



Instituto Politécnico Nacional
Escuela Superior de Cómputo



Practica 2

Integrantes:

Escutia López Arturo

López Santiago Daniel

4/Sept/2016

Unidad de aprendizaje: Análisis de algoritmos

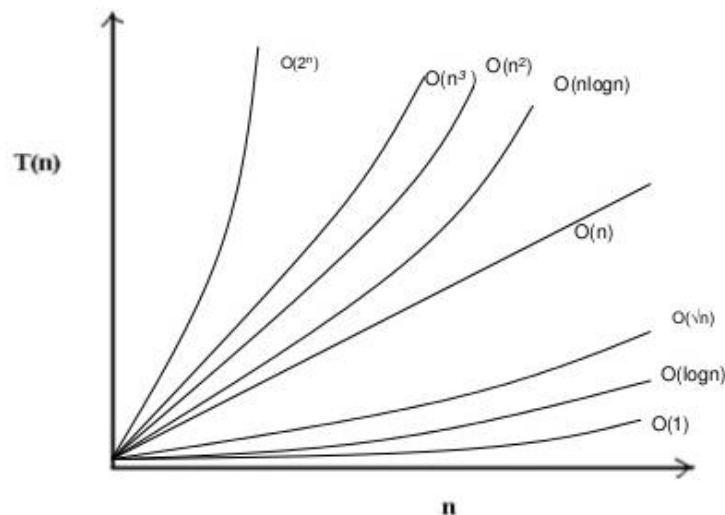
Profesor: Díaz Santiago Sandra

Teoría

La notación asintótica son funciones que nos permiten saber sobre las complejidades en los algoritmos para valores n muy grandes, ya que para valores suficientemente pequeños de n , el coste de ejecución de cualquier algoritmo es pequeño, incluyendo los algoritmos ineficientes, es por eso que para poder comparar la eficiencia entre diferentes algoritmos se considera el comportamiento de sus funciones de complejidad con valores de entradas muy grandes que sería en los peores casos de ejecución el número de operaciones.

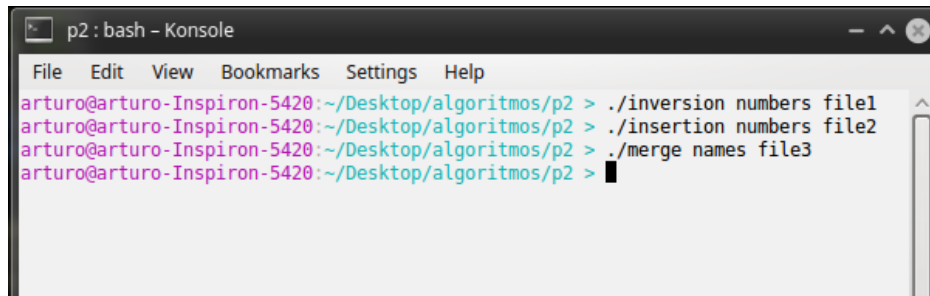
Cuando se llevan a cabo el cálculo de los costos de operaciones a veces es un poco difícil determinar el número de instrucciones que le toma a una computadora de manera exacta realizar una sentencia y esta solamente se le denomina como una constante c , es por esto que se desarrolló ésta notación tan conveniente ya que la notación de la gran O nos permite leer por ejemplo de la forma $O(n)$ (de orden n) en vez Cn .

En el caso de $f(n)$ pertenece $O(g(n))$ cuando $f(n) \leq cg(n)$ donde $g(n)$ es una cota superior, es decir, que a partir de un $n > n_0$ donde n_0 pertenece a los números naturales $f(n)$ está siempre por debajo de $g(n)$.



Funcionamiento

A continuación se muestra la ejecución de los programas, el primer parámetro de la ejecución es el nombre de donde se leerán los archivos que contendrán los datos y el segundo parámetro es para nombrar el archivo de salida que contendrá los resultados arrojados por la ejecución.



```
p2 : bash - Konsole
File Edit View Bookmarks Settings Help
arturo@arturo-Inspiron-5420:~/Desktop/algoritmos/p2 > ./inversion numbers file1
arturo@arturo-Inspiron-5420:~/Desktop/algoritmos/p2 > ./insertion numbers file2
arturo@arturo-Inspiron-5420:~/Desktop/algoritmos/p2 > ./merge names file3
arturo@arturo-Inspiron-5420:~/Desktop/algoritmos/p2 > █
```

		1	6
		2	3
		3	5
1	jose	4	7
2	arturo	5	22
3	miguel	6	4
4	daniel	7	6
5	sandra	8	8
6	pepe	9	9
7	javier	10	3
8	alberto	11	2
9	jimena	12	5
10	jorge	13	6
		14	4

1.1 Insertion_sort

En la función insertion recibe como parámetro la lista de los números de forma desordenada, es una forma de ordenamiento muy básica donde se inicia desde el primer elemento hasta el final del arreglo. El for nos permite barrer todo el arreglo y el while lo que nos permite es a partir del elemento actual irlo comparando con los valores anteriores a él, de esta forma se inserta el valor en su lugar correspondiente ya que el while termina ya sea cuando el valor este en la posición inicial dado que sea el elemento más pequeño del arreglo o mientras siga siendo menor al valor con el que se le está comparando recorriendo los demás valores mayores un lugar hacia la derecha.

```

#include <bits/stdc++.h>
using namespace std;

void insertion(vector<int> nums, char * file){
    int aux=0, j, i;
    ofstream myfile;
    for(i=1; i<nums.size(); i++){
        aux=nums[i];
        j=i-1;
        while ( j>=0 && nums[j]>aux){
            nums[j+1]=nums[j];
            j=j-1;
        }
        nums[j+1]=aux;
    }
    i=0;
    myfile.open (file);
    while(i<nums.size()){
        myfile<<nums[i]<< "\n";
        i=i+1;
    }
    myfile.close();
}

int main(int argc, char *c[]){
    ifstream f(c[1]);
    vector <int> nums;
    string n;
    int i;
    if(f.is_open()){
        while(getline(f, n))
            nums.push_back(atoi(n.c_str()));
        f.close();
    }

    insertion(nums, c[2]);
    return 0;
}

```

1.2 Inversión

Este algoritmo parte de la misma idea del insertion sort, ir recorriendo el arreglo de inicio a fin con un ciclo for , a partir de la posición actual, comparar el valor en la posición j con todos los valores anteriores a éste hasta llegar al inicio del arreglo, de tal forma que i es una posición < a j pero si $A[i] > A[j]$ se incremente el contador de las inversiones y se guarde el par de (i,j) en el file. Este algoritmo tiene una complejidad de $O(n^2)$ debido a que el while que n operaciones, pero este while es realizado n veces debido a que está dentro del for que recorre todo el arreglo, existe un algoritmo con complejidad menor a n^2 aplicando el uso de mergesort $O(n \log n)$.

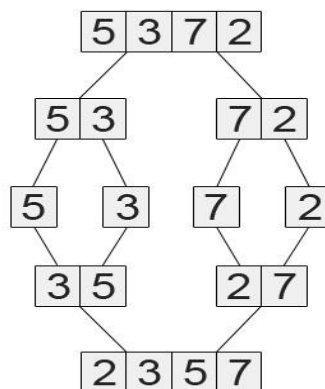
```

8  #include <bits/stdc++.h>
9  using namespace std;
10
11 void inversions(vector<int> nums,char *file){
12     int j,count=0;
13     ofstream myfile;
14     myfile.open (file);
15
16     for(int i=1 ;i<nums.size();i++){
17         j=i-1;
18         while(j>=0){
19             if(nums[j]>nums[i]){
20                 count+=1;
21                 myfile<<"("<j<<","<i<<")\n";
22             }
23             j=j-1;
24         }
25     }
26     myfile<<count<<" inversions";
27     myfile.close();
28 }
29
30
31 int main(int argc,char *c[]){
32     ifstream f(c[1]);
33     vector <int> nums;
34     string n;
35     int i=0;
36     if(f.is_open()){
37         while(getline(f,n))
38             nums.push_back(atoi(n.c_str()));
39         f.close();
40     }
41
42     inversions(nums,c[2]);
43     return 0;
44 }
45
34 (0,11)
35 (8,12)
36 (7,12)
37 (4,12)
38 (3,12)
39 (12,13)
40 (11,13)
41 (8,13)
42 (7,13)
43 (6,13)
44 (4,13)
45 (3,13)
46 (2,13)
47 (0,13)
48 47 inversions

```

1.3 MergeSort

En la función Merge lo que se hace es dividir por la mitad la lista cada vez más en subarreglos hasta que ya no sea posible dividirla más y contenga un solo elemento de la primera y la segunda mitad del arreglo/subarreglo , después manda a llamar merge lo cual lo que hace es comparar el elemento inicial entre los dos arreglos, dependiendo de cuál de éstos dos sea el elemento menor se inserta en uno nuevo y se avanza una posición, en caso de que uno de los dos arreglo ya se haya vaciado y el otro aun contenga elementos solo se procede a insertar los elementos uno tras otro del arreglo faltante.



```

22     }
23     mergeSort(0,v.size() - 1);
24     for(int i=0;i<v.size();i++) output << v[i] << " \n";
25     output << "\n";
26     return 0;
27 }
28
29 void mergeSort(int lower, int upper) {
30     if(lower < upper) {
31         int middle = (lower + upper) / 2;
32         mergeSort(lower,middle);
33         mergeSort(middle+1,upper);
34         merge(lower,middle,upper);
35     }
36 }
37
38 void merge(int lower, int middle ,int upper) {
39     queue <string> q1;
40     queue <string> q2;
41     for(int i=lower;i<=middle;i++) q1.push(v[i]);
42     for(int i=middle+1;i<=upper;i++) q2.push(v[i]);
43     int idx = lower;
44     while(!q1.empty() && !q2.empty()) {
45         if(q1.front() <= q2.front()) {
46             v[idx++] = q1.front();
47             q1.pop();
48         } else {
49             v[idx++] = q2.front();
50             q2.pop();
51         }
52     }
53     while(!q1.empty()) {
54         v[idx++] = q1.front();
55         q1.pop();
56     }
57     while(!q2.empty()) {
58         v[idx++] = q2.front();
59         q2.pop();
60     }
61 }
62

```

```

1  alberto
2  arturo
3  daniel
4  javier
5  jimena
6  jorge
7  jose
8  miguel
9  pepe
10 sandra
11
12


```

$i < j$ $A[i] > A[j]$ $A[j] > A[i]$ (i, j)
 i, j
 Arturo Escutia
 31-05-2016
 giaspwr.

a) $A = \{2, 3, 8, 6, 1\}$ $(0, 4)$ $(2, 4)$
 $(1, 4)$ $(3, 4)$
 $(2, 3)$

b) descendemente se obtiene el max # de inv.
 $(1, 2), (1, 3), (1, 4), \dots, (1, n-1), (1, n)$ $n-1$
 $(2, 3), (2, 4), \dots, (2, n-1), (2, n)$ $n-2$
 \vdots $n-3$
 $(n-1, n)$ 1
 $\frac{n(n-1)}{2}$

c)
 Count (A as array)
 for $i = 1$ to $n-1$
 $j \leftarrow i-1$
 While $j \geq 0$
 if $A[j] > A[i]$ then
 count \leftarrow count + 1
 end if
 $j \leftarrow j-1$
 end while
end for
return count

Arturo Escutia López
 31/Ago/2016


Referencias

- <http://eafranco.com/docencia/analisisdealgoritmos/files/05/Diapositivas05.pdf>
- <https://ccc.inaoep.mx/~jagonzalez/ADA/NotacionA.pdf>
- <http://www.cs.cornell.edu/courses/cs312/2004fa/lectures/lecture16.htm>