

Solució del problema 1

~~~~~

### 1.1

L'algorisme complet seria:

```
void subintervals_alt(int dif) {
    node_pila* api;
    node_pila* apf;
    if (primer != NULL) {
        api = primer;
        apf = primer; (o també apf = primer -> seg)
        while (apf != NULL) {
            if (api != apf and api->seg != apf) {
                node_pila* aptmp;
                aptmp = api->seg;
                api->seg = aptmp->seg; (o també api->seg = apf)
                delete aptmp;
                --longitud;
            }
            if (apf -> seg == NULL or
                ((apf -> seg -> info - apf->info) > dif)) {
                api = apf -> seg;
            }
            apf = apf -> seg;
        }
    }
}
```

### 1.2

L'algorisme no funciona correctament en el cas que la darrera seqüència de n elements tingui més d'un element, pero no els n. Si hi ha entre 2 i n-2 elements, fallarà en consultar el següent d'un apuntador que és NULL dins del bucle interior en acabar la seqüència de nodes abans d'arribar a n-1. Si hi ha exactament n-1 elements, fallarà en el següent if. L'algorisme tampoc actualitza la longitud de les llistes.

Per tant, la crida amb paràmetre n=2 donarà una pila 2 -> 4 -> NULL amb longitud 5 i una pila 1 -> 3 -> 5 -> NULL amb longitud 0.

La crida amb paràmetre n=3 en aquest cas donarà un error d'execució en arribar al final de la pila durant l'execució del while intern i passar a executar l'if que hi ha després d'aquest. La condició d'aquest implica consultar ap1->seg essent ap1 un apuntador NULL.

L'algorisme correcte seria:

/\* Pre: el p.i. conté una pila, el paràmetre n és un enter >= 2 i p es la pila buida.

Post: el p.i. és una pila on els elements que són el primer de cada n han estat eliminats i la pila p conté el primer element de cada n del p.i.

\*/

```
void un_de_cada_n(Pila &p, int n) {
    if (primer != NULL) {
        node_pila* ap1;
        node_pila* ap2;

        p.primer = primer;
        ap2 = p.primer;

        primer = primer -> seg;
        ap1 = primer;

        int cmpt = 1;
        --longitud;
```

```

        ++p.longitud;
    while (ap1 != NULL) {
        while ((cmpt < (n - 1)) and (ap1 != NULL)) {
            ap1 = ap1->seg;
            ++cmpt;
        }
        if (ap1 == NULL) {
            ap2 -> seg = NULL;
        } else if (ap1 -> seg == NULL) {
            ap2 -> seg = NULL;
            ap1 = ap1 -> seg;
        } else {
            ap2 -> seg = ap1 -> seg;
            ap2 = ap2 -> seg;
            ap1 -> seg = ap2 -> seg;
            ap1 = ap1 -> seg;
            --longitud;
            ++p.longitud;
        }
        cmpt = 1;
    }
}
}
}

```

## Solució del problema 2

~~~~~

2.1

```

// Pre: el paràmetre implícit és buit, v = V i no té elements repetits
// Post: el paràmetre implícit conté l'arbre de cerca de V
void cons_arbre_cerca(vector<T>& v) {
    v.sort();
    primer = i_cons_arbre_cerca(v,0,v.size()-1);
}

```

```

// Pre: (0 <= i) and (j < v.size())
// Post: el valor retornat apunta a una jerarquia de nodes
// que representa l'arbre de cerca corresponent a v[i..j]
static node* i_cons_arbre_cerca(const vector<T>& v, int i, int j) {
    if (i > j) return NULL;
    else {
        int k = (i+j)/2;
        node* n = new node;
        n->info = v[k];
        n->segE = i_cons_arbre_cerca(v,i,k-1);
        n->segD = i_cons_arbre_cerca(v,k+1,j);
        return n;
    }
}

```

2.2

```

/* Pre: El paràmetre implícit és un arbre de cerca d'algun conjunt C */
/* Post: El resultat diu si x pertany a C */
bool cerca(const T& x) {
    return i_cerca(primer,x);
}

```

```

// Pre: m apunta a l'arrel d'una jerarquia de nodes que és l'arbre de cerca d'algun
// conjunt C
// Post: el resultat diu si x pertany a C
static bool i_cerca(node* m, const T& x) {
    if (m == NULL) return false;
    else if (m->info == x) return true;
}

```

```

    else if (m->info > x) return i_cerca(m->segE,s);
    else return i_cerca(m->segD,s);
}

```

2.3

Si la jerarquia de nodes correspon a un arbre de cerca i té mida N , per construcció els dos fills de la jerarquia tenen mida $\leq N/2$, i així per cada node de la jerarquia. Això significa que l'altura de l'arbre és $\log_2 N$.

A cada crida, l'algorisme fa exactament una crida recursiva amb un dels fills, i si l'element no és a l'arbre s'arribarà a un dels fills buits d'una fulla. Per tant, el nombre de crides recursives és com a molt l'altura de l'arbre, que és $\log_2 N$.

A cada crida recursiva amb un arbre no nul es faran dues comparacions de tipus T ($=x$ i $<x$), ja que x no és a l'arbre.

Per tant el nombre de comparacions és $2\log_2 N$.

Errors comuns:

- Dir que l'algorisme recorre tot l'arbre. No ho fa.
- Dir o suposar que per ser arbre binari l'altura és $\log_2 N$.
no, això passa amb arbres que són de cerca, però no necessàriament amb arbres binaris.
- Dir que com que l'algorisme fa "una mena de cerca dicotòmica", es fan $\log_2 N$ comparacions, o bé que a cada comparació es descarten la meitat dels elements. De nou, l'analogia amb la cerca dicotòmica només és certa per la suposició que l'arbre és de cerca, i calia explicar-la com a dalt (les paraules "cerca dicotòmica" no fan màgia).