Cognoms **Nom** **DNI**

Examen Parcial EDA **Duració: 1h 30min** **06/11/2020**

- *L'enunciat té 3 fulls, 6 cares, i 2 problemes.*
- *Poseu el vostre nom complet i número de DNI a cada problema.*
- *Contesteu tots els problemes en el propi full de l'enunciat a l'espai reservat.*
- *Llevat que es digui el contrari, sempre que parlem de cost ens referim a cost asimptòtic en temps.*
- *Llevat que es digui el contrari,* **cal justificar les respostes.**

Problema 1 **(2.5 pts.)**

Respon a les següents preguntes:

(a) (1.5 pts.) Donat un vector v d'enters ordenats creixentment i un enter x, volem determinar si x apareix a v. En lloc d'implementar una cerca binària, ens proposen la idea d'escollir dos elements que parteixin el vector en tres parts iguals i determinar en quina d'aquestes tres parts cal buscar x. Completa el següent codi perquè sigui una implementació correcta d'aquesta idea:

```
bool tri_search (const vector<int>& v, int l, int r, int x) {
  if (l > r) return false;
  else {
    int n_elems = (r−l+1);
    int f = l + n_elems/3;
    int s = r − n_elems/3;

    if (                    ) return true;
    if (            ) return tri_search(v,       ,       , x);
    if (            ) return tri_search(v,       ,       , x);
    return tri_search (v,       ,       , x);
  }
}

bool tri_search (const vector<int>& v, int x) {
  return tri_search (v,0,v.size()−1, x);
}
```

Si n és el nombre d'elements del vector v, analitzeu el cost en cas pitjor d'una crida $tri\_search(v, x)$ en funció de n.

(b) (1 pt.) Doneu dues funcions f i g amb $f \notin \Theta(g)$ tals que ambdues siguin $\Omega(n)$ i $O(n \log n)$ però que cap sigui $\Theta(n)$ ni $\Theta(n \log n)$.

Cognoms Nom DNI

Problema 2 **(7.5 pts.)**

Donat un vector v d'n naturals volem determinar si existeix un element *dominant*, és a dir, si existeix un element que apareix més de $n/2$ vegades. Per exemple:

- Si $v = \{5, 2, 5, 2, 8, 2, 2\}$, aleshores 2 és l'element dominant perquè apareix $4 > 7/2$ vegades.
- Si $v = \{3, 2, 3, 3, 2, 3\}$, aleshores 3 és l'element dominant perquè apareix $4 > 6/2$ vegades.
- Si $v = \{6, 1, 6, 1, 6, 2, 9\}$, no hi ha cap element dominant perquè cap d'ells apareix més de $7/2$ vegades.

Volem obtenir una funció en C++ que rebi el vector v i retorni l'element dominant de v, o el nombre -1 en cas que no existeixi cap element dominant.

(a) (2.5 pts.) Un estudiant de *PRO2* ens suggereix la següent solució:

```
int dominant_pro2 (vector<int> v) {
  int n = v.size ();
  for (int i = 0; i < n; ++i) {
    if (v[i] ≠ −1) {
      int times = 0;
      int candidate = v[i];
      for (int j = i; j < n; ++j) {
        if (v[j] == candidate) {
          ++times;
          v[j] = −1;
          if (times > n/2) return candidate;
        } } } }
  return −1;
}
```

Analitzeu el seu cost en cas pitjor en funció de n. Expliqueu com construiríeu un vector de mida n pel qual es doni aquest cas pitjor.

Analitzeu el seu cost en cas millor en funció de n. Expliqueu com construiríeu un vector de mida n pel qual es doni aquest cas millor.

Si ens asseguren que, per a qualsevol n, el vector v sempre tindrà com a molt 100 naturals diferents, canviaria el seu cost en cas pitjor?

(b) (2 pts.) Un altre estudiant de *PRO2* se n'adona que si primer ordenem el vector existeix un algorisme ben senzill:

```
int dominant_sort (vector<int> v) {
  int n = v.size ();
  own_sort(v.begin (), v.end ());
  int i = 0;
  while (i < n) {
    int times = 0, j = i;
    while (j < n and v[j] == v[i]){
      ++times;
      ++j;
    }
    if (times > n/2) return v[i];
    i = j;
  }
  return −1;
}
```

Cognoms **Nom** **DNI**

Si *own_sort* es correspon a una ordenació per inserció, quin és el cost en cas millor i pitjor de *dominant_sort* en funció de n?

Si *own_sort* implementa un *quicksort*, quin és el cost en cas millor i pitjor de *dominant_sort* en funció de n?

(c) (3 pts.) Finalment, un estudiant d'*EDA* molt aplicat, encara que no brillant, ens suggereix una solució basada en dividir i vèncer. No obstant, s'han perdut parts del codi i us demanem que completeu la següent funció:

```
int times (const vector<int>& v, int l, int r, int x) {
  if (l > r) return 0;
  return (v[l] == x) + times(v,l+1,r,x);
}

int dominant_divide (const vector<int>& v, int l, int r) {
  if (l == r) return v[l];
  int n_elems = (r−l+1);
  int m = (l+r)/2;
  int maj_left  = dominant_divide(v, ______ , ______ );
  if ( maj_left  ≠ −1 and times( __________ ) > n_elems/2) return ______ ;
  int maj_right = dominant_divide(v, ______ , ______ );
  if (maj_right ≠ −1 and times( __________ ) > n_elems/2) return ______ ;
  return −1;
}

int dominant_divide (const vector<int>& v) {
  return dominant_divide(v,0,v.size()−1);
}
```

Analitzeu el cost de *dominant_divide* en cas pitjor en funció de n.