

Tecnología Digital III: Algoritmos y Estructura de Datos

Clase Practica 6: Ordenamiento

1 Ejercicio 1: Analisis de algoritmos

Decidir que algoritmos vistos en la clase teorica, es conveniente utilizar bajo las siguientes suposiciones. ¿Seria util utilizar una version modificada de dichos algoritmos? **Justificar**

- Ordenar una secuencia ya ordenada.
- insertar k elementos en su posicion en una secuencia v ordenada, con k significativamente mas chico que $v.size()$
- Encontrar los k elementos mas chicos de la secuencia v , con k significativamente mas chico que $v.size()$
- Dadas dos secuencias ordenadas, devolver una secuencia que contenga sus elementos ordenados.
- Ordenar una secuencia que esta ordenada de forma decreciente
- Encontrar los k elementos mas grandes de una secuencia v , con k significativamente mas chico que $v.size()$
- Ordenar una secuencia v en el que sus elementos estan desordenados en a lo sumo k posiciones, con k significativamente mas chico que $v.size()$.

2 Ejercicio 2: Complejidad y Ordenamiento

Se quiere implementar una función

```
vector<int> interseccion_ordenada(const vector<int> & v1, const vector<int> & v2)
```

que toma por parámetro dos vectores de `int`. Se tiene como precondition que `v1` y `v2` están ordenados de forma creciente. Se debe devolver un vector que contenga la intersección ordenada de ambos vectores. Un elemento está en la intersección si está presente en ambos vectores y la cantidad de veces que aparece en la intersección es la mínima entre las dos cantidades de aparición de ese elemento en los vectores.

Ejemplos:

- si `v1 = {1, 2, 2, 4, 5, 9, 9}`, `v2 = {-5, 0, 2, 5, 10}` se debe devolver `{2, 5}`.
- si `v1 = {1, 2, 2, 5, 9, 9}`, `v2 = {-5, 3, 6, 10}` se debe devolver `{}`.
- si `v1 = {0, 0, 5, 8, 8, 8, 11, 20}`, `v2 = {0, 0, 0, 8, 8, 6, 20}` se debe devolver `{0, 0, 8, 8, 20}`.


- (a) Implementar la función `interseccion_ordenada`. El algoritmo debe tener orden de complejidad temporal $O(n + m)$ en peor caso, donde $n = |v1|$, $m = |v2|$. Puede asumir que la operación `push_back`  `vector<int>` es de tiempo constante.

Figure 1: Enter Caption

3 Ejercicio 3: Ordenamiento

Dados `v1` y `v2` de tipo `vector<int>`, podemos compararlos de manera lexicográfica. Diremos que: si `v1[0] < v2[0]` entonces `v1` es menor a `v2`; si `v1[0] > v2[0]` entonces `v1` es mayor a `v2`; si `v1[0] = v2[0]` entonces se debe comparar `v1[1]` y `v2[1]` para decidir, siguiendo estos mismos casos. Notar que si, a su vez `v1[1] = v2[1]` se deberá comparar `v1[2]` y `v2[2]`, y así sucesivamente. Por ejemplo: `{5, 0, 0}` es mayor que `{4, 9, 85}`; `{15, 15, 0}` es menor que `{15, 15, 2}`; `{15, 15, 6}` es igual a `{15, 15, 6}`.

1. Implementar la función `bool es_menor_a(const vector<int> & v1, const vector<int> & v2)`, asumiendo que $|v1| = |v2|$.
2. Implementar la función `void insertion_sort(vector<vector<int>> & vs)`, asumiendo que `vs[i].size() = vs[j].size()` para todo índice `i` y `j` de `vs`. Esta función debe aplicar el algoritmo de ordenamiento **insertion sort** para ordenar el vector `vs` según el orden determinado por la función `es_menor_a`. Tener en cuenta que no se debe ordenar los valores de cada `vector<int>` sino reorganizar las posiciones de cada uno de ellos en el vector “grande” `vs`.

Por ejemplo dado `vs = {{15, 15, 2}, {5, 0, 0}, {4, 9, 85}, {15, 15, 0}}`, `selection_sort(vs)` debe modificar `vs`, y el estado final sería

$$vs = \{\{4, 9, 85\}, \{5, 0, 0\}, \{15, 15, 0\}, \{15, 15, 2\}\}.$$