Algorithms and complexity

using python.

Iván Andrés Trujilllo Abella

Facultad de Ingenieria
Pontificia Universidad Javeriana

trujilloiv@javeriana.edu.co

algorithms and complexity

The time of execution of a program could be important when large input data.

Sorting algorithms

The problem of sort an unsorted **list** is a recurrent problem to study the complexity and the efficient in the way how we can solve a problem.

Bubble sort algorithm

this strategy consist in compare the adjacent values and replace the adjacent only if is grater than the previous one.

```
lista = [a,b,c,d,e]
```

Assume that $a < b < c < d < e$
then the first iteration we have

```
[b,c,d,e,a]
```

translate the first element for all list until get the last position, in the worst case.
the following list must be iterate over n-1.

Bubble sort algorithm

Python

```
data = [99,98,97,96,95]
iter = len(data)
while iter >0: # or for i in range(0,iter-1):
  for i in range(0,iter-1):
    if data[i]> data[i+1]:
      data[i+1], data[i] = data[i], data[i+1]
  iter = iter -1
print(data)
```

Note that there are only needed $n-1$ iterations due to $n-1$ iteration the first element is the lesser value of the n values.

Preliminar 1

Remember that we can concatenate two list

```
listaOne=[1,2,3]
listaTwo=[4,5,6]
listaThree = listaOne+ListaTwo
print(listaThree)
```

will return:

```
[1,2,3,4,5,6]
```

Divide and conquer

This is a strategy to resolve different kind of problems, among them the reduction of iterations.

Quick Sort

We can think in the following method:

```
lista = [a,b,c,d,e,f]
```

lista contain random numbers, take the first of the last and called pivot, and try to sorted to the

Insights

for code implementation

The logic behind this is two positional index, beginning in 0-index, one index to keep position and the other to iterate over the data, the idea is keep the values greater than pivot with position index, and replace by first lesser value in the index iterator, after make the first swap then we need increment the positional index for the next more greater value than pivot. for this reason we can interchange the last positional index with the pivot value, due it is the first value greater than pivot.

Recursive instance

Note that we can broke up the list.

Big O notation

the big o notation represent the maximun bonduarie of the real time of algorithm.

$$g(n) * c \geq t(n) \tag{1}$$

could be exist a number n_0 in which satisfy the last equation.

Complexity in time

we can count the number of process? we can measure the time of execution?

Big O principles

$$k * n + u \tag{2}$$

we said that is $O(n)$ note that we only take the major term.


IHS
Pontificia Universidad
JAVERIANA
Bogotá

Big O in recursion

Primitive operations

Are consider primitive operations:

- Assign variables
- Comparation
- other statements..
- conditionals

Best-average-worse case

```
for x in range(0,n):
    print(x)
    print(x**2)
```

Note here that the complexity is $2n$

```
for x in range(0,n):
    if x%2==0:
        print(x)
        print(x**2)
```

In the best case all numbers are odd numbers, therefore the complexity will be n only will be executed the *if* statement. The worst case when list a full of even numbers the complexity will be $3n$, and the average will be then $2n$.