

Pregunta 1 (1 punt)

Disposem de dos algoritmes que solucionen un mateix problema i n'hem calculat les seves complexitats. En el cas del primer, l'algoritme1, és $\mathcal{O}(n^2)$; i l'altre, l'algoritme2, és $\mathcal{O}(n \log_2 n)$. Com no estem segurs de si ens hem equivocat, hem decidit fer una prova amb la llibreria JMH (que vau usar al Laboratori 1) i els resultats són els següents:

Benchmark	(N)	Mode	Cnt	Score	Error	Units
MyBenchmark.algoritme1	10	avgt	5	0,091 ± 0,011	ms/op	
MyBenchmark.algoritme2	10	avgt	5	0,152 ± 0,027	ms/op	

$$152 - 27 = 130$$

$$91 + 11 = 102 \quad 10^2 = 100 \quad 10 \cdot \log_2 10$$

Podem observar com el segon tarda més que el primer. Què està passant? Ens hem equivocat al calcular els costos? Hi ha alguna altra explicació? Raona la teva resposta.

Pregunta 2 (1 punt)

Donada aquesta funció recursiva múltiple:

```
static int combinations(int n, int k) {
    assert n >= 0 && k >= 0;
    if (k > n) {
        return 0;
    } else if (k == 0 || k == n) {
        return 1;
    } else {
        int c1 = combinations(n - 1, k - 1);
        int c2 = combinations(n - 1, k);
        return c1 + c2;
    }
}
```

Defineix l'enumerat EntryPoint i la classe Context que utilitzaríem en el seu pas a iteratiu segons la metodologia explicada al Laboratori 2.

Pregunta 3 (1 punt)

Degut al tractament que fa Java dels tipus genèrics, les següents instruccions compilen (només generen un warning):

```
List<Integer> counters = new ArrayList<Integer>();
List alias = counters;
```

Explica, amb l'ajuda d'un exemple, algun dels problemes que el codi anterior pot provocar.

Estructures de dades (primer parcial)

Problema 4 (3 punts)

Implementa **usant iteradors**, ja que ha de funcionar en temps lineal tant quan es treballa sobre un `ArrayList<E>` com quan es treballa sobre una `LinkedList<E>`, una funció que rebi una llista i n'elimini els elements que ocupen una posició parella (el segon, el quart, el sisè, etc.).

```
static <E> void removeInPairPosition(List<E> elements) {
    assert elements != null;
    ¿?
}
```

Un exemple d'utilització, és:

```
public static void main(String[] args) {
    var elems = List.of("a", "b", "c", "d", "e", "f", "g");
    var list = new ArrayList<>(elems);
    removeInPairPosition(list);
    System.out.println(list);
}
```

que escriu [a, c, e, g]

Problema 5 (4 punts)

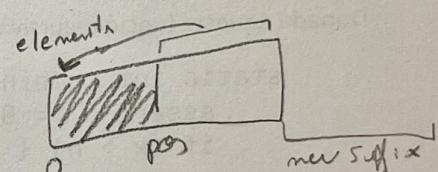
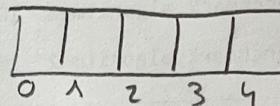
Donada la següent representació de les `ArrayList<E>`:

```
public class ArrayList<E> {
    Object[] elements = new Object[5];
    int size          = 0;
    int modCount      = 0;

    public ArrayList() { }

    public void shiftSuffixLeft(
        int position,
        ArrayList<? extends E> newSuffix) {
        ¿?
    }

    @Override
    public String toString() {
        return "ArrayList{" +
            "elements=" + Arrays.toString(elements) +
            ", size=" + size +
            ", modCount=" + modCount +
            '}';
    }
}
```



Implementa l'operació mostrada `shiftSuffixLeft` l'efecte de la qual és:

- Mou el sufix que comença a `position` de l'`ArrayList` receptor al seu principi (els elements del prefix que acaba a `position` desapareixen de la llista).
- Col·loca els elements del `newSuffix` al final del nou prefix

Aquesta operació llençarà excepcions (amb missatges adequats) en les següent condicions

- Si la `position` no és vàlida, és a dir, no està en el rang `[0, size]`, es llença `IndexOutOfBoundsException`
- Si la referència `newSuffix` és `null` es llença `NullPointerException`

La política de redimensionat serà la de doblar la mida de l'array fins a que hi hagi prou capacitat per guardar el resultat. Tots els moviments i copies d'elements als arrays s'han de programar i no es poden fer servir mètodes de la llibreria estàndard.

Tingueu en compte que la llista que es rep com a paràmetre és una instància de la pròpia classe.

Noteu els següents casos "extrems", per entendre millor el funcionament esperat del mètode:

- Si `position` és zero, l'operació afegeix el contingut de `newSuffix` a la llista receptora
 - I si `newSuffix` és buit, l'operació no fa res
- Si `position` és `size`, l'operació copia el contingut de `newSuffix` a la llista receptora
 - I si `newSuffix` és buit, la llista receptora quedarà buida
- Si `newSuffix` és buida, l'operació elimina el prefix `[0, position)` de la llista receptora

Feu diagrames per mostrar, pas a pas, els moviments dels elements de l'array. **El codi només serà avaluat després dels vostres diagrames i explicacions.**

Definint el `main` dins de la pròpia classe `ArrayList<E>`, podem fer:

```
public static void main(String[] args) {
    var suffix = new ArrayList<String>();
    suffix.elements = new Object[]{"X", "Y", "Z", "U", null};
    suffix.size = 4;
    suffix.modCount = 33;
    var list = new ArrayList<String>();
    list.elements = new Object[]{"a", "b", "c", "d", "e"};
    list.size = 5;
    list.modCount = 42;
    list.shiftSuffixLeft(3, suffix);
    System.out.println("list = " + list);
}
```

Programa que escriu:

```
list = ArrayList{elements=[d, e, X, Y, Z, U, null, null, null, null], size=6,
modCount=43}
```

Annex:

- interface Iterator<E>
 - boolean hasNext()
 - E next()
 - void remove()
- interface ListIterator<E> extends Iterator<E>
 - boolean hasPrevious()
 - E previous()
 - int nextIndex()
 - int previousIndex()
 - void set(E e)
 - void add(E e)
- interface List<E>
 - Iterator<E> iterator()
 - ListIterator<E> listIterator()
 - ListIterator<E> listIterator(int index)

Comparable

Comparable(A, B)
Comparable
CompareTo(b)