# Solució de l'Examen Parcial EDA 08/11/2022

## Proposta de solució al problema 1

(a) El bucle que inicialitza el vector $v$ triga temps $\Theta(n)$. El mateix passa amb la crida a *random_shuffle*.

Pel que fa als bucle ennierats, fixem-nos primer que dins el vector $v$ hi posem tots els nombres entre 1 i $n$, i a continuació els reordenem de manera aleatòria. Si no els haguéssim ordenat, per cada valor de $i$, el bucle més intern faria $v[i] = i+1$ voltes (i per tant incrementaria $s$ en una unitat $i+1$ vegades). Com que $i$ es mou des de 0 fins a $n-1$, el nombre de voltes totals del bucle intern seria $1+2+3+\ldots+n = n(n+1)/2$. Així doncs, podem afirmar que el codi sense l'ordenació calcula $n(n+1)/2$ i té cost $\Theta(n(n+1)/2) = \Theta(n^2)$.

L'única cosa que canvia degut a l'ordenació de $v$ és que deixa ser cert que per $i = 0$ el bucle intern faci 1 volta, que per $i = 1$ en faci 2, que per $i = 2$ en faci 3, etc. En el nostre codi, per cada valor de $i$ el bucle intern farà $v[i]$ voltes. Però com que $v$ és una permutació de $\{1, 2, \ldots, n\}$, hi haurà una iteració on es farà 1 volta, una altra on se'n faran 2, una altra on se'n faran 3, etc. Per tant, el codi calcula el mateix amb el mateix cost asimptòtic.

(b) El seu cost és $\Theta(n \log\log n)$. Anem a veure per què.

El bucle extern dona $n$ voltes, i el cost del bucle intern és independent del valor de $j$. Per tant, el cost total serà $n$ vegades el cost del bucle intern.

Pel que fa al bucle intern, aquest s'atura quan $k \geq n$. En entrar a la primera iteració $k$ val 2, a la segona val 4, a la tercera val 16, a la quarta val 256, etc. Podem afirmar que a la iteració $i$-èsima $k$ val $2^{2^i}$. Es donaran voltes, per tant, mentre $2^{2^i} < n$, el que equival a que $2^i < \log n$, i que $i < \log\log n$. Per tant, el bucle intern fa $\Theta(\log\log n)$ iteracions i el cost total del codi és $\Theta(n \log\log n)$.

(c) Per analitzar el cost de l'algorisme d'inserció, podem comptar el nombre d'intercanvis que s'han de fer. Sabem que el nombre d'intercanvis que es faran quan s'hagi de col·locar un nombre a la posició que li pertoca es correspon al nombre d'elements a la seva esquerra que són estrictament majors que ell en la configuració inicial.

Els dos primers nombres no tenen cap element major a la seva esquerra. Pel 2 i pel $2n-1$ en tenim 1 per a cadascun. Pel 3 i pel $2n-2$ en tenim 2 per a cadascun. Pel 4 i pel $2n-3$ en tenim 3 per a cadascun, i així successivament, fins a la parella $n$, $n+1$ pels que en tenim $n-1$ per cadascun. Així doncs, el nombre d'intercanvis serà $1+1+2+2+3+3+\ldots+(n-1)+(n-1) = 2 \cdot n(n-1)/2 = n(n-1)$, que és $\Theta(n^2)$. Per tant, aquest és el cost de l'algorisme d'ordenació per inserció per aquesta entrada.

## Proposta de solució al problema 2

(a) En el cas pitjor, realitzarem totes les iteracions ($n$) al bucle que hi ha dins *inefficient* i acabarem retornant $n$. A cada bucle es crida a la funció *find*. Com que es passa el vector per referència, el pas de paràmetres és $\Theta(1)$. Totes les altres operacions dins *inefficient* tenen cost $\Theta(1)$, pel que només ens hem de centrar en les crides a *find*.

Podem veure que *find* és una funció recursiva, on el que decreix és el valor de *pos*, que inicialment és $n-1$. Com que totes les operacions són constants, el seu cost ve descrit per la recurrència $T(n) = T(n-1) + \Theta(1)$, que té solució $T(n) \in \Theta(n)$.

Per tant, com que cada crida a *find* té cost $\Theta(n)$ i es fan $n$ crides, el cost total és $\Theta(n^2)$.

(b) Una possible solució és:

```
int efficient (const vector<int>& v, int l, int r) {
  if (l > r) return v.size();
  int m = (l+r)/2;
  if (v[m] > m) {
    if (m == l or v[m−1] == m−1) return m;
    else return efficient (v,l,m−1);
  }
  else return efficient (v,m+1,r); // we know v[m] == m
}
```