the array.

Recurrence: $T(n) = T(n-1) + cn$.

- $T(n-1)$: recursively sort remaining $n-1$ elements.
- cn: linear scan to find smallest element.

Upper bound Analysis

Expand the recurrence

$T(n) = cn + c(n-1) + c(n-2) + \cdots + c(n-n)$

each term at most would be cn. we have n terms. so $n \times cn = cn^2$

$T(n) \leq cn^2$.

Alternatively, we ~~are~~ can take c common.

$T(n) = c(n + n-1 + n-2 + \cdots + 1)$

so sum of first n integers $= \frac{n(n+1)}{2}$

$$T(n) \leq c\left(\frac{n(n+1)}{2}\right) = \frac{cn^2 + cn}{2}$$

$T(n) = O(n^2)$.

Date _____

Lower Bound Analysis:

We are only considering half the cells. It is a common lower bound technique. Even if we don't calculate the work for the whole recursion, if we can show that a big chunk of recursion already takes $\Omega(n^2)$, then the whole algorithm must take at least $\Omega(n^2)$ as well.

To prove the lower bound, we are looking at just a part of the total work - specifically, the work done in the last half of the recursive calls. The list size is b/w $n/2$ and n. (Still quite big).

In this, the list size is atleast $n/2$ always. Therefore, the time to find the smallest element in each of these calls is atleast $c \cdot \frac{n}{2}$.

And there are $n/2$ such calls where the size is $n/2$ or more.. so the total work is

$$\left(\frac{n}{2}\right)\left(\frac{cn}{2}\right) \Rightarrow \frac{cn^2}{4}$$

since the algo does atleast this much work, we have

$$T(n) = \Omega(n^2)$$ tht

And $T(n) \in \Theta(n^2)$ → Upper $O(n^2)$, Lower $\Omega(n^2)$.

Date _____

SOLVING RECURRENCES:

① solving by guess and inductive proof:

Make a guess and then prove the guess to be correct inductively. If our guess was incorrect, we will get clues for a better guess.

eg $T(n) = 7T\left(\frac{n}{7}\right) + n \quad ; \quad T(1) = 0$

We will start by trying a solution for $T(n) \leq cn$. for some $c > 0$. We would then assume it was true inductively for $n' < n$ and plug into our recurrence using $n' = \frac{n}{7}$ to get:

$$T(n) \leq 7\left(\frac{cn}{7}\right) + n$$
$$= cn + n$$
$$= (c+1)n.$$

Unfortunately, this isn't what we wanted as our c went up by 1 when our n went up by a factor of 7. Our multiplier is acting like $\log_7 n$.

Our new guess is that our multiplier is of the form $\log_7 n$. So our new guess is $T(n) \leq n \log_7 n$. If we assume this holds true inductively for $n' > n$, we get:

Date _____

$$T(n) \leq 7\left(\frac{n}{7}\log_7\frac{n}{7}\right) + n$$

$$= n\log_7\frac{n}{7} + n$$

$$= n\left(\log_7 n - \log_7 7\right) + n$$

$$= n\log_7 n - n + n$$

$$= n\log_7 n.$$

Therefore we have verified our guess.

BEWARE:

our initial guess cn was incorrect. Don't assume it to be correct thinking "~~let's~~ we assumed $T(n/7)$ was $\theta(n/7)$ and got $T(n) = \theta(n)$" because constants changed ($c \to c+1$) so they weren't really constants at all.

EXPLANATION:

Date _____

Acha so, solving recurrences essentially just means figuring out time complexities.
Guess and Prove, also refered to as the Substitution method. Very common for divide and conquer algorithms.

$$T(n) = 7T\left(\frac{n}{7}\right) + n.$$

was a typical recurrence relation where a problem of size n was broken into 7 subproblems of size n/7. with n additional work done at current level.

We start Guess and prof by guessing a form of the solution.
eg we started by assuming a linear upper bound.

$$T(n) \leq cn.$$

We then plug in (or substitute) this T(n) into the recurrence relation and assume the form will still hold:

$$T\left(\frac{n}{7}\right) \leq c\frac{n}{7}$$

Substitution looks like.

$$T(n) \leq 7\left(\frac{cn}{7}\right) + n = cn + n = (c+1)n.$$

Now this was a problem, which means our

Date _____

initial guess was wrong. The problem is the constant increased; The guess of $T(n) \leq cn$ doesn't hold.

Now instead of assuming c to be a constant factor let's assume it acts like $\log_7 n$. It actually works.

(2) Solving by unrolling.

Many times, the easiest way to solve is to unroll a recursion to get a summation.
We used this for selection sort. Another Example is.

$T(n) = n^5 + T(n-1)$

it unrolls to

$T(n) = n^5 + (n-1)^5 + (n-2)^5 + \cdots + 1^5$.

There are n terms each of which is at most n^5 so the sum is at most $n \times n^5 = n^6$. $O(n^6)$.
There are top $n/2$ terms where each are atleast $(n/2)^5$ which sums to $\frac{n}{2} \times \left(\frac{n}{2}\right)^5 = \left(\frac{n}{2}\right)^6$. $\Omega\left(\frac{n^6}{64}\right)$.

Therefore $\Theta(n^6)$.

A convenient way to look at summations of this form is to see them as approximations to an integral. eg $f(x) = x^5$ $\int_0^n x^5$ Sum is atleast the integral of $f(x) = x^5$ evaluated from 0 to N.

Date _____

at most evaluated from 1 to n+1.

This would give a range.

$$\int_0^n x^5 \qquad \int_1^{n+1} x^5$$

$$\left(\frac{x^6}{6}\right)_0^n \qquad \left(\frac{x^6}{6}\right)_1^{n+1}$$

$$\frac{n^6}{6} - \frac{0^6}{6} \qquad \frac{(n+1)^6}{6} - \frac{1}{6}$$

$$\frac{n^6}{6}$$

(At least)

$$\left[\frac{1}{6}n^6, \frac{1}{6}(n+1)^6\right].$$

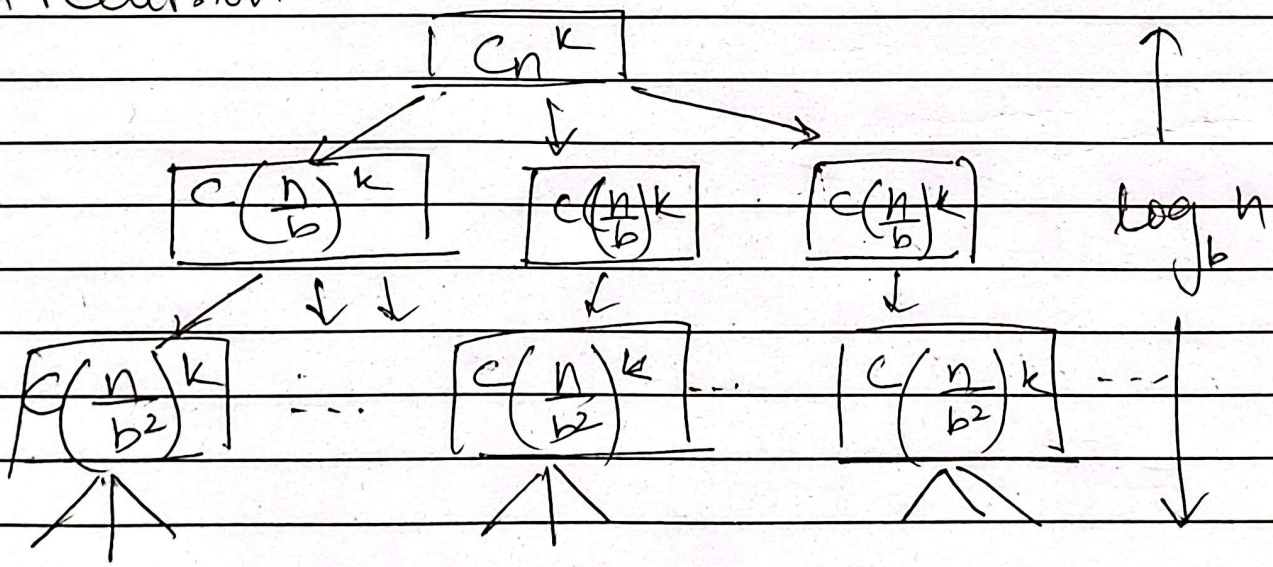
Date _____

(3) Recursion Trees, Stacking Bricks, Master formula:

$$T(n) = aT\left(\frac{n}{b}\right) + cn^k \qquad T(1) = c.$$

- a, b, c, k are all positive constants.
- The algorithm
  - ↳ does cn^k work up front.
  - ↳ divides the problem into a pieces of size $\frac{n}{b}$. Solving each one recursively.

Recursion tree represents this process. Each node contains work done up front and one child for each recursive call. Leaves are base cases of recursion.



To compute results of recurrence, add up all values in the tree. Add them up level by level.

Date _____

cn^k + Root

$a \times c\left(\frac{n}{b}\right)^k$ + 1st level

$a^2 \times c\left(\frac{n}{b^2}\right)^k$ + 2nd level

⋮ ⋮

depth of tree (not including root) is $\log_b n$.

Summation:

$$cn^k\left[1 + \frac{a}{b^k} + \left(\frac{a}{b^k}\right)^2 + \left(\frac{a}{b^k}\right)^3 + \cdots + \left(\frac{a}{b^k}\right)^{\log_b n}\right]$$

let $r = \frac{a}{b^k}$, the summation can be written as

$$cn^k\left[1 + r^1 + r^2 + r^3 + \cdots + r^{\log_b n}\right].$$

we get 3 cases.

Date

Case 1: $r < 1$. sum is a convergent series.

$= \frac{1}{(1-r)}$

upper bound $\frac{cn^k}{1-r}$; lower bound cn^k (by just the first term).

since both r & c are constants, $\Theta(n^k)$.

Case 2: $r = 1$. all terms in seq equal to 1.

$cn^k(\log_b n + 1) \in \Theta(n^k \log n)$.

Case 3: $r > 1$. last term of summation dominates. Visible by pulling it out.

$$Cn^k r^{\log_b n}\left[\left(\frac{1}{r}\right)^{\log_b n} + \cdots + \frac{1}{r} + 1\right].$$

since $\frac{1}{r} < 1$, the summation is at most $\frac{1}{1-\frac{1}{r}}$.

which is a constant. Therefore:

$$T(n) \in \Theta\left(n^k \left(\frac{a}{b^k}\right)^{\log_b n}\right)$$

$b^{k \log_b n} = n^k$, so simplifies to $T(n) \in \Theta(a^{\log_b n})$

$a^{\log_b n} = b^{(\log_b a)(\log_b n)} = n^{\log_b a}$ to get

$T(n) \in \Theta(n^{\log_b a})$.

Date _____

Master theorem

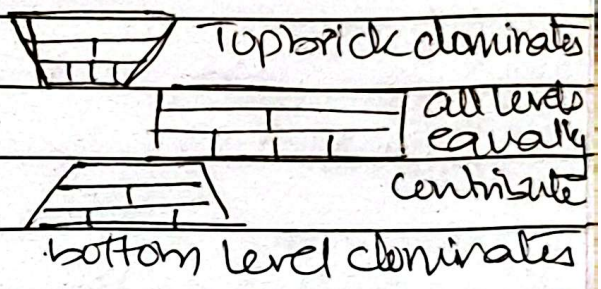
$$T(n) = aT\left(\frac{n}{b}\right) + cn^k$$

$$T(1) = c$$

$T(n) \in \Theta(n^k)$ if $a < b^k$

$T(n) \in \Theta(n^k \log n)$ if $a = b^k$.

$T(n) \in \Theta(n^{\log_b a})$ if $a > b^k$.



Think of each node in recursion tree as brick of height 1 and width equal to value inside it. Goal is to compute area of stack.

Theorem:

For constants $c, a_1, a_2 \cdots a_k$ s.t $\sum a_i < 1$

$$T(n) \leq T(a_1 n) + T(a_2 n) + \cdots + T(a_k n) + cn$$

Then $T(n) \in O(n)$

If $\sum_{a_i < 1} a_i = 1$ then $T(n) \in \Theta(n \lg n)$.

Date _____

RANDOMIZED (PROBABILISTIC) ALGORITHMS

I is some input.
T(I) is running time of our algorithm.
$T(n) = \max\{T(I)\}$ inputs I of size n.
↳ WORST CASE

In Avg case running time, we're concerned with performance on typical inputs I.

If we have worst case bound for some problem A, we can consider solving problem B by converting instances of B to instances of A.

Algorithms that have large gap b/w avg and worst case performances, we can improve worst case performance by actually adding randomization to the algorithm.

Date _____

QUICKSORT:

Given array of some length n,

1. Pick an element p of the array as pivot.
2. Split the array into sub arrays LESS, EQUAL and GREATER by comparing each element to the pivot.
3. recursively sort LESS and GREATER.

① BASIC QUICKSORT:

→ same as above. Choose the leftmost element in array as the pivot (always).

→ worst case: if the array is already sorted then in step 2 all elements except p will go into the Greater bucket.

→ since GREATER array is in sorted order, this process continues recursively resulting in $\Omega(n^2)$

→ for n element array, step 1 executed at most n times, step 2 takes at most n steps. Therefore $O(n^2)$.

→ Avg case is $O(n \log n)$.

② RANDOMIZED QUICKSORT

②: same as above. each time picking a random element in array as pivot.

→ $E(T(I))$ is $O(n \log n)$. Worst case Expected time bound.

(This is better than avg case bound because we aren't assuming any special properties of the input)

Making the algorithm probabilistic gives us more control over the running time.

Date _____

BASICS OF PROBABILISTIC ANALYSIS:

Experiment → rolling 2 dices.
Possible outcomes → 36 (elementary elements)
If for dice, prob for each outcome → 1/36.
Sample space S, probability measure P.
In discrete probability distribution
$P(e) \geq 0$ for all $e \in S$. and $\sum_{e \in S} P(e) = 1$
prob always greater than or equal to 0. sum of prob of all elements is 1.
↳ Basic prob rules ↙ (from o'level maths).
Pr(e) same thing as P(e) (don't get confused).

event : subset of S. eg sum of 2 dice = 7.

Random Variable : function mapping elementary elements to integers or reals.
eg X_1 ü result of first die
X_2 ü result of second die.
$X = X_1 + X_2$ represents sum of the two.
prob $(X = 7)$?
} Just a more formal way to write stuff.

Using random variables we care about expectations:

Useful when you have a full list of all possible outcomes and their probabilities (and you can manually write em out).

Date

For a discrete random variable X over sample space S, the expected value of X is:

$$E[X] = \sum_{e \in S} Pr(e) X(e)$$

→ multiply each possible value of X by its probability.
→ sum up these products.

This expectation is basically average value you'd get of X over S if you weigh each element according to its probability.

Eg: expected value when you roll a dice.

$$E[X] = \frac{1}{6}\times 1 + \frac{1}{6}\times 2 + \frac{1}{6}\times 3 + \frac{1}{6}\times 4 + \frac{1}{6}\times 5 + \frac{1}{6}\times 6$$

$$= \frac{1+2+3+4+5+6}{6} \Rightarrow \frac{21}{6} \Rightarrow 3.5$$

You can group together according to different values of random variable and get

$$E[X] = \sum_{a} Pr(X=a)\, a$$

when you're give prob mass func (PMF)

If you partition the probability space into disjoint events, we can get expectation of random variable X as:

$$E[X] = \sum_{i} \sum_{e \in A_i} Pr(e) X(e)$$

$$\hookrightarrow \sum_{i} Pr(A_i)\, E[X|A_i]$$ → expected value of X given A_i.

$$\hookrightarrow \frac{1}{Pr(A_i)} \sum_{e \in A_i} Pr(e) X(e).$$

Date _____

LINEARITY OF EXPECTATION (THEOREM)

for any two random variables X and Y

$$E[X+Y] = E[X] + E[Y]$$

(analyzing a complicated random variable by writing it as a sum of simple random variables and then separately analysing these separate random variables).

PROOF:

$$E[X+Y] = \sum_{e \in S} Pr(e)(X(e)+Y(e))$$

$$= \sum_{e \in S} Pr(e)X(e) + \sum_{e \in S} Pr(e)Y(e)$$

$$= E[X] + E[Y].$$

EG 1: CARD SHUFFLING

Unwrap a fresh deck of cards and shuffle until cards are completely random. How many cards expected to be in same position as start?

Expected value of a random variable X denoting the number of cards ended in same position as they started.

X can be written as sum of random variables X_i, one for each card

$$X_i = \begin{cases} 1 & \text{if ith card ends in position } i \\ 0 & \text{otherwise.} \end{cases}$$

Date _____

$Pr(X_i = 1) = \frac{1}{n}$ // n is the no. of cards.

$Pr(X_i = 1)$ is also $E[X_i]$

↳ $E[X] = E\left(\sum_i X_i\right) = \sum_i E(X_i)$

$E(X_i) = Pr(X_i = 1) = \frac{1}{n}$

↳ $E[X] = \sum_i \frac{1}{n} = 1.$

↳ using linearity of expectation:

$E[X] = E[X_1 + X_2 + \cdots + X_n]$

$= E[X_1] + E[X_2] + \cdots + E[X_n]$

$= 1$

no matter how large a deck we consider, expected no. of cards ending in same position as start is 1.

Date

Random variable is numerical outcome of a random experiment. Assigns a real number to each possible outcome of an experiment.

① $E[X] = \sum_{e \in S} Pr(e) X(e)$ // Basic Expectation formula

Eg. Roll a dice : $\frac{1}{6} \times 1 + \frac{1}{6} \times 2 + \frac{1}{6} \times 3 + \frac{1}{6} \times 4 + \frac{1}{6} \times 5 + \frac{1}{6} \times 6$

② $E[X] = \sum_{a} Pr(X = a) a$ // Grouped Expectation formula

Eg: when PMF being used : eg ~~roll 2 dice~~ flip 2 coins, what's the expected no. of heads you can get.

X = Number of heads.

2 coins HH TT HT TH

2 0 1 1

$Pr(X=0) = \frac{1}{4}$; $Pr(X=1) = \frac{2}{4} = \frac{1}{2}$; $Pr(X=2) = \frac{1}{4}$

$E(X) = \frac{1}{4} \times 0 + \frac{1}{2} \times 1 + \frac{1}{4} \times 2$

(easier than listing all individual outcomes).

If were using ① then.

$E[X] = \frac{1}{4} \times 2 + \frac{1}{4} \times 0 + \frac{1}{4} \times 1 + \frac{1}{4} \times 1$ } is eg mein chal gaya lekin large S mein issue hoga.

HH TT HT TH

SHAHEER

Date _____

(3) $E[X] = \sum_i Pr(A_i) E[X|A_i]$ // using conditional expectation.

→ when breaking prob into cases is easier.
→ used in algo analysis
→ when conditional dependencies exist.

eg biased coin toss.
0.7 prob of coin being fair.
0.3 prob of coin always landing on heads.

X no. of heads in single toss.

CASE 1: Fair coin → A_1
$Pr(A_1) = 0.7$
$E[X|A_1] = 0.5$ // 50/50 H & T chance

CASE 2: Biased coin → A_2
$Pr(A_2) = 0.3$
$E[X|A_2] = 1$ // always head.

Apply Formula.
$E[X] = Pr(A_1) E[X|A_1] + Pr(A_2) E[X|A_2]$
$= (0.7 \times 0.5) + (0.3 \times 1)$
$= 0.35 + 0.3$
$= 0.65$

INVERSIONS IN A RANDOM PERMUTATION

Date

pair of elements that appear out of order relative to their natural order.

eg P = [3, 1, 4, 2]

3 1 → (1)
3 4 nope.
3 2 → (2)
1 4 nope.
1 2 nope.
4 2 → (3)

Total 3 inversions.

eg [1, 2, 3, 4] no inversion
[4, 3, 2, 1] max inversions.

formal Def:

Given a permutation P = [$P_1, P_2, \dots, P_n$] of n distinct elements, an inversion is a pair (i, j) such that $i < j$ and $P_i > P_j$
(elements at index i is greater than element at index j, but appear earlier in the seq).

To find Exact No. of inversions:

~~Brute force Approach O(n^2)~~

Merge sort Modification. O(n lg n).

↳ leverage the merge step.

if $P_i > P_j$ and $i < j$ then all remaining elements in left half will also be greater than P_j, contributing to more inversions.

SYI SHAHEER

Date _____

count them in merge step leading to overall $O(n \lg n)$ complexity.

To find Expected no. of inversions:

Linearity of Expectation

Indicator random variable for each pair (i, j) where $i < j$ to check if an inversion occurs.

Let $I_{ij} = \begin{cases} 1 & \text{if } P_i > P_j \text{ i.e an inversion occurs.} \\ 0 & \text{otherwise.} \end{cases}$

Since each pair (i, j) is equally likely to be in either order in a random permutation, the prob that $P_i > P_j$ is

$$P(P_i > P_j) = \frac{1}{2}$$

Total no. of inversions in permutation

$$X = \sum_{i<j} I_{ij}$$

Using linearity of Expectation

$$E[X] = \sum_{i<j} E[I_{ij}]$$

Since $E[I_{ij}] = Pr(P_i > P_j) = \frac{1}{2}$ we get

Date _____

$$E[X] = \sum_{i<j} \frac{1}{2}$$

computing Summation.

the no. of pairs in n-element permutation is

$$\binom{n}{2} \Rightarrow {}^nC_2 \Rightarrow \frac{n(n-1)}{2}$$

So. $$E[X] = \frac{1}{2} \times \frac{n(n-1)}{2}$$

$$= \frac{n(n-1)}{4} \quad \text{Ans.}$$