Cognoms **Nom** **DNI**

Examen Parcial EDA **Duració: 2h** **02/11/2023**

- *L'enunciat té 3 fulls, 6 cares, i 2 problemes.*
- *Poseu el vostre nom complet i número de DNI a cada full.*
- *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 **(3.5 pts.)**

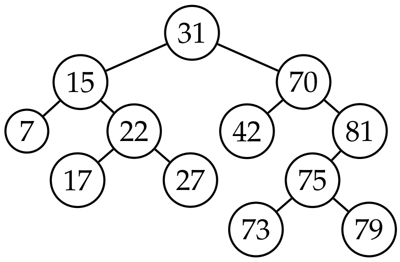
Respon a les següents preguntes:

(a) (1 pt.) Cada fila de la següent taula representa una certa etapa de l'execució d'un algorisme d'ordenació conegut (selecció, inserció, quicksort i mergesort) sobre el vector | 10 | 2 | 5 | 3 | 7 | 13 | 1 | 6 |. Escriviu al costat de cada fila a quin d'aquests algorismes correspon. Cada algorisme ha d'aparèixer exactament una vegada. 1 o cap resposta correcta puntuarà 0 punts i 2 respostes correctes, 0.5 punts. Si repetiu algun algorisme, us atorgarem 0 punts. No cal que justifiqueu la vostra resposta.

2	3	5	10	1	6	7	13	
2	3	5	10	7	13	1	6	
6	2	5	3	7	1	13	10	
1	2	3	5	6	13	10	7	

(b) (1 pt.) Ordena les funcions $(\log_2 \log_2 n)^2$ i $\log_2 n$ segons el seu creixement asimptòtic. Recorda que cal justificar la resposta.

(c) (1.5 pts.) Sigui T un arbre binari de cerca que implementa un diccionari, on les claus són nombres enters. Donades dues claus diferents x, y, que apareixen a T, definim el *menor ancestre comú de x i y* com l'ancestre comú de x i y que té major distància respecte l'arrel. Per exemple, en el següent arbre el menor ancestre comú de 7 i 17 és el 15, el de 42 i 73 és el 70, i el de 15 i 17 és 15.



Descriviu a alt nivell (no cal que doneu codi en C++) un algorisme per calcular el menor ancestre comú de dos claus x i y que compleixen $x < y$. Es valorarà l'eficiència (no només asimptòtica) de l'algorisme.

Cognoms **Nom** **DNI**

Problema 2 **(6.5 pts.)**

Donat un vector v d'n nombres naturals, volem calcular un vector que contingui totes les parelles $<z, t>$ tal que el nombre z apareix a v exactament t vegades, amb $t > 0$. L'ordre de les parelles en el vector no ens importa. Per exemple, donat el vector $(4, 1, 5, 1, 3, 4, 5, 1)$ un resultat correcte seria $(<3, 1>, <5, 2>, <1, 3>, <4, 2>)$.

(a) (1 pt.) Considereu el següent codi, que resol el problema plantejat:

```
vector<pair<int,int>> sort_and_process (vector<int>& v) {
  vector<pair<int,int>> res;
  sort (v.begin (), v.end ());
  int i = 0;
  while (i < v.size ()) {
    int elem = v[i];
    int times = 0;
    while (i < v.size () and v[i] == elem) {++times; ++i;}
    res.push_back({elem, times});
  }
  return res;
}
```

Quin és el cost en cas pitjor del programa anterior en funció d'n si la funció *sort* implementa una ordenació per inserció? I si implementa un mergesort? *Nota:* en tot aquest problema assumiu que el cost d'un *push_back* és $\Theta(1)$.

(b) (2 pts.) Considerem una segona solució al mateix problema:

```
vector<pair<int,int>> mark_and_process (vector<int>& v) {
  vector<pair<int,int>> res;
  for (int i = 0; i < v.size(); ++i){
    if (v[i] ≠ −1) {
      int times = 1, elem = v[i], j = i+1;
      while (j < v.size()) {
        if (v[j] == elem) {++times; v[j] = −1;}
        ++j;
      }
      res.push_back({elem,times});
    }
  }
  return res;
}
```

Quin és el cost en cas pitjor d'aquest programa en funció d'n? I el seu cost en cas millor? Doneu dos vectors pels quals s'apliqui el cas pitjor i millor, respectivament.

Cognoms **Nom** **DNI**

(c) (1.5 pts.) Ens proporcionen ara una solució al problema basada en dividir i vèncer:

```
vector<pair<int,int>> combine (const vector<pair<int,int>>& v1,
                               const vector<pair<int,int>>& v2) {
  vector<pair<int,int>> res;
  vector<bool> present(v2.size(), false);
  for (int i = 0; i < v1.size(); ++i) {
    int elem = v1[i].first;
    int times = v1[i].second;
    for (int j = 0; j < v2.size(); ++j) {
      if (v2[j].first == elem) {
        times += v2[j].second;
        present[j] = true;
      }
    }
    res.push_back({elem, times});
  }
  for (int j = 0; j < v2.size(); ++j)
    if (not present[j]) res.push_back(v2[j]);
  return res;
}

vector<pair<int,int>> merge_count (const vector<int>& v, int l, int r) {
  if (r == l) return {{v[l],1}};
  else {
    int m = (l+r)/2;
    vector<pair<int,int>> r1 = merge_count(v,l,m);
    vector<pair<int,int>> r2 = merge_count(v,m+1,r);
    return combine(r1,r2);
  }
}

vector<pair<int,int>> merge_count (const vector<int>& v) {
  return merge_count(v,0,v.size()-1);
}
```

Quin és el cost en cas pitjor d'una crida a $merge\_count(v)$ en funció d'n?

(d) (2 pts.) Finalment, ens demanen modificar la funció *combine* per tal que una crida a $merge\_count(v)$ tingui cost $\Theta(n \log n)$ en cas pitjor. Ompliu el codi següent per tal que proporcioni una solució correcte i tingui el cost desitjat.

Pista: intenteu que *merge_count* retorni el vector de parelles **ordenat**.

```
vector<pair<int,int>> combine (const vector<pair<int,int>>& v1,
                               const vector<pair<int,int>>& v2) {
  vector<pair<int,int>> res;
  int i = 0, j = 0;
  while (i < v1.size() and j < v2.size()) {



  }
  while (i < v1.size()) {res.push_back(v1[i]); ++i;}
  while (j < v2.size()) {res.push_back(v2[j]); ++j;}
  return res; }
```

Cognoms **Nom** **DNI**

Examen Parcial EDA **Duració: 2h** **21/04/2023**

- *L'enunciat té 4 fulls, 8 cares, i 2 problemes.*
- *Poseu el vostre nom complet i número de DNI a cada full.*
- *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 **(4.5 pts.)**

Donats dos vectors de nombres naturals v_1 i v_2 de mida $n > 0$, volem determinar si podem trobar un element a cada vector tal que, si els intercanviem, els dos vectors resultants sumin el mateix. Per exemple, si $v_1 = (6, 4, 3, 9)$ i $v_2 = (7, 0, 2, 5)$, aleshores intercanviant el 6 i el 2 obtenim dos vectors que sumen el mateix. En canvi, si $v_1 = (2, 0, 9, 5)$ i $v_2 = (2, 4, 5, 7)$ no existeix cap parell d'elements amb la propietat desitjada.

(a) (1 pt.) Considereu la solució següent al problema plantejat:

```
int suma (const vector<int>& v){
  int s = 0;
  for (int x : v) s += x;
  return s;
}

pair<int,int> sol_facil (vector<int>& v1, vector<int>& v2) {
  for (int i = 0; i < v1.size (); ++i)
    for (int j = 0; j < v2.size (); ++j) {
      swap(v1[i], v2[j]);
      int s1 = suma(v1);
      int s2 = suma(v2);
      swap(v1[i], v2[j]);
      if (s1 == s2) return {v1[i], v2[j]};
    }
  return {−1, −1}; // No hi ha solució
}
```

En funció d'n, quin és el cost en el cas pitjor d'una crida a sol_facil?

(b) (2 pts.) Completeu el codi següent per tal que sigui una solució vàlida al problema plantejat:

```
bool cerca (const vector<int>& v, int x, int e, int d) {
  if (e > d) return false;
  else {
    int m = (e+d)/2;
    return v[m] == x or cerca(v,x,e,m−1) or cerca(v,x,m+1,d);
  }
}
pair<int,int> sol (const vector<int>& v1, const vector<int>& v2) {
  int dif = suma(v2) − suma(v1); // suma és la funció de l'apartat anterior
  if (dif%2 ≠ 0) return {−1,−1};
  for (int i = 0; i < v1.size(); ++i) {
    int x = [          ]
    if (cerca(v2, x, 0, v2.size()−1)) return {v1[i], x};
  }
  return {−1,−1};
}
```

Analitzeu, en funció d'n, el cost en el cas pitjor d'una crida a sol.

Cognoms **Nom** **DNI**

(c) (1.5 pts.) Expliqueu com modificaríeu la solució de l'apartat anterior per tal que el seu cost, en el cas pitjor, sigui millor asimptòticament. No cal que doneu codi concret, una descripció a alt nivell serà suficient. Quin és el cost, en el cas pitjor, de la nova solució?

Aquesta cara estaria en blanc intencionadament si no fos per aquesta nota.

Cognoms **Nom** **DNI**

Problema 2 **(5.5 pts.)**

Donat dos nombres naturals n, k diferents de zero, volem expressar n com la suma ordenada d'exactament k potències de 2 amb exponent no negatiu (és a dir, 2^{-3} no la considerem una potència de 2 vàlida). Per exemple, si $n = 21$ i $k = 4$, una solució és $21 = 2^3 + 2^3 + 2^2 + 2^0$. Fixem-nos que l'ordre desitjat és decreixent.
Observació: la representació en binari d'n ens dona una manera d'expressar n com a suma de potències de 2, però pot no tenir exactament k sumands. De fet, és la representació amb el menor nombre de sumands possible.

(a) (0.75 pts). Escriviu solucions per a $n = 10$ i $k = 2, 3, 4$ i 5. No cal justificar com les obteniu.

(b) (1.25 pts.) Quines són les dues úniques situacions on no hi ha solució?

(c) (1.5 pts.) Completeu la funció següent per tal que donat un natural n retorni, ordenades de menor a major, les posicions de la representació en binari d'n on apareix un 1. Per exemple, si $n = 19$, cal retornar el vector $(0, 1, 4)$.

```
vector<int> pos_uns (int n) {
  vector<int> v;
  int pos = 0;

  return v;
}
```

Quin és el seu cost en funció d'n?

Cognoms **Nom** **DNI**

(d) (2 pts.) Completeu el codi següent per tal que resolgui el problema plantejat:

```
void escriu_suma_potencies (int n, int k) {
  vector<int> uns = pos_uns(n);
  if (                                   )
    cout << "No hi ha solucio" << endl;
  else {
    priority_queue<int> Q;



    bool primer = true;
    while (not Q.empty()){
      if (not primer) cout << " + ";
      else primer = false;
      cout << "2^" << Q.top();
      Q.pop();
    }
    cout << endl;
  }
}
```

Aquesta cara estaria en blanc intencionadament si no fos per aquesta nota.

Cognoms **Nom** **DNI**

Examen Parcial EDA   Duració: 1h 30min   03/11/2022

- *L'enunciat té 3 fulls, 6 cares, i 2 problemes.*
- *Poseu el vostre nom complet i número de DNI a cada full.*
- *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 (4 pts.)

Responeu les preguntes següents:

(a) (2 pts.) Considereu el codi següent:

```
int f (int x, int n) {
  if (n == 1) return x;
  else {
    int tmp = f(x,n−1);
    int res = 0;
    for (int i = 0; i < x; ++i) res += tmp;
    return res;
  }
}

int main(){
  int N;
  cin >> N;
  cout << f(N,N) << endl;
}
```

Si assumim que $N \geq 1$, què escriu per pantalla el programa anterior? Justifica la teva resposta.

Quin és el cost del programa anterior en funció d'N?

(b) (2 pts.) Siguin $f(n) = \ln\left(\ln(n^2)\right)$ i $g(n) = \ln(\ln n)$. Podem afirmar que $f(n) \in \Theta(g(n))$?

Definim ara

$$F(n) = \begin{cases} n^2, & \text{si } 0 \leq n \leq 10 \\ f(n) & \text{si } n > 10 \end{cases} \qquad G(n) = \begin{cases} n^3, & \text{si } 0 \leq n \leq 10 \\ g(n) & \text{si } n > 10 \end{cases}$$

on f i g són les funcions anteriorment definides. Podem afirmar que $F(n) \in \Theta(G(n))$?

Cognoms **Nom** **DNI**

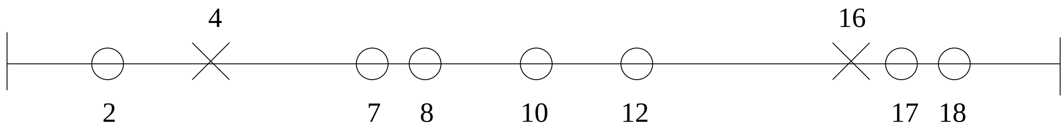
Problema 2 **(6 pts.)**

L'ajuntament d'una gran ciutat decideix comprar estufes de carrer per tal d'escalfar totes les escoles del municipi. Totes les escoles estan situades sobre una mateixa recta, i disposem d'un vector s d'enters amb les distàncies de totes elles al punt quilomètric zero. Sabem que els elements d's són tots diferents.

Els assessors en matèria energètica han decidit ja quantes estufes comprar i en quin punt de la recta situar-les. Disposem d'un vector h que indica el punt quilomètric en la recta de cadascuna de les estufes. Altra vegada, tots els elements d'h són diferents.

Per estalviar energia, podem ajustar les estufes pes tal d'escalfar qualsevol escola que estigui a distància com a molt d. Si volem ajustar totes les estufes de la mateixa manera, quina és la mínima d que ens permet escalfar totes les escoles?

Gràficament, si $s = [8, 17, 2, 12, 18, 7, 10]$ i $h = [16, 4]$ tenim la situació



i podem veure que la solució és $d = 6$. Si prenem $d = 5$, per exemple, l'escola a la posició 10 no seria escalfada per cap estufa.

Per simplificar els raonaments, assumiren en tot aquest problema que $n = s.size() = h.size()$.

(a) (0.5 pts.) La següent funció ens proporciona una solució senzilla al problema:

```
int radius (const vector<int>& h, const vector<int>& s) {
  int rad = 0;
  for (int i = 0; i < s.size(); ++i) {
    int rad_s = inf; // inf és l'int més gran
    for (int j = 0; j < h.size(); ++j)
      rad_s = min(rad_s, abs(h[j] − s[i]));
    rad = max(rad,rad_s);
  }
  return rad;
}
```

En funció d'n, quin és el cost d'una crida a aquesta funció?

(b) (2 pts.) Assumim en aquest apartat que tant h com s estan ordenats de manera creixent. Ompliu els buits de la funció següent perquè resolgui el problema que tenim entre mans:

```
int radius_2 (const vector<int>& h, const vector<int>& s) {
  int r = 0, j = 0;
  for (int i = 0; i < s.size(); ++i){
    while (j < h.size() and h[j] < s[i]) ++j;
    int rad;
    if (j == h.size()) rad = [                    ]
    else if (j == 0)   rad = [                    ]
    else               rad = [                    ]
    r = max(r,rad);
  }
  return r;
}
```

Quin és el cost en cas pitjor d'una crida a *radius_2* en funció d'n?

(c) (3 pts.) Assumim en aquest apartat que h està ordenat de forma creixent. Donada una escola a la posició p, volem trobar la mínima k tal que $p \leq h[k]$ i $0 \leq k < n$. Si no existeix cap k que compleixi aquestes dues condicions (és a dir, si p és més gran que qualsevol element d'h) cal retornar n.

Ompliu els buits de la funció següent perquè retorni aquest valor k en temps $\Theta(\log n)$ en cas pitjor. Solucions que no tinguin aquest cost rebran zero punts.

```
int find (const vector<int>& h, int p) {return find(h, 0, h.size()−1, p);}
```

Cognoms **Nom** **DNI**

```
int find(const vector<int>& h, int l, int r, int p) {
  if (r < l) {

  }
  else {

  }
}
```

Justifica per què la teva solució té cost, en cas pitjor, $\Theta(\log n)$.

(d) (0.5 pts.) Si en la funció *radius_2* de l'apartat (b) reemplacem la línia del **while** per $j = find(h, s[i]);$ quin seria el cost en cas pitjor d'una crida a *radius_2* en funció d'n?

Cognoms **Nom** **DNI**

Examen Parcial EDA Duració: 1h 30min 31/03/2022

- *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 (6 pts.)

Responeu les preguntes següents:

(a) (1.5 pts.) Considereu el codi següent:

```
int n; cin >> n;
vector<int> v(n);
for (int i = 0; i < n; ++i) v[i] = i+1;
random_shuffle(v.begin(), v.end()); // Theta(n)
int s = 0;
for (int i = 0; i < n; ++i)
  for (int j = 0; j < v[i]; ++j) ++s;
cout << s << endl;
```

Recordem que, donat un vector v, la instrucció *random_shuffle(v.begin(),v.end())* reordena els elements del vector v de manera aleatòria en temps lineal en la mida del vector. En funció d'n, què calcula el codi anterior i quin és el seu cost asimptòtic?

(b) (2 pts.) Considereu ara el codi següent:

```
int n; cin >> n;
for (int j = 1; j < n; ++j){
  int k = 2;
  while (k < n) k = k * k;
}
```

En funció d'n, el seu cost és $\Theta(\boxed{})$. Justifiqueu la vostra resposta:

Cognoms **Nom** **DNI**

(c) (2.5 pts.) Per a qualsevol nombre natural $n \geq 1$, definim el vector de naturals següent:

$$(1, 2n, 2, 2n-1, 3, 2n-2, 4, 2n-3, \ldots, n, n+1)$$

En funció d'n, quin és el cost de l'algorisme d'ordenació per inserció sobre aquest vector?

Aquesta cara estaria en blanc intencionadament si no fos per aquesta nota.

Cognoms **Nom** **DNI**

Problema 2 **(4 pts.)**

Donat un vector v d'n naturals diferents i ordenats de forma creixent, volem saber quin és el natural més petit que no apareix a v. Recordem que el 0 és un natural.

(a) (1.5 pts.) Contactem amb un amic que és ben conegut per proporcionar solucions estranyes i innecessàriament complicades, i ens comenta que la funció *inefficient* soluciona aquest problema:

```
bool find (int x, const vector<int>&v, int pos){
  if (pos < 0) return false;
  return v[pos] == x or find(x,v,pos−1);
}

int inefficient (const vector<int>& v) {
  int n = v.size();
  for (int i = 0; i < n; ++i)
      if (not find(i,v,n−1)) return i;
  return n;
}
```

En funció d'n, quin és el cost en cas pitjor d'una crida a la funció *inefficient*?

(b) (2.5 pts.) Doneu vosaltres una funció que solucioni aquest problema i que trigui, en cas pitjor, $\Theta(\log n)$.

```
int efficient (const vector<int>& v, int l, int r) {

}

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

Cognoms **Nom** **DNI**

Examen Parcial EDA   Duració: 1h 30min   08/11/2021

- *L'enunciat té 4 fulls, 8 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 (5 pts.)

Responeu les preguntes següents:

(a) (1 pt.) Considereu la funció *mystery*:

```
bool mystery (int n) {
    if (n ≤ 1) return false;
    if (n == 2) return true;
    if (n%2 == 0) return false;
    for (int i = 3; i*i ≤ n; i += 2)
        if (n%i == 0) return false;
    return true;
}
```

La funció *mystery* determina si ______ i el cost en el cas pitjor, en funció d'n, és $\Theta($ ______ $)$. Justifiqueu aquest cost:

(b) (1 pt.) Considereu ara el codi següent:

```
int j = 0;
int s = 0;
for (int i = 0; i < n; ++i)
  if (i == j*j) {
    for (int k = 0; k < n; ++k) ++s;
    ++j;
  }
```

En funció d'n, el cost és $\Theta($ $\boxed{}$ $)$. Justifiqueu la vostra resposta:

(c) (1 pts.) Donat un vector v d'n enters, volem calcular el nombre total de parelles (i, j) tals que $0 \leq i < j \leq n-1$ i $v[i] = v[j]$. Per tal de solucionar el problema ens donen el codi següent:

```
int pairs (const vector<int>& v, int l, int r) {
  if (l ≥ r) return 0;
  else {
    int m = (l+r)/2;
    int n_left = pairs (v,l,m);
    int n_right = pairs (v,m+1,r);
    int n_crossed = 0;
    for (int i = l; i ≤ m; ++i)
      for (int j = m+1; j ≤ r; ++j)
        if (v[i] == v[j]) ++n_crossed;
    return n_left + n_right + n_crossed;
  }
}
int pairs (const vector<int>& v) {return pairs(v,0,v.size() − 1);}
```

Cognoms **Nom** **DNI**

En funció d'n, el cost d'una crida a $pairs(v)$ és $\Theta($ $)$. Justifiqueu la vostra resposta:

(d) (2 pts.) Ens asseguren ara que tots els nombres de v són naturals estrictament menors que un cert paràmetre enter K, que és constant i no depèn d'n. En aquesta situació, el codi següent és una solució al problema de l'apartat anterior:

```
int pairs_2 (const vector<int>& v) {
  vector<int> times(K,0);
  int n = v.size ();
  for (int i = 0; i < n; ++i) ++times[v[i ]];

  int res = 0;
  for (int i = 0; i < K; ++i) res += (times[ i ]*( times[ i]−1))/2;
  return res ;
}
```

Si n és la mida de v, el cost de $pairs\_2$ en funció d'n és $\Theta($ $)$. Justifiqueu la vostra resposta:

Expliqueu per què el codi anterior és una solució correcta:

Cognoms **Nom** **DNI**

Problema 2 (5 pts.)

Donat un vector v d'n enters ordenats creixentment i un enter x, volem determinar el nombre de vegades que x apareix a v.

(a) (1.5 pts.) Diem que la *primera aparició* d'x dins v és el menor índex i amb $0 \leq i < n$ i $v[i] = x$, o bé -1 si x no apareix a v. Un amic ens comenta que si sabem trobar la primera aparició, aleshores trobar una solució eficient al nostre problema no és massa difícil. Ompliu el codi següent per tal que trobi la primera aparició d'x dins v de manera que el seu cost en cas pitjor sigui $O(\log n)$.

```
int first_occurrence (int x, const vector<int>& v, int l, int r) {
  if (l > r) return −1;
  else {
    int m = (l+r)/2;
    if      (v[m] < x) return first_occurrence (x,v,m+1,r);
    else if (v[m] > x) return first_occurrence (x,v,l,m−1);



  }
}

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

(b) (1.5 pts.) Amb la funció anterior funcionant ja correctament, ens demanen que omplim el codi següent per tal que calculi el nombre d'aparicions d'x dins v:

```
int p = first_occurrence (x,v);

int n = v.size ();
for (int i = 0; i < n; ++i) v[i] = −v[i];
for (int i = 0; i ≤ n/2 − 1; ++i) swap(v[i],v[n−1−i]);
int q = first_occurrence (−x,v);

int res;
if (p == −1) res = 0;
else         res = 

cout << res << endl;
```

Determineu de forma raonada, el cost en cas pitjor del codi anterior en funció d'n.

Expliqueu per què el codi anterior calcula correctament el nombre d'aparicions d'x dins v.

Cognoms **Nom** **DNI**

(c) (2 pts.) Existeix un algorisme per calcular el nombre d'aparicions que sigui, en cas pitjor, asimptòticament més eficient que el codi anterior? Si existeix, expliqueu-lo a alt nivell (no cal codi concret) i analitzeu el seu cost. Si no existeix, expliqueu per què no pot existir.

Aquesta cara estaria en blanc intencionadament si no fos per aquesta nota.

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.