$\Theta(\log n!) = \Theta(n \log n)$

$\Theta(\log(\binom{M}{B})) = \Theta(B \log(\frac{M}{B}))$

máx lista → n-1 comp

máx + min → $\frac{3}{2}n - 2$

máx + 2° máx → $n + \lfloor \log n \rfloor - 2$

$E(X) = \sum_{i=1}^{n} i \cdot Pr(x=i)$

Btrees: - árbol balanceado
- grado mínimo $t \geq 2$ (salvo en raíz)
- cada nodo contiene entre $[t-1, 2t-1]$ llaves (salvo raíz)
- llaves ordenan los hijos

Heap: - árbol binario completo salvo última capa (de izq a der)
- se representa como arreglo A[1]... A[n]
  Parent(i) = $\lfloor i/2 \rfloor$
  Left(i) = 2i, Right(i) = 2i+1
- No hay orden entre hermanos

Encontrar máx = $\Theta(1)$

Eliminar máx = $\Theta(\log n)$

Insertar / incrementar llave = $\Theta(\log n)$

Counting-Sort input A[1..n], output B[1..n], aux C[1...k] (dígitos ≠)

```
for i ← 0 to k
    do C[i] ← 0
for j ← 1 to n
    do C[A[j]] ← C[A[j]] + 1      C[i]: # de veces que aparece i en A
for i ← 1 to k
    do C[i] ← C[i] + C[i-1]       C[i] es # de elts ≤ a i
for j ← n to 1
    do B[C[A[j]]] ← A[j]
       C[A[j]] ← C[A[j]] - 1
```

Costo $\Theta(n+k)$ // Es estable, mantiene orden.

RadixSort(A, d)
  for i ← 1..d
    do CountingSort en dígito i

ordenar por dígito menos significativo

$\Theta(d(n+k))$ si $d \ll n \Rightarrow \Theta(n)$

$\binom{n}{k} = \frac{n!}{k!(n-k)!}$, $\sum_{i=0}^{n} r^i = \frac{1-r^{n+1}}{1-r}$, $\sum_{i=0}^{n} 2^i = 2^{n+1}-1$

$\sum_{i=0}^{\infty} x^i = \frac{1}{1-x}$ $\xrightarrow{\frac{d}{dx}}$ $\sum_{i=0}^{\infty} i x^i = \frac{x}{(1-x)^2}$

Variables indicatorias

$X_i = \begin{cases} 1 & \text{si algo} \\ 0 & \text{w} \end{cases}$ $\quad X = \sum_i X_i \Rightarrow E(X) = \sum_i E(X_i) = \sum_i Pr(X_i = 1)$ (lineal)

Desigualdad de Markov

$\forall a > 0$, $Pr(X \geq a) \leq \frac{E(X)}{a}$

Dist. Geométrica

$P(X=x) = (1-p)^{x-1} p \longrightarrow E(X) = \frac{1}{p}$

ALG PARALELOS PRAM parallel random access memory

$T(n,p)$ = # pasos en entrada tamaño n con $p \geq 1$ procesadores

$S(n,p) = \frac{T(n,1)}{T(n,p)}$ cuanto mejor es tener p proc vs 1

$E(n,p) = \frac{S(n,p)}{p}$ cuánta mejora otorga cada proc extra

Lemma Brent si existe alg CREW con $T(n,p)=t$, $w(n)=s$
$\Rightarrow \exists$ alg CREW con $T(n, s/t) = \Theta(t)$

Pointer Jumping

```
while ∃ i tq next[i] ≠ NULL:        ← prefix compute
    for i in parallel
        if next[i] ≠ NULL
            y[next[i]] ← y[i] ⊕ y[next[i]]      magia aquí
            next[i] ← next[next[i]]
```

Teo Maestro: Sea $f: \mathbb{N} \to \mathbb{R}$ creciente
$f(n) \leq a \cdot f(\lceil n/b \rceil) + cn^d$
$a, b, c, d \in \mathbb{Z}$, $c \in \mathbb{R}$, $b > 1$, $d \geq 0$

$$f(n) = \begin{cases} \Theta(n^d) & a < b^d \\ \Theta(n^d \log n) & a = b^d \\ \Theta(n^{\log_b a}) & a > b^d \end{cases}$$

Feria infinita: $\exists k$ total
a) sin devolución: k veces peor → adversario verde 1 1 1 1 1 (k) al comprar
b) con devolución: comprar hasta gastar $\frac{k}{2}$
Luego solo compro producto $p > \$\frac{k}{2}$

Máximo de arreglo: con torneo
$T(n, n/2) = \Theta(\log n)$
$S(n, n/2) = \Theta(n/\log n)$
$E(n, n/2) = \Theta(1/\log n)$
} mejor? Tomar $p = \frac{n}{\log n}$
calcular máx secuencial $\frac{n}{p}$ y luego torneo

Máx en $\Theta(1)$ CRCW. n^2 procesadores, solo pueden escribir "0" concurrentemente → arreglo de n eltos, inicializados 1. Asigno n proc por número, comparar todas las parejas. Solo el máximo no es sobreescrito. si a<b se escribe 0 en a, si no 0 en b.

Multiplicar matrices en paralelo.
ALG $\Theta(n)$ con n^2 procesadores, $C = A \cdot B \Rightarrow C_{ij} = \sum_k a_{ik} b_{kj}$ CREW

```
Prod(A,B,C)
  for procesador i,j
    Cij ← 0
    for k = 1..n
      Cij ← Cij + Aik · Bkj
```

$p = n^2$
$T(n,p) = n$
$T(n,1) = n^3$
$w(n) = n^3$
$E(n) = \frac{n^3}{n^2 \cdot n} = 1$

Coloreo de Grafos: $\Theta(k^n)$ k colores n vértices, verificar es NP-COMPL
ALG aprox Greedy → orden aleatorio de vért a pintar
→ buscar orden óptimo es NP-HARD

ALG aprox coloreo 6-C:
- 2-C? pintar. Costo lineal. Ver que no tenga ciclos de largo impar. Bipar
- Encontrar vértice no pintado de grado a lo más 5 (∃, por prop E,
- Sacarlo y todos sus arcos y pintar recursivamente (hasta 2-C)

Empaquetamiento. NP-COMPLETO. Conjunto $S = \{s_1, \ldots, s_n\}$ tamaños e paquetes de tamaño K. Empaquetar todos los eltos en menor cantidad de paq
ALG 2-Aprox: meter el sgte elemento en paquete donde quepa. si no, alocar nuevo paquete. No pueden haber 2 paquetes usados a menos de la mitad.

- **Costo amortizado tabla duplicada** 1) Análisis agregado: n-ésima inserción pagué n por d inserción + veces que dupliqué
$n + \sum_{i=0}^{\lfloor \log(n-1) \rfloor} 2^i \leq n + 2n = 3n$ 2) Créditos: $c_i = 3$ (1 inserción + 2 reale

¿Cómo bajar probabilidad de error de alg Montecarlo?
Repetir ALG Montecarlo $\log_p(q)$ veces, si p y q son constantes, mant el o

REDUCCIONES tomar un problema NP-COMPLETO y lo arreglo para simular el nuevo.

- Vertex-Cover es NP-COMPLETO (buscar subconjunto de vértices q tocan todos los arcos)
- Independent Set: es NP-COMPLETO (buscar subconjunto de vértices que tienen arcos en común
- Set cover NP-COMPLETO? con vertex cover. $U = E$, crear subconjunto S_v con $v \in V$, S_v contiene arcos adyacentes. Además hay que prob que es NP. ⇒ La verificación de la respuesta debe hacerse en tiemp polinomial
- Dominating set NP-COMPLETO? (conjunto de vértices S talque $\forall v \in V \setminus S$ está conectado al menos con 1 arco a al menos 1 vértice de S)

HASHING $h: U \to \{0, \ldots, n-1\}$, $H = \{h \mid h: U \to \{0, \ldots, m-1\}\}$
H es universal si $\forall k, l \in U$ $k \neq l$ $|\{h \in H \text{ tq } h(k) = h(l)\}| \leq |H|/m$
Hashing universal: escoger función de hash aleatoriamente de forma que sea indep. de las llaves.
- Si la llave k no está en la tabla, el largo esperado de lista $h_{h(k)}$ es $\leq \alpha$
- Si la llave ESTÁ, $\leq 1 + \alpha$

variable indicadora $X_{kl} = I\{h(k) = h(l)\}$, $Pr\{h(k) = h(l)\} \leq \frac{1}{m} \Rightarrow E[X_{kl}] \leq$ ...

Usando Hashing universal y encadenamiento en tabla de tamaño m, toma $\Theta(n)$ procesar secuencia de n op inserción, busq, borrado, con $\Theta(m)$ inserciones

CONTROLES 2014-1

Left(i) = 2i, Right(i) = 2i+1
-No hay orden entre hermanos

contrar máx = Θ(1)
minar máx = Θ(log n)
ertar / incrementar llave = Θ(log n)

unting-Sort input A[1..n], output B[1..n], aux C[1..k] (dígitos ≠)

for i ← 0 to k
  do C[i] ← 0

for j ← 1 to n
  do C[A[j]] ← C[A[j]] + 1 C[i] = # de veces que aparece i en A

for i ← 1 to k
  do C[i] ← C[i] + C[i-1] C[i] es # de elts ≤ a i

for j ← n to 1
  do B[C[A[j]]] ← A[j]
     C[A[j]] ← C[A[j]] - 1

osto Θ(n+k)|| Es estable, mantiene orden.

RadixSort (A, d)
  for i ← 1..d
    do CountingSort en dígito i

ordenar por dígito menos significativo
Θ(d(n+k)) si d << n ⇒ Θ(n)
(base 10)

$\sum_{i=0}^{\infty} x^i = \frac{1}{1-x}$; $\frac{d}{dx} \Rightarrow \sum_{i=0}^{\infty} i x^i = \frac{x}{(1-x)^2}$

$\binom{n}{k} = \frac{n!}{k!(n-k)!}$; $\sum_{i=0}^{n} r^i = \frac{1-r^{n+1}}{1-r}$; $\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$

Variables indicatorias.
$X_i = \begin{cases} 1 & \text{si algo} \\ 0 & \text{si no} \end{cases}$ $X = \sum_i X_i \Rightarrow E(X) = \sum_i E(X_i) = \sum_i P_r(X_i = 1)$ (lineal)

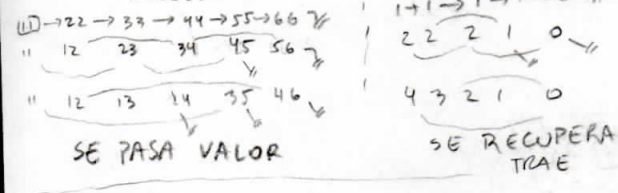
Desigualdad de Markov
$\forall a > 0$, $Pr(X \geq a) \leq \frac{E(X)}{a}$

Dist. Geométrica
$P(X = x) = (1-p)^{x-1} p \longrightarrow E(X) = \frac{1}{p}$

ALG PARALELOS PRAM parallel random access memory
$T(n,p)$ = # pasos en entrada tamaño n con $p \geq 1$ procesadores
$S(n,p) = \frac{T(n,1)}{T(n,p)}$ cuanto mejor es tener p proc vs 1
$E(n,p) = \frac{S(n,p)}{p}$ cuánta mejora otorga cada proc extra

Lemma Brent si existe alg CREW con $T(n,p) = t$, $w(n) = s$
$\Rightarrow \exists$ alg CREW con $T(n, s/t) = \Theta(t)$

Pointer Jumping
while ∃ i tq next[i] ≠ NULL: ← prefix compute
  for i in parallel
    if next[i] ≠ NULL
      y[next[i]] ← y[i] ⊕ y[next[i]] (magia aquí)
      next[i] ← next[next[i]]



SE PASA VALOR
SE RECUPERA VALOR TRAE

PROBLEMAS

Estudiante borracho doblar distancia → girar 180° es 9-competitivo
$2 \cdot (2^{\lceil \log n + 1 \rceil} - 2^i) + n \leq 2 \cdot 2^{\log n + 2} + n = 2 \cdot n \cdot 2^2 + n = 9n$

Subasta: doubling vendedor se queda con todo $
$\frac{ALG}{OPT} = \frac{\sum_{i=0}^{n+1} 2^i}{2^n + 1} = \frac{2^{n+2} - 1}{2^n + 1} \leq \frac{2^{n+2}}{2^n} = 4$
precio ong

Colorear arcos OPT → Δ colores, con Δ = máx grado de nodo
#ALG → 2Δ - 1
cota ↓ ? caso part Δ = 4
cota ↑ ? 2Δ
paso 2Δ - 1
Δ colores Δ-1 colores
color 2Δ Δ+1 ≤ Δ
necesita 7 cols

Coloreo de Grafos: Θ(k^n) k colores n vértices, verificar es NP-COMPL.
ALG aprox Greedy → orden aleatorio de vért a pintar
→ buscar orden óptimo es NP-HARD.

ALG aprox coloreo 6-C:
- 2-C? pintar. Costo lineal. Ver que no tenga ciclos de largo impar. Bipartito
- Encontrar vértice no pintado de grado a lo más 5 (∃, por prop de Euler
- Sacarlo y todos sus arcos y pintar recursivamente (hasta 2-C)

Empaquetamiento. NP-COMPLETO. Conjunto $S = \{s_1, \dots, s_n\}$? tamaños $\in \mathbb{N}$
paquetes de tamaño K. Empaquetar todos los elts en menor cantidad de paquetes
ALG 2-Aprox: meter el sgte elemento en paquete donde quepa. si no, alocar nuevo paquete. No pueden haber 2 paquetes usados a menos de la mitad.

- Costo amortizado tabla duplicada 1) Análisis agregado: n-ésima inserción pagué n por d inserción + veces que dupliqué
$n + \sum_{i=0}^{\lfloor \log(n-1) \rfloor} 2^i \leq n + 2n = 3n$ 2) Créditos: $c_i = 3$ (1 inserción + 2 realoca

¿Cómo bajar probabilidad de error de alg Montecarlo?
Repetir ALG Montecarlo $\log_p(q)$ veces, si p y q son constantes, mantie el orde

REDUCCIONES tomar un problema NP-COMPLETO y lo arreglo para simular el nuevo.
- Vertex-Cover es NP-COMPLETO (buscar subconjunto de vértices que toquen todos los arcos)
- Independent set: es NP-COMPLETO (buscar subconjunto de vértices que no tienen arcos en común
- Set cover NP-COMPLETO? con vertex cover. U = E, crear subconjuntos S con $u \in V$, S contiene arcos adyacentes. Además hay que probar que es NP. ⇒ La verificación de la respuesta debe hacerse en tiempo polinomial
- Dominating set NP-COMPLETO? (conjunto de vértices S talque $\forall v \in V \setminus S$ está conectado al menos con 1 arco a al menos 1 vértice de S)

HASHING $h: U \to \{0, \dots, n-1\}$, $H = \{h \mid h: U \to \{0, \dots, m-1\}\}$
H es universal si $\forall k, l \in U$ $k \neq l$ $|\{h \in H \text{ tq } h(k) = h(l)\}| \leq |H|/m$
Hashing universal: escoger función de hash aleatoriamente de forma que sea indep. de las llaves.
- Si la llave k no está en la tabla, el largo esperado de lista $h_{h(k)}$ es $\leq \alpha$
- Si la llave ESTÁ, $\leq 1 + \alpha$

variable indicadora $X_{kl} = I\{h(k) = h(l)\}$, $Pr\{h(k) = h(l)\} \leq \frac{1}{m} \Rightarrow E[X_{kl}] \leq \frac{1}{m}$

Usando Hashing universal y encadenamiento en tabla de tamaño m, toma Θ(n) procesar secuencia de n op inserción, busq, borrado, con Θ(m) inserciones

CONTROLES 2014-1

C1
1) Verificar grafo no dirigido sin loops es acíclico con preg. ∃? arco en u y v? PDQ
peor caso $\binom{n}{2}$ comp
2) Quicksort con intervalos [I P D] (lo que intersectan P)

3) Buscar x en arreglo A no ordenado eligiendo índices al azar con repeticiones. Número esperado de pasos. R: equiprobable $Pr(x=i) = \frac{1}{n}$
con dist. geométrica, $\Rightarrow E(X) = n$
También $E(X) = \sum_{i=1}^{\infty} i \frac{(n-1/n)^{i-1}}{n} = \sum_{i=1}^{\infty} \frac{i(n-1/n)^i}{n-1}$, y como $\left|\frac{n-1}{n}\right| < 1$, equipa $\sum_{k=1}^{\infty} k x^k = \frac{x}{(1-x)^2}$
y si hay valores repetidos? Ahora la probabilidad de encontrar x
$\frac{k}{n} = E(x) = \frac{k}{n} \sum_{i=1}^{\infty} i(n-1/k)^{i-1} \Rightarrow \frac{n}{k}$

C2

1) Demuestre que es posible construir suffix trie de palabra w (largo n) sobre alf. finito Σ en $\Theta(n^2)$

Suffix tree se construye iterativamente en $n+1$ etapas ($\$$)

número de pasos realizado $\Theta(1+2+\dots+n) = \Theta(n^2)$

b) modificar construcción para que dado w' $|w|=m \le n$ construya conjunto de ocurrencias en $\Theta(m + |Occ(w,w')|)$. La construcción del Suff tree debe ser $\Theta(n^2)$

Sol: Construir suffix tree con substrings en los arcos. Se puede hacer en $\Theta(n^2)$. También sin costo adicional, agregar hojas con índice. Luego buscar en $\Theta(m)$ el nodo que representa a w' en w. Hacer DFS y buscar etiquetas. En $\Theta(|Occ(w,w')|)$ se construye arreglo con etiquetas
↳ es $\Theta(|Occ(w,w')|) = \Theta(k)$. Bajo nodo x hay k hojas, y los nodos son de grado al menos 2 $\Rightarrow$ # nodos en el subárbol es $\sum_{i=0}^{\infty} \lceil \frac{k}{2^i} \rceil = \Theta(k)$.

2) Búsqueda e inserción en conj n eltos. Supongamos $k = \lceil \log(n+1) \rceil$ y rep. binaria de n $a_{k-1} \cdots a_2 a_1 a_0$. Usar k arreglos ordenados $A_0 \cdots A_{k-1}$ $|A_i| = 2^i$. Cada arreglo está lleno si $n_i = 1$.

a) Búsqueda en $\Theta(\log^2 n)$ → búsqueda binaria en todos los arreglos i
costo $\sum_{i=0}^{k-1} \Theta(\log 2^i) = \Theta(k(k-1)/2) = \Theta(\log^2 n)$

b) Insertar? costo peor caso y costo amortizado.
Insertar elto e en A_0. Si está lleno hacer merge y pasar a A_1. si ya existía Insertar elto e en A_0. Si está lleno hacer merge y pasar a A_1. si ya existía A_1, crear A_2 hacer merge del nuevo y viejo A_1 y así. En el peor caso toma tiempo $\sum_{i=0}^{k} \Theta(2^i) = \Theta(2^k) = \Theta(n)$

Costo amortizado: hacer m inserciones, representado binaria $n_{k-1} \cdots n_1 n_0$
el bit n_0 cambia cada vez, el bit n_1 vez por medio, n_2 cada 2 veces, etc
cuando bit i cambia 1 a 0 $\Rightarrow$ merge que tiene costo 2^{i+1}. Luego costo total de m inserciones está acotado $\sum_{i=0}^{k-1} \lfloor m/2^i \rfloor 2^{i+1} \le 2mk$
$= m\,\Theta(\log n)$, luego el costo de ins es $\Theta(\log n)$

3) Stack en mem secundaria.
Mem ppal buffer S con tamaño B (página). Cuando se llena e insertamos, vaciamos S a disco e insertamos el siguiente en S. Al revés si estamos S a disco e insertamos el siguiente en S. Al revés si está vacío. Demostrar costo amortizado $\Omega(1)$ de acc. a disco.
Sol: sea secuencia $2n + (B+1)$ ops: primero insertar $B+1$ eltos y luego n veces sacar 2 y meter 2. Esta secuencia requiere $n+1$ accesos a disco. Para esta última secuencia de n ops con ese proc se hicieron $\Omega(1)$ accesos a disco.

b) Cómo implementar para costo amortizado $\Theta(1/B)$ accesos a disco?
Extendemos $|S| = 2B$. Cuando S se llena, llevamos 1 mitad a disco, y al revés cuando se vacía. Como siempre habrá B elementos exact después de cada acceso a disco, siguiendo el proc. no se accesarán al disco durante los próximas B ops. Entonces el costo total de n ops con el proc anterior es $\Theta(n/B)$ y el costo amortizado es $\Theta(1/B)$

C3

1) Alumno borracho

a) ✓

b) Demostrar que no existe alg online det que sea ≤ 5-comp.
por contradicción. Puntos p_1, p_2, p_3 Si $|p_2| > |p_1|$, $\exists$ p "más allá" de p_1 $|p| + \varepsilon < (|p_1|+|p_2|)/2$. Luego costo de ALG para encontrar P es $2|p_1| + 2|p_2| + |p|$. OPT es $|p|$. Supongamos $2|p_1| + 2|p_2| + |p| \le 5|p|$
$\Rightarrow 2|p_1| + 2|p_2| \le 4|p| \Rightarrow |p_1| + |p_2| \le 2|p| \Rightarrow\Leftarrow$
Consideremos $|p_2| \le |p_1|$. Si $|p_2| < |p_1|$ usamos argumento anterior esta vez con p "más allá" de p_2 con $|p| < (|p_2|+|p_3|)/2$.
Supongamos $|p_1| = |p_2|$. Dado que $|p_3| > |p_1| = |p_2|$, $\exists$ P "un poco más allá" de p_2 tq $|p| < (|p_2|+|p_3|)/2$. Por tanto el costo del alg para encontrar P es $2|p| + 2|p_2| + 2|p_3| + |p|$. OPT es $|p|$
Supongamos $2|p_1| + 2|p_2| + 2|p_3| + |p| \le 5|p| \Rightarrow 2|p_1| + 2|p_2| + 2|p_3| \le 4|p|$
$\Rightarrow |p_2| + |p_3| \le 2|p| \Rightarrow\Leftarrow$.

2) Máx de n eltos con n^2 proc en $\Theta(1)$ y CRCW

3) ✓

3) profundidad de cada no de arbol binario, con $\Theta(n)$ procesadores (tiene A, B, C) → A B C (registros) 1 0 -1

Sol: prefix sum! sobre el arreglo que se obtiene de punteros.
La profundidad es la suma de los nodos (registros) en el camino hasta el procesador C de ese nodo.
Luego se hace suma de prefijos en $\Theta(\log n)$ como se vio en clases y se preserva como profundidad lo guardado en C.

a) Búsqueda en $\Theta(\log^2 n) \rightarrow$ búsqueda binaria en todos los arreglos i
Costo $\sum_{i=0}^{k-1} \Theta(\log 2^i) = \Theta(k(k-1)/2) = \Theta(\log^2 n)$

b) Insertar? costo peor caso y costo amortizado.
Insertar elto e en A_0. Si está lleno hacer merge y pasar a A_1. Si ya existía A_1, crear A_2 hacer merge del nuevo y viejo A_1 y así. En el peor caso toma tiempo $\sum_{i=0}^{k} \Theta(2^i) = \Theta(2^k) = \Theta(n)$
Costo amortizado: hacer m inserciones, representado binaria $n_{k-1} \dots n_1 n_0$
el bit n_0 cambia cada vez, el bit n_1 cada 2 veces, n_2 cada 2 veces, etc
cuando bit i cambia 1 a 0 $\Rightarrow$ merge que tiene costo 2^{i+1}. Luego
costo total de m inserciones está acotado $\sum_{i=0}^{k-1} \lfloor m/2^i \rfloor 2^{i+1} \leq 2mk$
$= m\,\Theta(\log n)$, luego el costo de ins es $\Theta(\log n)$

3) Stack en mem secundaria.
Mem ppal buffer S con tamaño B páginas. Cuando se llena e insertamos vaciamos S en disco e insertamos el siguiente en S. Al revés si está vacío. Demostrar costo amortizado $\Omega(1)$ de acc. a disco.
Sol: sea secuencia $2n + (B+1)$ ops: primero insertar B+1 eltos y luego n veces sacar 2 y meter 2. Esta secuencia requiere n+1 accesos a disco. Para esta última secuencia de n ops con ese proc se hicieron $\Omega(1)$ accesos a disco.
b) Cómo implementar para costo amortizado $\Theta(1/B)$ accesos a disco?
Extendemos $|S| = 2B$. Cuando S se llena, llevamos 1 mitad a disco, y al revés cuando se vacía. Como siempre habrá B elementos exactamente después de cada acceso a disco, siguiendo el proc. no se accesarán al disco durante los próximas B ops. Entonces el costo total de n ops con el proc anterior es $\Theta(n/B)$ y el costo amortizado es $\Theta(1/B)$

C3

1) Alumno borracho
a) ✓
b) Demostrar que no existe alg online det que sea ≤ 5-comp.
por contradicción. Puntos p_1, p_2, p_3. Si $|p_2| > |p_1|$, $\exists$ p "más allá" de p_1 tq $|p| < (|p_1| + |p_2|)/2$. Luego costo de ALG para encontrar P es $2|p_1| + 2|p_2| + |p|$. OPT es $|p|$. Supongamos $2|p| + 2|p_2| + |p| \leq 5|p|$
$\Rightarrow 2|p_1| + 2|p_2| \leq 4|p| \Rightarrow |p_1| + |p_2| \leq 2|p| \Rightarrow \Leftarrow$
Consideremos $|p_2| \leq |p_1|$. Si $|p_2| < |p_1|$ usamos arg anterior
esta vez con p "más allá" de p_2 con $|p| < (|p_1| + |p_2|)/2$.
Supongamos $|p_1| = |p_2|$. Dado que $|p_3| > |p_1| = |p_2|$, $\exists$ P "un poco más allá" de p_2 tq $|p| < (|p_2| + |p_3|)/2$. Por tanto el costo del alg para encontrar P es $2|p| + 2|p_2| + 2|p_3| + |p|$. OPT es $|p|$
Supongamos $2|p_1| + 2|p_2| + 2|p_3| + |p| \leq 5|p| \Rightarrow 2|p_1| + 2|p_2| + 2|p_3| \leq 4|p|$
$\Rightarrow |p_2| + |p_3| \leq 2|p| \Rightarrow \Leftarrow$.

2) Máx de n altos con n^2 proc en $\Theta(1)$ y CRCW
a) ✓
b) Demostrar que cada paso realizado por alg CRCW con p proc en ese modelo puede ser simulado con la misma cant. de proc con alg CREW en $\Theta(\log^2 p)$.
Sol: Usar arreglo A tamaño p, y asignar a cada proc P_i la celda A[i] si tal proc quería escribir el dato x_i en el registro l_i en el alg CRCW de a) ahora escribe (l_i, x_i) en A[i] (CREW) Luego se ordena el arreglo por el valor de los l_i's, que toma tiempo $\Theta(\log^2 p)$ en el modelo CREW usando alg odd-even (clases). Esto hace que todos los datos que iban a ser escritos al mismo registro queden juntos en la salida
Ahora cada proc. EREW inspeciona $A[i] = (l_i, x_i)$ y $A[i-1] = (l_j, x_j)$
si $l_i \neq l_j$ o $i = 1$ entonces P_i escribe el dato x_i en registro l_i
De otra forma el proc no hace nada. Como el arreglo está ordenado por la primera coordenada, a lo más 1 proc escribe en cada registro l. Escr. excl.