Nota Cátedra:	
Prom. Controles:	

Universidad Católica del Norte
Facultad de Ingeniería y Ciencias Geológicas
Departamento de Ingeniería de Sistemas y Computación

ESTRUCTURA DE DATOS – CÁTEDRA 3 (II-2018)

- Total: 240 pts (escala 60%) -

Pauta

Nombre - RUT:
Fecha: 19 de enero del 2019.

Competencias a evaluar: análisis de complejidad algorítmica, programación de estructuras de datos básicas utilizando listas enlazadas, programación de estructuras de datos utilizando árboles, identificación de las ventajas y desventajas del paralelismo y concurrencia mediante memoria compartida, y comparación de distintas organizaciones de archivos y estructuras de almacenamiento.

60 pts 1. Determine el orden de complejidad de los algoritmos no-recursivos óptimos para resolver los siguientes problemas (evaluar en el mejor y peor caso). Mostrar el pseudocódigo asociado:

30 a. Dada una matriz A de NxN, con N par, debe determinar en total cuántas veces aparece el valor X en las filas y columnas impares de la misma.

30 b. Invertir el orden de los nodos de una lista con nexo simple L que contiene N datos (N > 0).

60 pts 2. En un árbol B+ con M=3 y M_{HOJA}=4, mostrar paso a paso al:

30 a. Insertar: 50, 70, 22, 14, 33, 2, 91, 46, 67, 34, 23, 48, 13, 40, 99, 19, 30, 108, 88, 1, 6, 25, 10

30 b. Eliminar: 48, 33, 19, 14, 25, 108, 23, 46, 6, 1, 10, 30

60 pts 3. Paralelismo y concurrencia con memoria compartida: responda las siguientes preguntas **con letra clara y ordenada**. Expláyese:

20 a. ¿Por qué no se recomienda implementar soluciones pensando en un hilo por procesador?

20 b. ¿Qué es *mutex*? ¿Para qué sirve? ¿Cuáles son sus operaciones y cómo funcionan?

20 c. Dado el siguiente código en C++, describa qué resultado estaría imprimiendo y por qué:

```
1   #include <thread>
2   #include <iostream>
3
4   void marcarCelda(char* arr, int lo, int hi, char tipoHilo){
5       if (hi-lo == 1)
6           arr[lo] = tipoHilo;
7       else{
8           std::thread izquierda(&marcarCelda, arr, lo, lo+(hi-lo)/2, 'I');
9           std::thread derecha(&marcarCelda, arr, lo+(hi-lo)/2, hi, 'D');
10          izquierda.join();
11          derecha.join();
12      }
13  }
14
15  int main() {
16      int len = 9;
17      char* arreglo = new char[len];
18      marcarCelda(arreglo, 0, len, 'M');
19
20      for(int i=0; i<len; i++)
21          std::cout << arreglo[i] <<" ";
22
23      return 0;
24  }
```

60 pts 4. Organización de archivos e introducción a las bases de datos: responda las siguientes preguntas **con letra clara y ordenada**. Expláyese:

20 a. Según Elmasri & Navathe, ¿de qué formas se pueden resolver las colisiones en el direccionamiento indirecto? Explique cada una brevemente.

20 b. Al utilizar un índice dinámico como un árbol B+ ¿qué estrategias se pueden utilizar para obtener los registros deseados? Nómbrelas y explíquelas.

20 c. Dados los siguientes formatos de archivos, indique cuál debiese ser su clave primaria:

i. Familia.txt:

nombre_persona_1, rut_persona_1, nombre_persona_2, rut_persona_2, relación

```
Sofía, 1.234.567-8, Raúl, 2.345.678-9, hermana
Hugo, 3.456.789-0, Hugo, 4.567.890-K, tío
Hugo, 3.456.789-0, María, 5.678.901-2, esposo
...
Hugo, 3.456.789-0, María, 5.678.901-2, primo
```

ii. Estado_uber.txt:

estado, patente, origen, destino, rut_cliente

```
ocupado, HXHG99, municipalidad, UCN, 4.785.125-9
ocupado, CLZK01, UA, estadio, 16.234.105-K
libre, CTCC13, , ,
...
ocupado, BZMR72, la vega, teatro municipal, 11.453.088-7
```

iii. Notas_EDatos.txt:

rut_alumno, cat1, cat2, cat3, examen, t1, t2, t3, t4, año, semestre

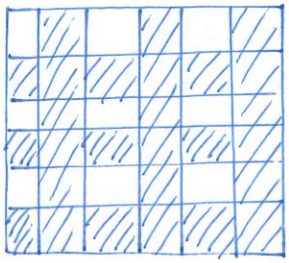
```
20.202.020-2, 1.0, , , , 7.0, 6.5, , , 2018, II
20.202.020-2, 3.0, 3.6, 3.8, 2.0, 7.0, 7.0, 7.0, 6.0, 2018, I
19.191.919-1, 7.0, 6.0, , , 5.0, 4.0, , , 2018, II
...
18.181.818-1, 5.5, 4.0, 4.5, , 4.0, 6.5, 5.0, 5.5, 2017, II
```

iv. Proveedores_restaurant.txt:

producto, proveedor, cantidad, frecuencia_entrega

```
papa, Frutas&Verduras S.A., 50, cada 2 días
limón, Frutas&Verduras S.A., 20, diario
limón, FrootasFrescas, 20, diario
...
bebida gaseosa, The Coca Cola Company, 50, cada 5 días
```

1. a) i) Pseudocódigo:



```
cont = 0;
for (i = 1 → N-1; i = i+2) {
    for (j = 0 → N-1; j++) { // fila completa
        if (A[i,j] = x) {
            cont++;
        }
        if (j%2 = 0 AND A[j,i] = x) {
            cont++;
        }
    }
}
print ("Aparece " + x + " veces");
```

15 pts

ii) Selección O.A: Consultar para verificar si hay sumar al acumulador. 2 pts

```
if (A[i,j] = x)
if (j%2 = 0 AND A[j,i] = x)
```

iii) Conteo O.A: Mejor Caso = Peor Caso → Siempre se debe pasar por la misma cantidad de elementos en función al valor de N. 3 pts

$\frac{n}{2}$ veces {
- $i = 1 \rightarrow \underbrace{\frac{j=0}{2} \quad \frac{j=1}{2} \quad \cdots \quad \frac{j=N-1}{2}}_{n \text{ veces}}$
- $i = 3 \rightarrow \underbrace{\frac{j=0}{2} \quad \frac{j=1}{2} \quad \cdots \quad \frac{j=N-1}{2}}_{n \text{ veces}}$
- $\vdots$
- $i = N-1 \rightarrow \underbrace{\frac{j=0}{2} \quad \frac{j=1}{2} \quad \cdots \quad \frac{j=N-1}{2}}_{n \text{ veces}}$

}

$T(n) = 2 \cdot n \cdot \frac{n}{2} = n^2$ 2 pts

5 pts (ruteo)

iv) Complejidad: $T(n) = n^2 = O(n^2)$ 3 pts

30 pts

b) i) Pseudocódigo:

```
Stack s;
Nodo aux = L.first();
while (aux){
    s.push(aux);
    aux= aux -> next;
}
```

15 pts

```
aux = s.pop();
L.first = aux;
while (!s.isEmpty()){
    aux -> next = s.top();
    aux = s.pop();
}
aux -> next = null;
delete s;
```

30 pts

ii) Selección O.A.: llenado y vaciado de la pila.

```
s.push(aux);
aux = aux -> next;

aux -> next = s.top();
aux = s.pop();
```

5 pts

iii) Conteo O.A.: Mejor Caso = Peor Caso → siempre se deben mover todos los punteros de la lista. 2 pts

> Primer while: - N ciclos
> - 2 operaciones por ciclo
> } 2n

> Segundo while: - N-1 ciclos
> - 2 operaciones por ciclo
> } 2(n-1)

3 pts

$T(n) = 2n + 2(n-1)$

$T(n) = 2n + 2n - 1$

$\boxed{T(n) = 4n - 1}$

2 pts

iv) Complejidad: $T(n) = 4n - 2 = O(n)$ 3 pts

2. I) Índice : cant. máx. hijos = 3 ⇒ cant. máx. claves = 2

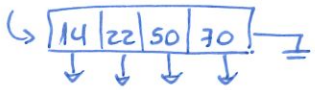
cant. min. hijos = $\lceil \frac{3}{2} \rceil = 2$ ⇒ cant. min. claves = 1

II) Conj. Secuencia : cant. máx. claves = 4

cant. min. claves = $\lceil \frac{4}{2} \rceil = 2$

$j = \left\lceil \frac{M_{HOJA} + 1}{2} \right\rceil = \left\lceil \frac{4+1}{2} \right\rceil = 3$

a) +50
+70
+22
+14

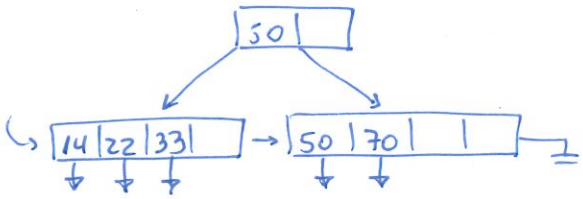


+33 overflow : j entradas en el nodo original

14 22 33 ‖ (50) 70

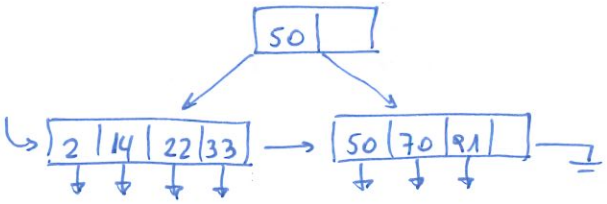
j+1

3 pts

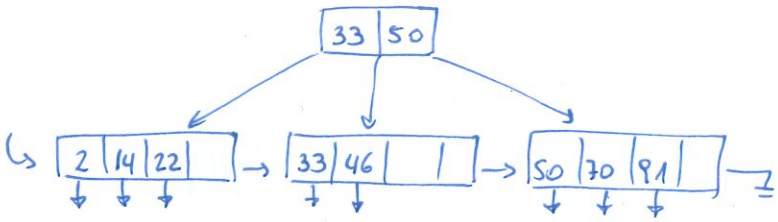


+2

+91

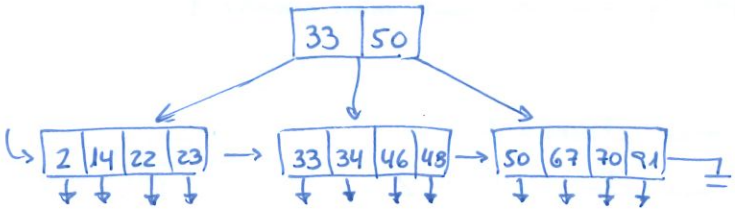


+46 overflow: 2 14 22 ‖ (33) 46



3 pts

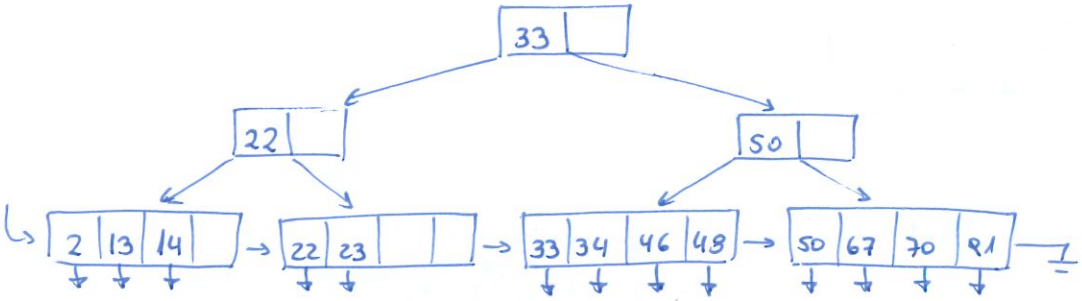
+67
+34
+23
+48



+13 overflow: 2 13 14 ‖ (22) 23

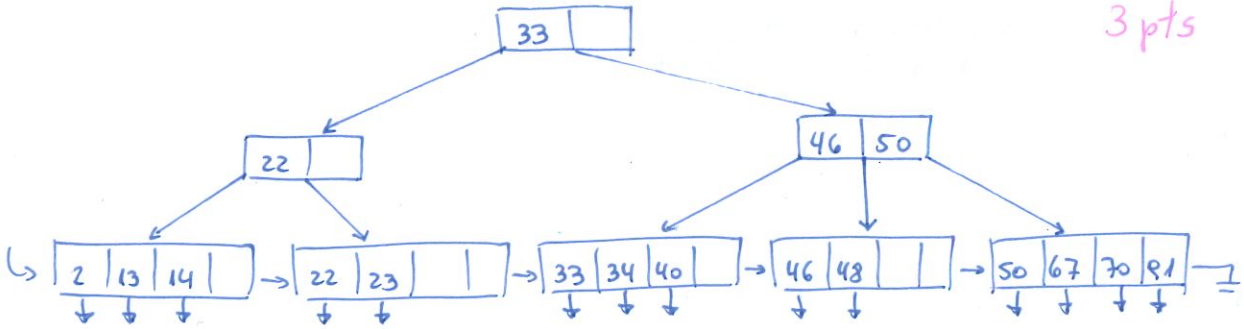
overflow (B): 22 (33) 50 $\lceil \frac{n}{2} \rceil = \lceil \frac{3}{2} \rceil = 2$

6 pts



+40 overflow: 33 34 40 ‖ (46) 48

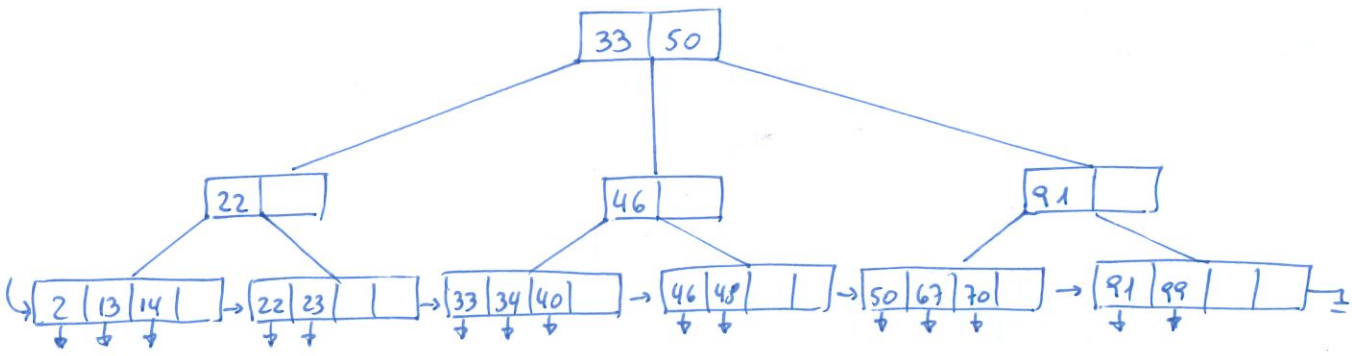
3 pts



+99 overflow: 50 67 70 ‖ (91) 99

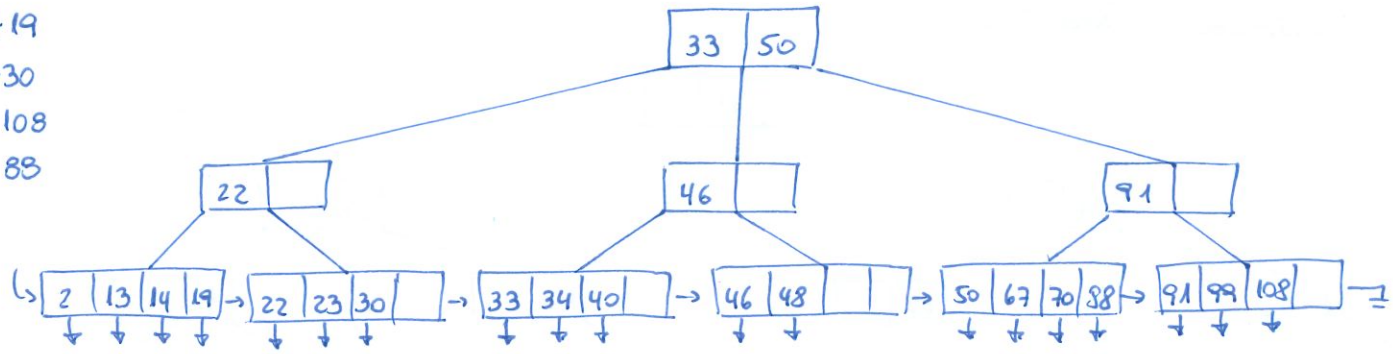
overflow (B): 46 (50) 91

6 pts



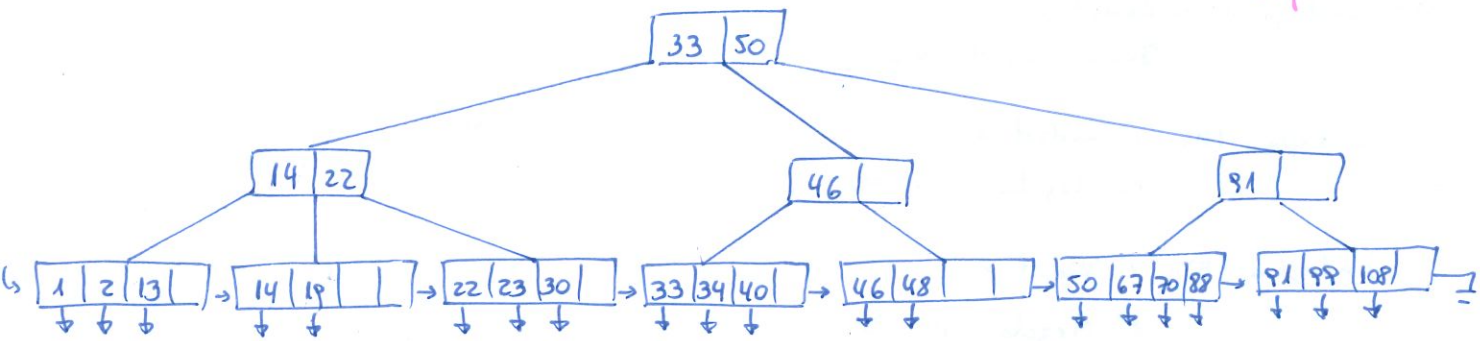
4

+19
+30
+108
+88

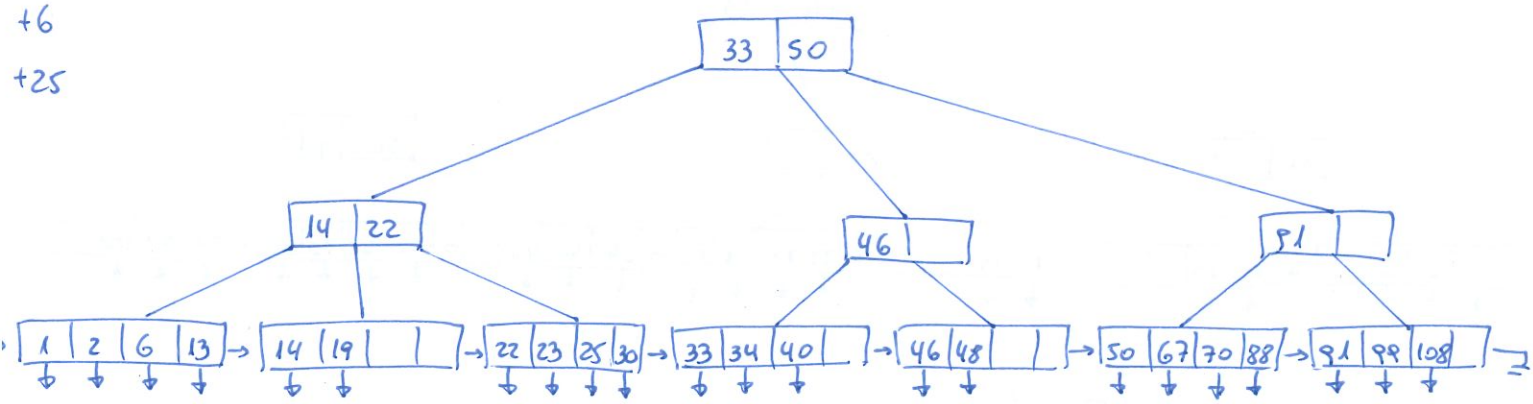


+1 overflow: 1 2 13 || (14) 19

3 pts



+6
+25



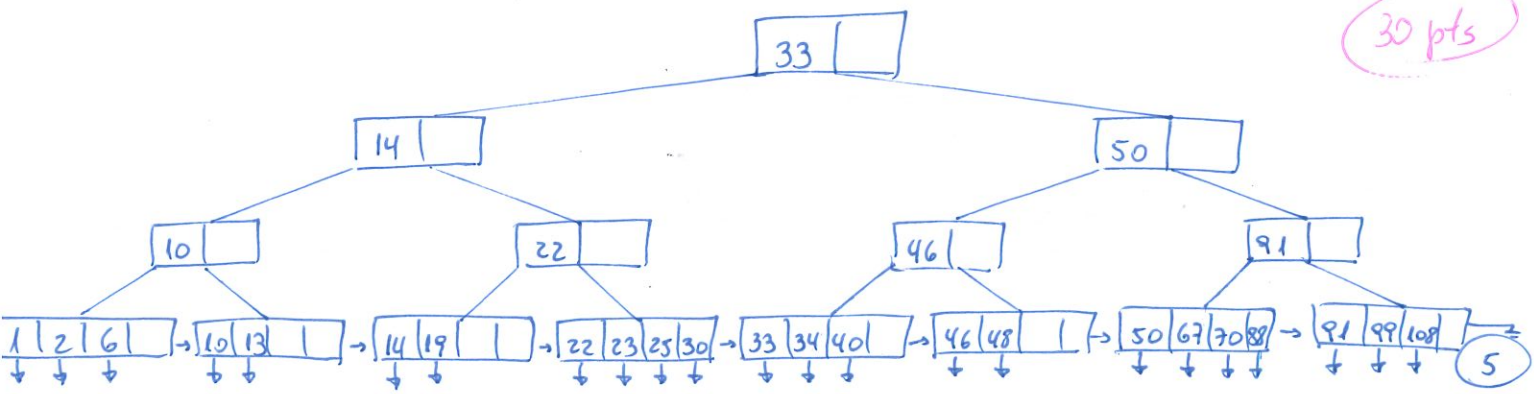
+10 overflow: 1 2 6 || (10) 13

overflow (B): 10 (14) 22

overflow (B): 14 (33) 50

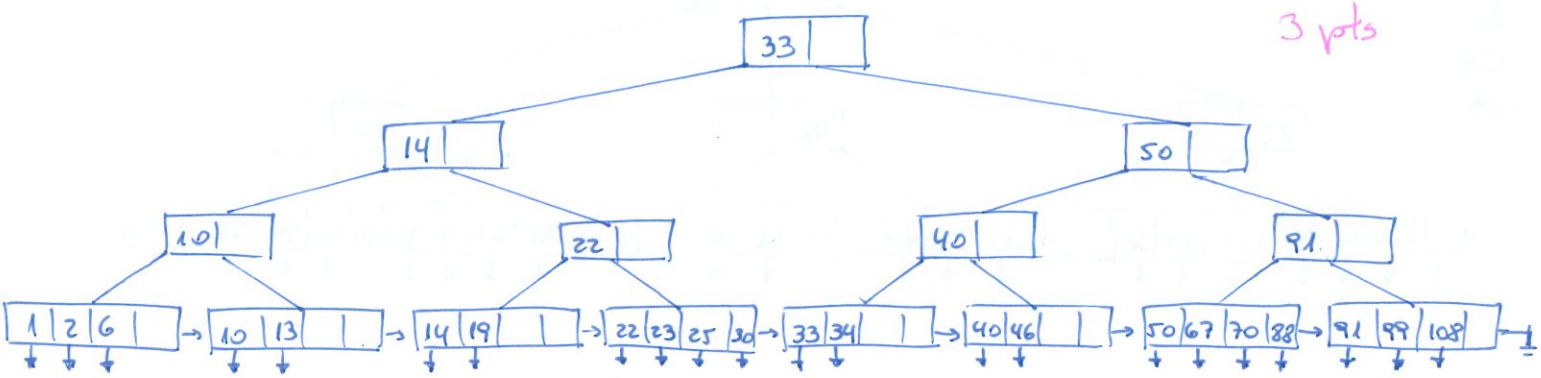
6 pts

30 pts



b) -48 underflow: 1) Redist. ✓

3 pts

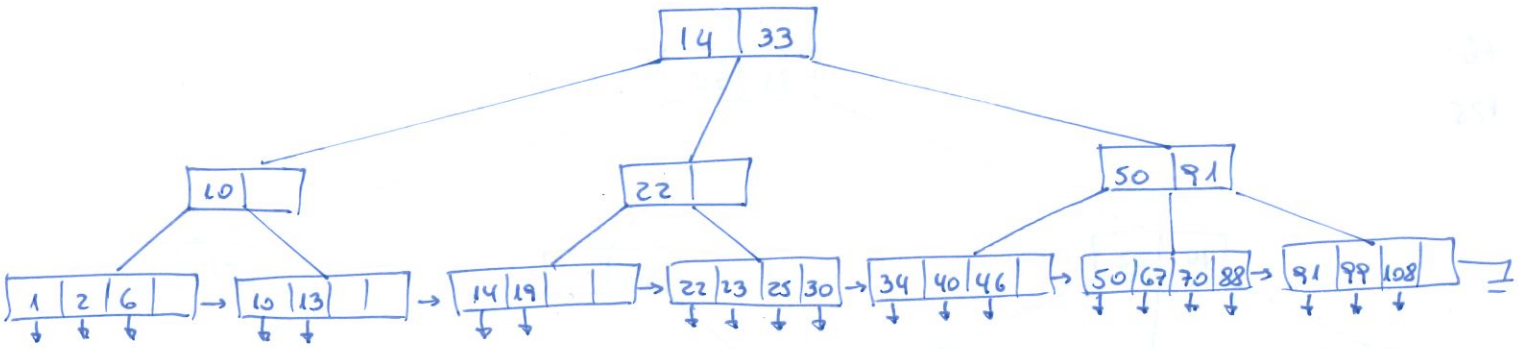


-33 underflow: 1) Redist x
2) Fusión: 34 40 46

underflow (B): 1) Redist x
2) Mezcla: 50 91

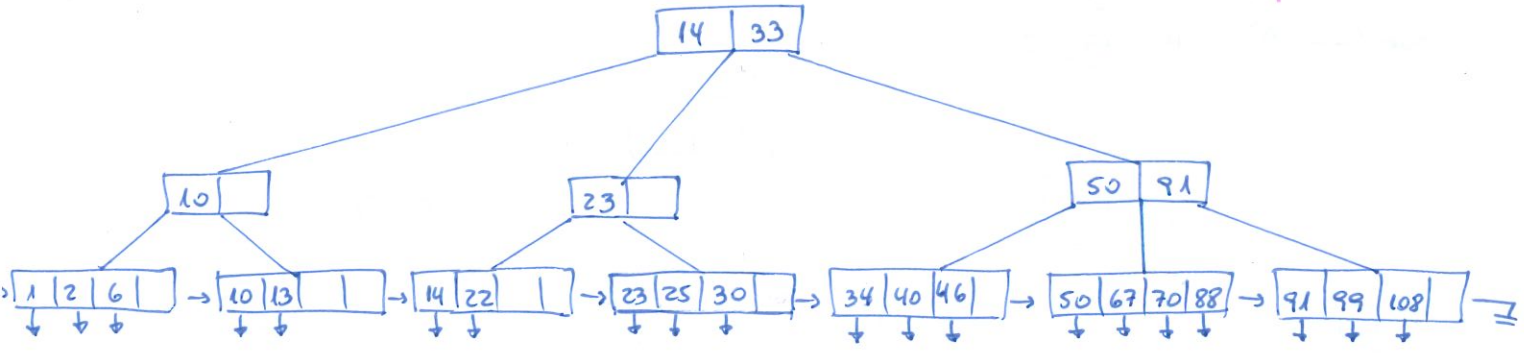
9 pts

underflow (B): 1) Redist x
2) Mezcla: 14 33

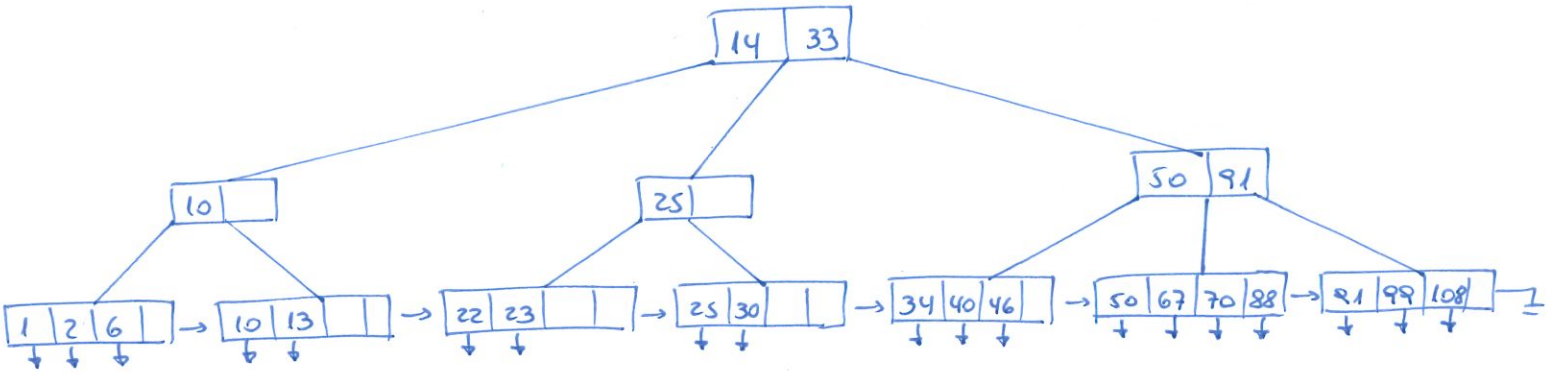


-19 underflow: 1) Redist. ✓

3 pts



-14 underflow: 1) Redist ✓

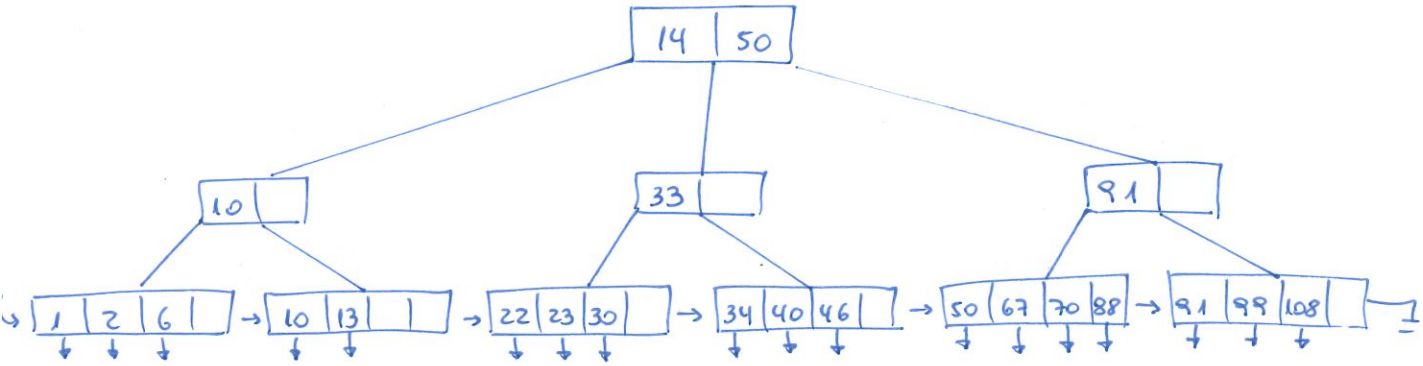


-25 underflow: 1) Redist ×

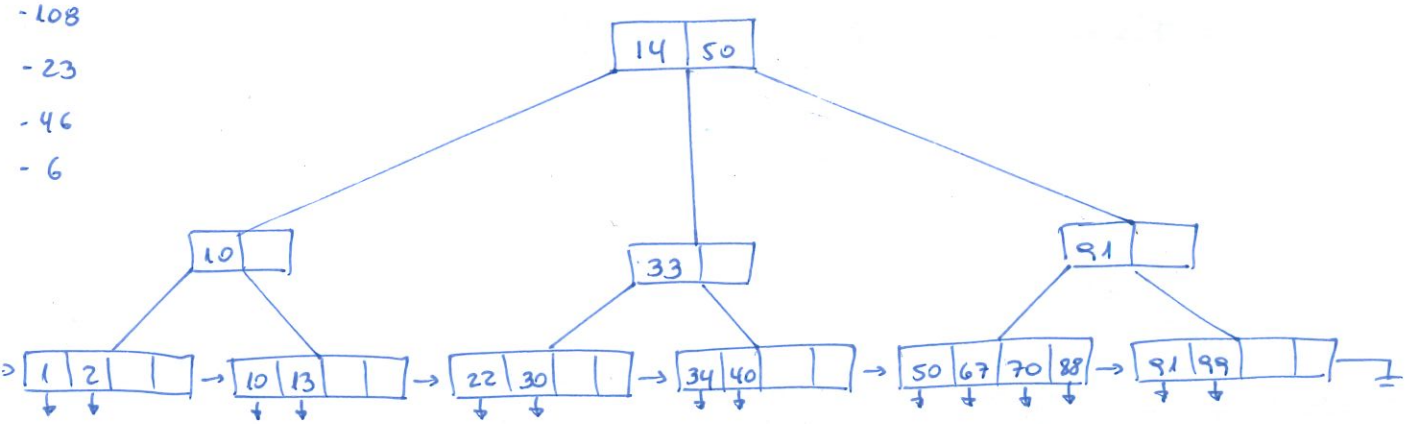
2) Fusión: 22 23 30

6 pts

underflow(B): 1) Redist ✓



- 108
- 23
- 46
- 6



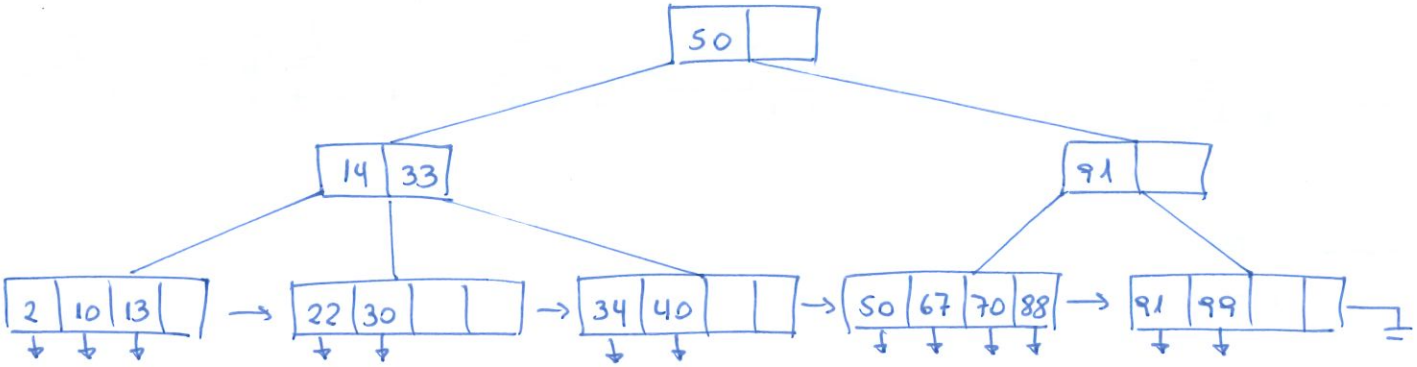
-1 underflow: 1) Redist x

2) Fusión: 2 10 13

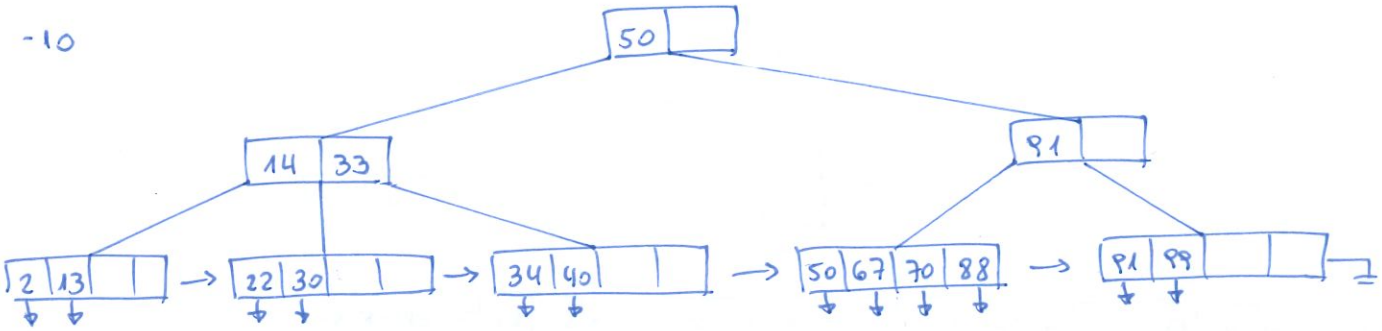
6 pts

underflow: 1) Redist x

2) Mezcla: 14 33



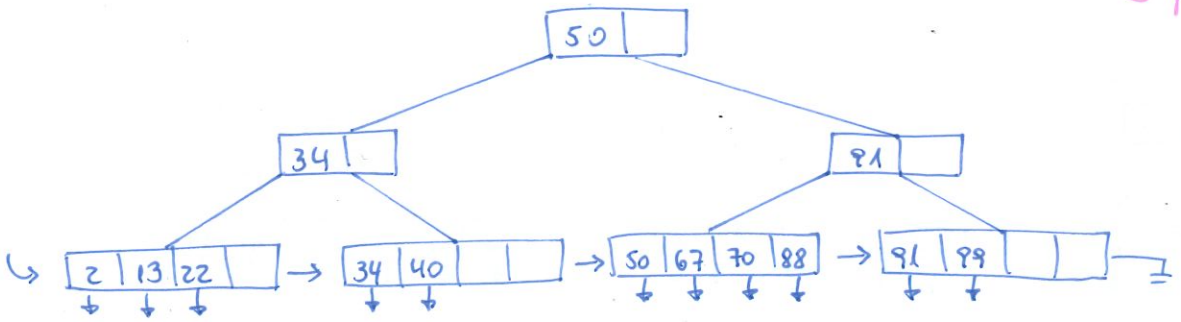
-10



-30 underflow: 1) Redist x

2) Fusión: 2 13 22 34 40 $\lceil \frac{n}{2} \rceil = \lceil \frac{5}{2} \rceil = 3$

3 pts



30 pts

3. a) Porque en la realidad difícilmente se estará ejecutando un hilo por procesador, y con recursos ociosos el programa se vuelve ineficiente. 2 pts

Las razones por las que esta aproximación no suele funcionar son:

i) Distintos computadores tienen disto número de procesadores: 3 pts

Si se define un nº arbitrario de hilos en base a las características del computador en que se codificó el programa, solo será "eficiente" en máquinas con la misma estructura. 3 pts

Si se llegase a ejecutar en un ambiente con mayor cantidad de procesadores, no todos estarían trabajando.

ii) No siempre podemos predecir una división de trabajo en partes iguales: 3 pts

Dependiendo del tipo de problema, se pueden presentar situaciones en las que algunos hilos encuentren más trabajo que otros, según cuáles fueron los datos que le tocó procesar. Si no libera luego el recurso (procesador), atrasa a los demás hilos que deben esperar a que termine su funcionamiento para poder continuar. 3 pts

iii) Los procesadores disponibles para parte del código pueden cambiar: 3 pts

No siempre todos los núcleos de un computador están disponibles para ejecutar nuestro programa. El S.O. y el ambiente de C++ son los que deciden cómo se van asignando los hilos a los procesadores disponibles (scheduling). 3 pts

20 pts

b) Es un lock de exclusión mutua. 3 pts

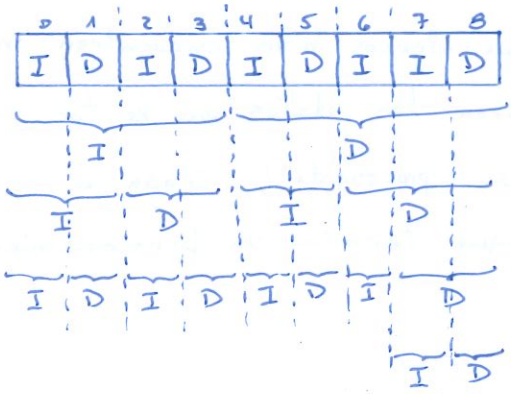
Se utiliza para asegurar que un solo hilo esté trabajando en una sección crítica a la vez. 4 pts

20 pts

Soporta 3 operaciones:

- New: Crea un nuevo lock, inicialmente abierto. 2 pts
  2 pts
- Acquire: Toma un lock y se bloquea hasta que esté abierto (lo que podría ser instantáneamente si el lock no está tomado en primer lugar). Marca el lock como cerrado (held), y retorna. 3 pts
  2 pts
- Release: Toma un lock y lo marca como abierto. Es decir, lo libera. 2 pts
  2 pts

c) Imprimiría el contenido en cada posición del vector:



9 pts

20 pts

Lo que hace el código es marcar en cada celda cuál fue el hilo que llegó a operar en ella:

- D: hilo derecho. Al que se le envía la mitad derecha del rango.
- I: hilo izquierdo. Al que se le envía la mitad izquierda del rango.
- M: hilo principal (main). Quien inicia la ejecución del programa.

11 pts

Al ir dividiendo la solución recursivamente en partes iguales, sólo se llegará a operar efectivamente sobre una celda cuando el rango entregado sea igual a 1.

Así, la división se va haciendo como muestra el dibujo anterior.

4. a) i) Direccionamiento Abierto: 2 pts

Partiendo de la posición ocupada que especifica la dirección, el programa examina las posiciones subsecuentes en orden hasta encontrar una posición no utilizada (vacía). 4 pts

20 pts

ii) Encadenamiento: 2 pts

Se mantienen algunas áreas de desbordamiento, normalmente mediante la extensión del array con varias posiciones de desbordamiento.

Además, se agrega un campo puntero a cada posición del registro. Las colisiones se resuelven colocando el nuevo registro en una posición de desbordamiento desocupada y haciendo que el puntero de la posición ocupada, que le correspondía por su dirección, apunte a la dirección de desbordamiento. 5 pts

iii) Direccionamiento Calculado Múltiple: 2 pts

El programa aplica una segunda función de direccionamiento calculado si la primera produce colisión. Si se produce otra colisión, el programa usa Direccionamiento Abierto o aplica una tercera función de direccionamiento calculado y luego utiliza direccionamiento abierto si fuese necesario. 5 pts

b) i) Búsqueda por llave racimo: desde la raíz busca hasta llegar a la hoja. 2 pts 2 pts

1 pto

ii) Búsqueda secuencial ("scan"): recorre todo a nivel de datos. 2 pts 2 pts

1 pto

iii) Búsqueda por rangos: busca el inicio del rango como en i), y recorre como en ii) hasta el final del rango. 2 pts 2 pts

1 pto

iv) Scan parcial: como en ii), pero se detiene cuando encontró todo lo que buscaba. 2 pts 2 pts

1 pto

c) i) rut_persona 1 + rut_persona 2 + relación 5 pts

ii) patente 5 pts

20 pts

iii) rut_alumno + año + semestre 5 pts

iv) producto + proveedor 5 pts