Algoritmos y Estructuras de Datos I

Primer Cuatrimestre 2020

Guía Práctica 8

Tiempo de Ejecución de Peor Caso de un Programa

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Ejercicio 1.
Contar la cantidad de operaciones elementales que realizan los siguientes programas.

```
int ultimo1(vector<int>& v) {
  return v[v.size() - 1];
}

int ultimo2(vector<int>& v) {
  int i = v.size();
  return v[i - 1];
}

int ultimo3(vector<int>& v) {
  int i = 0;
  while (i < v.size()) {
    i++;
  }
  return v[i - 1];
}
```

Ejercicio 2.
Calcular el tiempo de ejecución de peor caso (en notación O grande) de los siguientes programas con respecto al tamaño de los secuencias de entrada. Recordar que tanto la lectura como la escritura de un elemento en un vector tiene tiempo de ejecución perteneciente a $O(1)$.

```
void f1(vector<int> &vec){
  i = vec.size() / 2;
  while( i >= 0){
    vec[vec.size() / 2 - i] = i;
    vec[vec.size() / 2 + i] = i;
    i--;
  }
}

// pre: |vec| > 20000
void f2(vector<int> &vec){
  i = 0;
  while(i < 10000){
    vec[vec.size() / 2 - i] = i;
    vec[vec.size() / 2 + i] = i;
    i++;
  }
}

// pre: e pertenece a v1
int f3(vector<int> &v1, int e){
  int i = 0;
  while (v1[i] != e){
    i++;
  }
  return i ;
}
```

```
void f4(vector<int> &vec){
  int rec = 0;
  int max_iter = 1000;
  if max_iter > vec.size(){
    max_iter = vec.size();
  }

  for(int i=0; i < max_iter; i++){
    for(int j=0; j < max_iter; j++){
      res += vec[i] * vec[j];
    }
  }
}

void f5(vector<int> &v1, vector<int> &v2){
  vector<int> res(v1.size()+v2.size(),0);
  // inicializa vector en O(|a|+|b|)
  for(int i=0; i < v1.size(); i ++){
    res[i]=v1[i]; // O(1)
  }
  for(int i=0; i < v2.size(); i ++){
    res[v1.size()+i]=v2[i]; // O(1)
  }
  return;
}
```

Ejercicio 3.
Verdadero o falso (justificar).

a) A y B son dos programas que resuelven el mismo problema con un vector v como única entrada. Para un vector de tamaño 100 A demora 2 minutos en ejecutarse y bajo las mismas condiciones (misma entrada, computadora, recursos, etc) B demora 20 segundos. Por lo tanto, el programa B es más eficiente.

b) Si el tiempo de ejecución de peor caso de dos programas pertenece a $O(n)$, entonces el tiempo de cómputo es equivalente (variando un poco según el estado de la computadora en el momento de ejecutar).

c) Se tienen dos programas cuyo tiempo de ejecución de peor caso perteneciente a $O(n)$ y a $O(log(n))$ respectivamente. Para cualquier entrada con tamaño mayor a 100, el segundo programa demorará menos tiempo en ejecutar.

d) Se tienen dos programas con tiempo de ejecución de peor caso perteneciente a $O(n)$ y a $O(log(n))$ respectivamente. Para cualquier entrada con tamaño mayor a un cierto número, el segundo programa demorará menos tiempo en ejecutar.

e) Si se hace un llamado a una función cuyo tiempo de ejecución de peor caso pertenece a $O(n^2)$ dentro de un ciclo, el tiempo de ejecución de peor del ciclo resultante pertenecerá a $O(n^3)$.

Ejercicio 4.

```
int mesetaMasLarga(vector<int> &v) {
  int i = 0;
  int maxMeseta = 0;
  int meseta;
  while (i < v.size()) {
    int j = i + 1;
    while (j < v.size() && v[i] == v[j]) {
      j++;
    }
    meseta = j - i;
    i = j;

    if (meseta > maxMeseta) {
      maxMeseta = meseta;
    }
  }
  return maxMeseta;
}
```

a) ¿Qué hace este programa?

b) Calcular el tiempo de ejecución de peor caso de este programa en función del tamaño del vector.

c) ¿Es posible escribir otro programa que resuelva el problema utilizando solo un ciclo?

Ejercicio 5.

```
vector<int> hacerAlgo(vector<int> &v) {
  vector<int> res;
  for (int i = 0; i < 100; i++) {
    res.push_back(contarApariciones(v, i+1));
  }

  return res; // copia el vector
}

int contarApariciones(vector<int> &v, int elem) {
  int cantAp = 0;
  for (int i = 0; i < v.size(); i++) {
  if (v[i] == elem) {
      cantAp++;
```

```
    }
  }
}
```

Calcular el tiempo de ejecución de peor caso del programa `hacerAlgo` con respecto a $|v|$.

Ejercicio 6.
Dadas las siguientes funciones:

```
bool f(vector<float> s) {
    float p;
    float res = 0;
    for (int i = 0; i < s.size(); i++) {
        p = g(h(s, i)));
        if (p > res) {
            res = p;
        }
    }
    return res;
}

float g(vector<float> v) {
    float p = 1;
    for (int i = 0; i < v.size(); i++) {
        p = p * v[i];
    }
    return p;
}
```

```
// Pre: n <= v.size()
vector<float> h(vector<float> w, int n) {
    vector<float> res;
    int i = 0;
    while (i < n && w[i] > 0) {
        res.push_back(w[i]);
        i++;
    }
    return res;
}
```

1. Explicar brevemente con sus palabras qué cómputo realiza la función f. Hacerlo en a lo sumo un párrafo acompañado de un ejemplo (no explicar cómo lo hace).
2. Indicar el tiempo de ejecución de peor caso de h en función del n. Justificar.
3. Dar un ejemplo del peor caso y un ejemplo de mejor caso para la función h. Justificar.
4. Indicar el tiempo de ejecución de peor caso de f en función del tamaño de s. Justificar.

Ejercicio 7.

```
int sumarPotenciaHasta(int n) {
  int res = 0;
  int i = 1;
  while(i < n) {
    res = res + i;
    i = i * 2;
  }
  return res;
}
```

a) ¿Qué tiempo de ejecución de peor caso tiene el programa en función del valor n?

b) ¿Es posible calcular la suma de potencias hasta n con un tiempo de ejecución de peor caso asintóticamente mejor que el programa del punto anterior si la pre condición es True?

c) ¿Es posible calcular la suma de potencias hasta n con un tiempo de ejecución de peor caso asintóticamente mejor que el item a si la pre condición es $(\exists x : \mathbb{Z})\ n = 2^x + 1$?

Ejercicio 8.
Una matriz cuadrada se dice *triangular* si todos los elementos por debajo de la diagonal son iguales a cero.

a) Escribir un programa que calcule el determinante de una matriz triangular. Recordar que el determinante de una matriz triangular es el producto de los elementos de su diagonal.

b) Escribir un programa que determine si una matriz de $N \times N$ es o no triangular.

c) Calcular el tiempo de ejecución de peor caso de los programas:

   a) en función de la cantidad de elementos de la matriz.

   b) en función de la cantidad de *filas* de la matriz.

Ejercicio 9.
Dadas dos matrices A y B se desea multiplicarlas siguiendo esta especificación:

`proc multiplicar` (in m1: $seq\langle seq\langle \mathbb{Z} \rangle\rangle$, in m2: $seq\langle seq\langle \mathbb{Z} \rangle\rangle$, out res: $seq\langle seq\langle \mathbb{Z} \rangle\rangle$) {

`Pre` {*completar*}

`Post` $\{|res| = |m1| \wedge_L (\forall i : \mathbb{Z})(0 \leq i < |m1| \rightarrow_L (|res[i]| = |m2[0]| \wedge_L (\forall j : \mathbb{Z})(0 \leq j < |m2[0]| \rightarrow_L res[i][j] = \sum_{k=0}^{|m2|-1} m1[i][k] * m2[k][j])))\}$

}

a) Completar la precondición del problema.

b) Escribir un programa que retorne AB

c) Determinar el tiempo de ejecución de peor caso de este programa en función de:

   a) La cantidad de filas y columnas de cada una de las matrices.

   b) Suponiendo que $N = \text{filas}_{m1} = \text{filas}_{m2} = \text{columnas}_{m1} == \text{columnas}_{m2}$

Ejercicio 10.
Sea *mat* una matriz de $N \times N$ y la siguiente función:

```
void sumasAcumuladas(vector<vector<int>> &mat, int N){
  for(int i = N - 1; i >= 0, i--) {
    for(int j = N - 1, j >= 0; j--) {
      mat[i][j] = sumHasta(mat, i, j, N);
    }
  }
}

int sumHasta(vector<vector<int> &mat, int I, int J, int N) {
  int res = 0;
  int lim;
  for (int i = 0; i <= I; i++) {
    if (i == I) {
      lim = J
    } else {
      lim = N - 1;
    }
    for (int j <= lim; j++) {
      res += mat[i][j]
    }
  }
  return res;
}
```

1. Calcular el tiempo de ejecución de peor caso del programa `sumasAcumuladas` en función de la cantidad de elementos de la matriz.
2. Dar un programa que resuelva sumasAcumuladas cuyo tiempo de ejecución en peor caso sea $O(n^2)$

Ejercicio 11. Dada una secuencia de n números enteros, dar un programa que encuentre la máxima cantidad de elementos impares consecutivos cuya tiempo de ejecución de peor caso pertenezca a $O(n)$.

Ejercicio 12. Escribir un programa que sea correcto respecto de la siguiente especificación, y cuyo tiempo de ejecución de peor caso pertenezca a $O(|s|)$.

```
proc restarAcumulado (in s: seq⟨ℤ⟩, in x: ℤ, out res: seq⟨ℤ⟩)  {
      Pre {True}
      Post {|res| = |s| ∧_L (∀i : ℤ)(0 ≤ i < |s| →_L res[i] = x − Σ_{j=0}^{i} s[j])}
}
```

Ejercicio 13. Sea m una matriz de $N \times N$ donde cada posición contiene el costo de recorrer dicha posición (dicho costo es positivo para todas las posiciones). Asumiendo que la única forma de recorrer la matriz es avanzando hacia abajo y hacia la derecha, escribir un programa que calcule el mínimo costo para llegar desde la posición $(1, 1)$ hasta la posición (N, N) y cuyo tiempo de ejecución de peor caso pertenezca a $O(N^2)$.

Ejercicio 14.
Sea A una matriz cuadrada y n un número natural.

a) Escribir un programa que calcule $\prod_{i=1}^{n} A$ (es decir A^n) (reutilizar el programa del ejercicio **??**) ¿Cuál es el tiempo de ejecución de peor caso de esta función?

b) Resolver el punto anterior suponiendo que $n = 2^m$ (n potencia de 2) ¿Se pueden reutilizar cuentas ya realizadas? ¿Cuál es el tiempo de ejecución de peor caso para cada programa?

Ejercicio 15.
Dada una matriz de booleanos de n filas y m columnas con n impar. Se sabe que hay exactamente una fila que no está repetida, y el resto se encuentra exactamente dos veces en la matriz.

a) Escribir un programa que devuelva un vector con los valores de la fila que no se repite. ¿Cuál es su tiempo de ejecución de peor caso descripto ?

b) ¿Es posible un programa que recorra cada casillero de la matriz sólo una vez? En caso de no haberlo resuelto con esta restricción, modificar el programa para que la cumpla. ¿Cuál es su tiempo de ejecución de peor caso ?

c) La solución al punto anterior, ¿utiliza vectores auxiliares?, en caso que lo haga, escribir un programa que no los necesite.

RESOLUCIONES.

Ejercicio 1 .

```
1  int ultimo1(vector<int> &v){
2     return v[v.size()-1];      //3
3  }                             //t(n)=3 -> O(1)
4
5  int ultimo2(vector<int> &v){
6     int i=v.size();            //2
7     return v[i-1];             //3
8  }                             //t(n)=5 -> O(1)
9
10 int ultimo3(vector<int> &v){  //v.size()=n
11    int i=0;                 //1
12    while(i<v.size()){       //3, n iteraciones
13       i++;                  //1
14    }                        //t(n)= 3+4n
15    return v[i-1];           //3
16 }                           //t(n)= 4n +7 -> O(n)
```

.

Ejercicio 2 .

```
1  void f1(vector<int> &vec){    //v.size()=n
2     i=vec.size()/2;            //3
3     while(i>=0){               //2, n/2 iteraciones
4        vec[vec.size()/2-i]=i;     //6
5        vec[vec.size()/2+i]=i;     //6
6        i--;                       //1
7     }                          //t(n)=2+ 7n
8  }                             //t(n)=5+7n -> O(n)
9
10 // pre:  |vec|>20000
11 void f2(vector<int> &vec){
12    i=0;                       //3
13    while(i<10000){            //2, n/2 iteraciones
14       vec[vec.size()/2-i]=i;     //6
15       vec[vec.size()/2+i]=i;     //6
16       i++;                       //1
17    }                          //t(n)=2+7n
18 }                             //t(n)=5+7n -> O(n)
19
20 // pre: e pertenece a v1
21 int f3(vector<int> &v1, int e){
22    int i = 0;              //1
23    while (v1[i] != e){     //4, n iteraciones
24       i++;                    //1
25    }                       //t(n)=4+5n
26    return i ;              //1
27 }                          //t(n)=6+5n -> O(n)
28
29 void f4(vector<int> &vec){
30    int rec = 0;               //1
31    int max_iter = 1000;       //1
32    if (max_iter > vec.size()){   //3
33       max_iter = vec.size();  //3
```

```
34    }
35
36    for(int i=0; i < max_iter; i++){ //init: 1, guarda: 3, incremento: 1, max_iter iteraciones
37       for(int j=0; j < max_iter; j++){ //init: 1, guarda: 3, incremento: 1, max_iter iteraciones
38          res += vec[i] * vec[j];          //7
39       }                                 //t(n)=4+11n
40    }                                  //t(n)=4+(4+4+11n)n
41 }                             //t(n)=12+8n+11n^2 -> O(n^2)
42
43 void f5(vector<int> &v1, vector<int> &v2){//|v1|=n |v2|=m
44    vector<int> res(v1.size()+v2.size(),0);   //4
45    // inicializa vector en O(|a|+|b|)
46    for(int i=0; i < v1.size(); i ++){     //init: 1, guarda: 3, incremento: 1, n iteraciones
47       res[i]=v1[i];                     // 4
48    }                               //t(n)=4+8n
49    for(int i=0; i < v2.size(); i ++){     //init: 1, guarda: 3, incremento: 1, m iteraciones
50       res[v1.size()+i]=v2[i];           // 7
51    }                             //t(n)=4+11n
52    return;                          //0
53 }                        //t(n)=4+4+4+0+8n+11n -> O(n)
```

.

Ejercicio 3 .

a) Las generalizaciones son la muerte, más eficiente en relación a qué?. Si suponemos que los vectores de entrada rondan alrededor del tamaño donde se realizo la comparación entonces sin lugar a dudas A es más eficiente, sin embargo no tengo ese dato, y como no se como se comportan los algoritmos en el limite del tamaño del vector tendiendo a infinito no puedo asegurar que uno sea mejor que otro, por lo tanto esa afirmación es falsa.

b) Esta afirmación es falsa, ambos algoritmos difieren en catidad de operaciones por una constante, por lo tanto su tiempo de ejecución no tiene por que ser el mismo, lo que si se puede afirmar es que a medida que n se haga cada vez más grande sus tiempos de ejecución se van a empezar a asemejar.

c) Por intuición y porque como ya dije antes, las generalizaciones son la muerte, se que no puedo decir que simplemente para todo programa con complejidad $O(log(n))$, si $n > 100$ entonces se va a ejecutar más rapido que otro de complejidad $O(n)$, que clase de numero divino es el 100 como para poder afirmar semejante guasada. Un contraejemplo pavo seria el siguiente:

- $t(n) = n \rightarrow O(n)$
- $t(n) = 101 + log(n) \rightarrow O(log(n))$

d) Es intuitivo, pero vamos a demostrarlo, para eso hay que demostrar que $log(n) \leq n$, vamos a hacerlo usando inducción. Reescribamos la expresión para que quede mas manejable.

$$\begin{aligned} log(n) &\leq n \\ e^{log(n)} &\leq e^n \\ n &\leq e^n \end{aligned}$$

Mi hipotesis inductiva es :

HI: $n \leq e^n$

$$
\begin{aligned}
\text{quiero ver que} \quad & n+1 \leq e^{n+1} \\
\text{por HI} \quad & n \leq e^n \\
\leftrightarrow\ & n+1 \leq e^n + 1 \\
\text{por transitividad} \quad & e^n + 1 \leq e^{n+1} \\
& \frac{e^n+1}{e^n} \leq \frac{e^{n+1}}{e^n} \\
\text{nueva HI} \quad & \frac{1}{e^n} \leq e-1 \\
\text{qvq} \quad & \frac{1}{e^{n+1}} \leq e-1 \\
\text{por HI} \quad & \frac{1}{e^n} \leq e-1 \\
\leftrightarrow\ & \frac{1}{e^n}\frac{1}{e} \leq \frac{e-1}{e} \\
\leftrightarrow\ & \frac{1}{e^{n+1}} \leq 1 - \frac{1}{e} \\
\text{por transitividad} \quad & 1 - \frac{1}{e} \leq e-1 \\
\leftrightarrow\ & 2 \leq e + \frac{1}{e}
\end{aligned}
$$

□

e) Falso, contraejemplo: el ciclo de afuera no tiene porque necesariamente depender de n.

Ejercicio 4.

a) Devuelve la posición del elemento que tenga mas elementos repetidos a su izquierda.

b) .

```
1  int mesetaMasLarga(vector<int> &v) {   //|v|=n
2     int i = 0;              //1
3     int maxMeseta = 0;      //1
4     int meseta;             //1
5     while (i < v.size()) {  //3, n iteraciones
6        int j = i + 1;          //3
7        while (j < v.size() && v[i] == v[j]) { //9, n-i-1 iteraciones
8           j++;                 //1
9        }                       //t(n)= 9+10(n-i-1)
10       meseta = j - i;         //3
11       i = j;                  //1
12       if (meseta > maxMeseta) {  //3
13          maxMeseta = meseta;        //1
14       }
15    }              //t(n)=3+ 8n+ suma(i=0,n-1)(9+10(n-i-1))
16    return maxMeseta;       //1
17 }              //t(n)=7+17n+10n^2 -n -n(n-1)/2 -> O(n^2)
```

.

c) no se me ocurre.

Ejercicio 5 .

```
1  vector<int> hacerAlgo(vector<int> &v) {   //|v|=n
2     vector<int> res;  //1
3     for (int i = 0; i < 100; i++) {     //init: 1, guarda: 2, incremento: 1, 100 iteraciones
4        res.push_back(contarApariciones(v, i+1)); //4+5+9n
5     }     //t(n)=4+(3+9+9n)*100
6     return res; // copia el vector
7  }     //t(n)=125+900n    -> O(n)
8
9  int contarApariciones(vector<int> &v, int elem) {
10    int cantAp = 0;      //1
11    for (int i = 0; i < v.size(); i++) {//init: 1, guarda: 3, incremento: 1, n iteraciones
12       if (v[i] == elem) {  //4
13          cantAp++;      //1
14       }
15    }        //t(n)=4+ (4+5)n
16 }     //t(n)=5+9n ->O(n)
```

.

Ejercicio 6.

a) Basicamente lo que hace es devolver falso si todos los elementos del vector son menores o iguales a cero. EJ $f([-1, 0, -3, -5, 0]) = false$

b) .

```
1  bool f(vector<float> s) {
2     float p;          //1
3     float res = 0;    //1
4     for (int i = 0; i < s.size(); i++) {   //init: 1, guarda: 3, incremento: 1, n iteraciones
5        p = g(h(s, i)));     //1+(10+11w)(6+8i)???????
6        if (p > res) {       //3
7           res = p;          //2
8        }
9     }                 //t(n)= 4+
10    return res;       //1
11 }                    //t(n)=
12
13 float g(vector<float> v) {
14    float p = 1;      //1
15    for (int i = 0; i < v.size(); i++) {   //init: 1, guarda: 3, incremento: 1, n iteraciones
16       p = p * v[i];        //4
17    }                 //t(n)=4+8n
18    return p;         //1
19 }                    //t(n)=6+8n -> O(n)
20
21
22 //Item 2:
23 // Pre: n <= v.size()
24 vector<float> h(vector<float> w, int n) {
25    vector<float> res;      //1
26    int i = 0;              //1
27    while (i < n && w[i] > 0) {   //7, n iteraciones
28       res.push_back(w[i]);    //3
29       i++;                    //1
30    }                     //7+11n
31    return res;           //1
```

```
32  }                          //t(n)=10 + 11n   -> O(n)
```

.

c)
- Mejor caso: $f([-1, 0, -3, -5, 0])$
- Peor caso: $f([1, 2, 3, 4, 5, 6, 7])$

Ejercicio 7 .

a) .

```
1  int sumarPotenciaHasta(int n) {
2     int res = 0;      //1
3     int i = 1;        //1
4     while(i < n) {    //3, log(n) iteraciones
5        res = res + i;    //3
6        i = i * 2;        //3
7     }                 //t(n)=3+9log(n)
8     return res;       //1
9  }                    //t(n)= 6+9log(n)  -> O(log(n))
```

.

b) Por Algebra I se que $\sum_{i=0}^{x} 2^i = 2^{x+1} - 1$, entonces busco el primer x tq $2^x >= n$, y devuelvo la $\sum_{i=0}^{x-2} 2^i$, que seria igual a $2^{x-1} - 1$

```
1  int sumarPotenciaHasta(int n) {
2     int res = 0;      //1
3     int x = 0;        //1
4     while(pow(2,x) < n) {      //3, log(n) iteraciones    //3
5        x++;        //1
6     }                 //t(n)=3+4log(n)
7     return pow(2,x-1)-1;       //1
8  }                    //t(n)= 6+4log(n)  -> O(log(n))
```

.

c) Se que $(\exists x : \mathbb{Z})\ n = 2^x + 1$, entonces puedo despejarlo.

$$
\begin{aligned}
n &= 2^x + 1 \\
log(n-1) &= log(2^x) \\
log(n-1) &= xlog(2) \\
\frac{log(n-1)}{log(2)} &= x \\
log(n-1) - log(2) &= x
\end{aligned}
$$

```
1  int sumarPotenciaHasta(int n) {
2     int res = 0;      //1
3     int x = log(n-1)-log(2);         //O(1)
4     res=pow(2,x-1)-1;             //O(1)
5     return res;       //1
6  }                    //t(n)= 1+O(1)+O(1)  -> O(1)
```

.