El costo amortizado no da garantías sobre el tiempo de ninguna operación individual.

R: V, ya que el análisis amortizado sirve para mostrar que el costo promedio de una operación es pequeño, aunque pueden haber operaciones individuales costosas.

Se puede ordenar en tiempo o(nlogn) en promedio mediante comparaciones binarias.

R: Quick sort? -> YES (cuando se elige el pivote al azar, es decir, caso promedio)

También Mergesort es nlogn.

Un algoritmo probabilístico aleatorizado puede fallar algunas veces si y otras no sobre una misma entrada.

R: V, porque una misma entrada no te asegura que el algoritmo hará los mismos
pasos cada vez, debido a su naturaleza probabilistica/aleatorizada.

Una cola de prioridad en memoria secundaria puede conseguir tiempo amortizado O(1/B) por operación, siendo B el tamaño del bloque.

R: Si es por comparaciones es óptimo y toma 1/B por entrar al buffer. (?????)

En el resumen de lucas del c1 sale que el costo en realidad es O(1/B logm n)

Si existe un esquema de aproximación completamente polinomial para un problema, entonces éste no es NP-completo.

R: F, cuando hacemos algoritmos aproximados estamos haciendo a-aproximaciones sobre un problema que en varios casos son NP-completo.

El hashing perfecto ofrece garantías de tiempo esperado, no de peor caso.

R: Ofrece garantías para el peor caso. El que ofrece garantías para el tiempo esperado de ejecución es el hashing universal.

P1) Preguntas conceptuales:

1- ¿Qué significan y cómo se relacionan la complejidad de un problema, una cota inferior a un problema, una cota superior a un problema, la complejidad del mejor caso de un algoritmo y la complejidad de peor caso de un algoritmo?

R: La complejidad de un problema es el costo del peor caso del mejor algoritmo que resuelve dicho problema. Por lo tanto, cada nuevo algoritmo que se encuentra para resolver ese problema establece una nueva cota superior de complejidad. La cota inferior es el costo mínimo que cualquier algoritmo que resuelva determinado problema debe pagar.

2- Describa el modelo de costo usado frecuentemente (y en el curso) para evaluar algoritmos en memoria secundaria. ¿Qué ventajas y desventajas tiene?

R: El modelo que se utiliza es calcular el costo de I/O de cada algoritmo, despreciando el costo de mover el cabezal y las operaciones en CPU.

Ventajas: una ventaja es que esto considera que el costo de los algoritmos en memoria secundaria es solo el costo de operaciones a disco que en realidad representa las operaciones mas costosas (ordenes de magnitud mas lento que las operaciones que se hacen en RAM) y te olvidas de analizarlas.

Desventajas: Puede darse un cálculo erróneo en casos en que un algoritmos necesite mucho mas mas movimiento de cabezal que otro, su costo será el mismo. También puede pasar que haya operaciones en memoria interna que sean costosas y que no las consideremos.

3- ¿En qué difieren los conceptos de tiempo promedio, tiempo esperado y tiempo amortizado?

R: El costo esperado promedia sobre las distintas ejecuciones posibles del algoritmo frente al mismo input y considera el peor input posible.

El costo promedio se refiere a los distintos inputs posibles, para cada uno de los cuales un algoritmo determinístico tiene un costo fijo.

Costo amortizado de una operación al hacer un análisis sobre una secuencia de operaciones.

4- ¿En qué difieren los conceptos de algoritmo probabilístico, aleatorizado y aproximado?

R: El algoritmo probabilístico se equivoca con cierta probabilidad de error, la que puede ser arbitrariamente pequeña con un incremento moderado en el costo del algoritmo.

Si siempre hacen lo mismos son determinísticos, mientras que si hay una componente de azar en su ejecución son aleatorizados. Pudiendo tener un costo distinto cada vez que se enfrente al mismo input.

Por otro lado los algoritmos aleatorizados son aquellos algoritmos ya existentes a los que se les introduce una componente de aleatorización, típicamente para mejorar su complejidad de tiempo esperado (por ejemplo quicksort). No realizan las mismas operaciones frente a un mismo input y podrían equivocarse.

Algoritmos aproximado: se tiene un algoritmo muy complejo por lo que se crea un algoritmo que en tiempo polinomial que resuelve el problema aproximándose al óptimo.

5- ¿Cómo se relaciona la dificultad de los problemas de decisión con los problemas asociados de optimización?

R: para todo problema de decisión hay un problema de optimización que lo resuelve

6- ¿Cómo se refinan los problemas NP-completos en términos de aproximabilidad? ¿Cómo puede mostrarse que ciertos problemas no son aproximables?

R: Se crea un algoritmo que resuelva algo parecido al problema que es NP. Se dice que el nuevo algoritmo es una aproximación del problema original. Para demostrar que un problema no es aproximable hay que demostrar que no existe ningún ro(n) tal que |ALG|/|OPT| < ro(n) (o > en el caso de maximización, creo que debería ser |OPT|/|ALG| < ro(n) para maximización). (ESTA BIEEEEN?)
Por contradicción se puede decir que si hay un algoritmo aproximado para un problema NP-completo, pero se debe mostrar que si lo hay, existe otro problema NP que puede ser resuelto con nustra aproximacion -> no existe a-aproximacion para el primer problema.
P2) Se desea una estructura de datos para almacenar un conjunto de strings en memoria secundaria. Estos strings pueden ser bastante largos, incluso más que el espacio de una página de disco.
P3) Tiene usted k$ y quiere gastarse la mayor cantidad posible en una feria a lo largo de una calle de un solo sentido (es decir, usted solo puede hacer una pasada por ella), de modo que cuando compra algo ya no puede devolverlo (además hay un solo ejemplar de cada producto). Los productos tienen distintos precios entre 1 y k. Pero por ejemplo si se gasta 1$ en el primer producto, y todos los demás cuestan k$, no podrá comprar nada más.
R: La mejor opción que tiene el problema es comprar todo lo que se pueda hasta que se acabe el dinero, ya que si se intentara esperar un resultado óptimo se corre el riesgo de no encontrar nada de no existir dicho input. Para este algoritmo, el peor caso sería aquel en que el primer objeto vale 1$ y todo el resto k$, por lo que solo se habría gastado 1$. Esto es k-competitivo ya que el óptimo logra gastar k, lo que hace que esta solución sea k-veces peor que la óptima.
2- Considere ahora una variante en la que usted puede devolver uno o más productos ya comprados y recuperar el dinero (pero no puede comprar algo por lo que ya pasó). Muestre un algoritmo 2-competitivo para este caso.
R: El algoritmo compra todo mientras no se llene, y de encontrarse con algo que lo llene tiene dos opciones:
-Si vale menos o igual a k/2$ no lo compra y termina, ya que de hacer que se llene, significa que hasta el momento se ha gastado al menos k/2$.
-Si vale más de k/2$ entonces vende todo lo que tiene y compra el objeto, haciendo que gaste más de k/2$.
En ambos casos de gasta más de k/2$, por lo que el algoritmo es 2-competitivo, ya que solo es dos veces peor que el óptimo (que logra gastarlo todo).

P4) Dado un grafo no dirigido G = (V, E), el problema de MAX-CUT es el de dividir V = S ∪ S', con S' = V − S, de manera de maximizar la cantidad de aristas que cruzan de S a S'.
1- Considere el algoritmo que toma cada vértice y tira una moneda para decidir si lo pone en S o S'. Pruebe que, en el caso esperado, se obtiene una 2-aproximación.
R:
E(suma de los pesos de las aristas que pertenecen al maxcut)
= Suma(E(pesos de las aristas que pertenecen al maxcut))
= Suma(peso de la arista * P(la arista pertenece al maxcut))
=Suma(peso de la arista) * ½ //1/2 es la P de que elija esa arista (por la moneda)
=|E| / 2 //porque consideramos que todas las aristas podrían estar en el max-cut
P1) Indique, en los siguientes casos, la única respuesta correcta a la pregunta "¿de qué está hablando esto?". Los desaciertos no restan puntos.
1. Obtener la mediana determinísticamente requiere al menos 2,95n comparaciones.
a) Algoritmos aproximados
b) Algoritmos probabilísticos
c) Cotas inferiores (esto?)
d) Algoritmos paralelos
e) Cotas superiores
2. Se puede obtener la mediana con 1,5n + o(n) comparaciones esperadas.
a) Cotas inferiores
b) Algoritmos en memoria secundaria
c) Algoritmos aproximados
d) Algoritmos aleatorizados (este)
e) Análisis amortizado
3. Un índice que encuentra los k elementos más cercanos a una consulta se construye eligiendo K objetos del conjunto al azar y almacenando, para cada uno de ellos...
a) Algoritmos paralelos
b) Algoritmos aleatorizados
c) Algoritmos probabilísticos(aquí me parece que este por el tema del muestreo)
d) Análisis amortizado
e) Algoritmos aproximados
4. El algoritmo para construir el árbol de sufijos de T [1, n] procede de izquierda a derecha en T . Para cada nuevo carácter T [i], extiende todos los nodos internos terminados en T [i − 1]. En total el costo es O(n) porque...
a) Algoritmos aleatorizados
b) Cotas inferiores

c) An´alisis amortizado(creo q este)
d) Algoritmos en memoria secundaria
e) Algoritmos aproximados
5. Cuando se tiene el arreglo de n elementos, se busca cada uno de ellos en el B-tree, con lo cual el costo total es O(n logB n).
a) Análisis amortizado (este?)
b) Algoritmos aproximados
c) Algoritmos en memoria secundaria(creo q mejor es este, B es el tamaño de bloque de memoria)
d) Algoritmos en strings
e) Algoritmos paralelos
6. Con este método se logra eficiencia O(1/ log n), la cual se puede mejorar reduciendo p.
a) Cotas inferiores
b) Algoritmos aproximados
c) Algoritmos paralelos (este)
d) Algoritmos aleatorizados
e) Algoritmos en strings
P3) Se tiene un árbol escrito en un arreglo A[1..n], donde cada elemento A[i] = j indica que el padre de i es j. La raíz r indica A[r] = 0.
1- Diseñe un algoritmo PRAM para calcular la profundidad de cada nodo (distancia a la raíz) en un arreglo P[1..n], en tiempo O(logn). Analice T(n), W(n), speedup, eficiencia, y número óptimo de procesadores. ¿Qué tipo de modelo usó? (EREW, CREW, ...).
R:(IDEA) Hacer una relación con prefix paralell, cada nodo depende de la altura de su árbol (asi como en prefix paralell cada elemento depende de la suma acumulada hasta el i-1). Con esto el tiempo que demora es O(logn).
T(n) = nlogn (porque h se calcula en log n y hay q hacerlo para cada nodo).
T(n,n) = log n
W(n) = p * logn -> n * logn
S(n,n) = nlogn/logn = n
E(n,n) = nlogn/(n*logn) = 1
El modelo es CREW (varios leen del padre, pero solo un procesador escribe en su posición).
2- Lo mismo, ahora para calcular la altura de cada nodo (distancia a la hoja más profunda que desciende de él). Considere un modelo CRCW donde, si hay varias escrituras concurrentes, la celda se queda con el m´aximo valor escrito.
R: Cada procesador está apuntando a un valor del arreglo A. A la vez se tiene un arreglo vacío B.
En la primera iteración todos los procesadores van a la posición A[i] = j, y le suman 1 al valor B[j]. Si se encuentran en la raíz, entonces los procesadores dejan de trabajar (iterar).
Se puede ver que en cada iteración se van realizando menos operaciones, por lo que el algoritmo es logn.
T(n) = nlogn
T(n,n) = logn
S(n,n) = n
E(n,n) = n/n = 1
W(n,n) = nlogn
1. ¿Qué significan y cómo se relacionan la complejidad de un problema, una cota inferior a un problema, una cota superior a un problema, la complejidad de mejor caso de un algoritmo y la complejidad de peor caso de un algoritmo?
R: Complejidad de un problema: tiempo estimado de ejecución, escrito en función del tamaño de la entrada.
Cota inferior : mínimo que se demorará un problema. (mejor caso)
Cota superior: lo que se demora en el peor caso.
Complejidad del mejor caso : se define por la cota inferior y tiene que ver con el tiempo estimado del mejor caso.
2. Describa el modelo de costo usado frecuentemente (y en el curso) para evaluar algoritmos en memoria secundaria. ¿Qué ventajas y desventajas tiene?
R: el modelo de costos solo cuenta el numero de lecturas y escrituras a disco, una ventaja es que esto considera que el costo de los algoritmos en memoria secundaria es solo el costo de operaciones a disco que en realidad representa las operaciones mas costosas (ordenes de magnitud mas lento que las operaciones que se hacen en RAM) y te olvidas de analizarlas. Como desventajas puede ser lo que dice el Diego que puede que hayan algoritmos en memoria interna que sean costosos y el modelo no las considera como parte del costo. Otra desventaja que tiene es que el modelo no considera otros factores como la buena localidad de memoria de los algorimos que hace que el proceso sea más rápido debido al caché
2) Considere un árbol de altura h perfectamente balanceado, donde cada nodo interno tiene 3 hijos, por lo que hay n = 3^h hojas. Cada hoja contiene un valor booleano, 0 ó 1. El valor booleano de cada nodo interno se calcula como el mayoritario entre sus 3 hijos.
1. Muestre que cualquier algoritmo determinístico necesita en el peor caso examinar las n = 3^h hojas para encontrar el valor de la raíz.
R: Para poder saber el valor de un padre se necesita conocer el valor de al menos dos de sus hijos. Al querer revisar estos hijos, el adversario podría siempre entregar un input en que los dos primeros hijos sean distintos, obligando a revisar el tercero para saber cual es el valor mayoritario entre los tres hijos.
Por lo tanto sería necesario revisar los tres hijos de cada nodo, lo cual para una altura h equivaldría a hacer revisar n hojas.

2. Considere un algoritmo aleatorizado que elige dos hijos al azar, los evalúa recursivamente, y si dan el mismo valor evita calcular el tercero, de otro modo calcula el tercero para responder. Analice el costo esperado de este algoritmo y muestre que es o(n).
R: No importa el input, en un nodo siempre se cumple que al menos dos de sus hijos deben evaluar al mismo valor, por lo que el algoritmo tiene probabilidad ⅓ de revisar dos nodos y probabilidad ⅔ de revisar tres. Con esto, se esperan revisar <= ⅓*2 + ⅔*3 = 8/3 hijos.
1. ¿Qué significa eficiencia en paralelismo y por qué no puede ser mayor que 1?
R: La eficiencia hace referencia a la tasa de uso de los procesadores y se define como T(n,1) / p * T(n,p).
Por contradicción digamos que existe un algoritmo A que produce: T(n,1) / p * T(n,p) > 1 -> T(n,1) > p * T(n,p).
Supongamos que existe un algoritmo A' que lo único que hace es ejecutar A pero secuencialmente con un solo procesador. El tiempo que toma es a lo más:
TA'(n,1) <= p * TA(n,p).
Pero esto nos lleva a que TA'(n,1) < T(n,1), lo que es una contradicción porque A' tendría que ser mejor que el óptimo.
2) Se tiene un arreglo B[1...n] de bits. Dé algoritmos EREW para calcular (analice en términos de T(n), W(n), T(n,p), speedup y eficiencia:
1. Para todo i, R[i] como la cantidad de 1s en B[1...i]
R: Esto es un prefix sum.
2. Para todo i, S[i] como la posición del i-ésimo 1 en B. Note que esto debe calcularse sólo hasta el valor S[m], con m = R[n].
R: Tener un arreglo con las posiciones, un procesador por cada elemento de B. El procesador mira en indice anterior y mira si hay un 1 o 0 en su elemento. Si hay un 1, escribe indice + 1 en S, si no solo le suma 1 al indice actual.
T(n) = n
T(n,n) = logn (porque es como prefix sum)
W(n) = n  (hay que recorrer todos los elementos)
S(n,n) = n/logn
E(n,n) = n/n*logn = 1/logn

P3) Un árbol α-balanceado, para 1/2 < α < 1, es un árbol binario de búsqueda donde todo subárbol T = (root,Tl,Tr) cumple |Tl| ≤ α·|T| y |Tr| ≤ α·|T|. Las operaciones para buscar y mantener un´arbol α-balanceado son las mismas que para un árbol binario de búsqueda, excepto que luego de insertar o borrar un nodo, se busca el nodo m´as alto en el camino del punto de inserción/borrado hacia la raíz, que no esté α-balanceado, y se lo reconstruye como árbol perfectamente balanceado (el costo es proporcional al tamaño del subárbol que se reconstruye).
1. Muestre que la búsqueda en un árbol α-balanceado cuesta O(log n), y que lo mismo ocurre con las inserciones y borrados, si no consideramos las reconstrucciones. ¿Qué constante obtiene multiplicando el log n?
R:
P4) Se tienen k arreglos apuntados por los punteros L1...Lk, cada arreglo en un área distinta de memoria. TambiÉn se sabe el largo de cada arreglo, l1 ...lk. Sea n = l1 + ... + lk. Se desea concatenar los k arreglos en orden en un gran arreglo L de largo n. Note que los li pueden ser muy distintos.

Proponga un algoritmo CREW PRAM eficiente para resolver este problema y analícelo (tiempo, trabajo, speedup, eficiencia). Indique cuántos procesadores podría utilizar como mínimo para obtener el mejor tiempo posible de su solución.

R: