Universidad de Concepción
Facultad de Ingeniería
Departamento de Ing. Informática y Cs
de la Computación

Profesor
Diego Seco

Laboratorio 2:

Estructuras de Datos (503220)

Estudiante: Nicolás Araya (2018448613)

1.

```
#ifndef LIST_H
#define LIST_H

class List {
    public:
        virtual void insert(int) = 0;
        virtual void pop() = 0;
        virtual int at(int) = 0;
        virtual int size() = 0;
};

#endif
```

2.

ArrayList.h:

```
#include "List.h"

class ArrayList : public List {
    private:
        int mysize, capacity;
        int *myarray;
        void agrandar();
    public:
        ArrayList();
        ~ArrayList();
        void insert(int n);
        void pop();
        int at(int pos);
        int size();
        void print();
};
```

ArrayList.cpp

```cpp
#include "ArrayList.h"
#include <iostream>

using namespace std;

ArrayList::ArrayList(){
    capacity=5000;
    mysize=0;
    myarray = new int(capacity);
}
ArrayList::~ArrayList(){
    delete[] myarray;
    mysize=0;
}
void ArrayList::insert(int n){   //O(n)
    mysize++;
    if(capacity<mysize){
        cout<<"Capacidad Excedida"<<endl;
        agrandar();
        cout<<"extendido"<<endl;
    }
    int aux = myarray[0];
    for(int i = 1; i<mysize; i++){   //O(n)
        int aux2 = myarray[i];
        myarray[i]=aux;
        aux=aux2;
    }
    myarray[0]=n;
}
```

```cpp
void ArrayList::pop(){          //O(1)
    myarray[mysize-1]=0;
    mysize--;

}
int ArrayList::at(int pos){    //O(1)
    return myarray[pos-1];
}
int ArrayList::size(){
    return mysize;
}
void ArrayList::agrandar(){          //O(n)
    int * aux = myarray;
    myarray = new int(capacity*2);
    for(int i=0; i<mysize; i++){          //O(n)
        myarray[i]=aux[i];
    }
    capacity=capacity*2;
    free(aux);
}

void ArrayList::print(){
    for(int i=0; i<mysize; i++){
        cout<<myarray[i]<<" ";
    }
}
```

3. LinkedList.h:

```cpp
#include "List.h"

struct nodo {
    int n;
    nodo *siguiente;
};

class LinkedList: public List {
    private:
        struct nodo *head;
        int mysize;
    public:
        LinkedList();
        ~LinkedList();
        void insert(int n);
        void pop();
        int at(int pos);
        int size();
        void print();
};
```

LinkedList.cpp:

```cpp
using namespace std;

LinkedList::LinkedList(){
    mysize = 0;
    head = new nodo();
}
LinkedList::~LinkedList(){
    struct nodo *aux;
    for(int i=0; i<mysize; i++){
        aux=head->siguiente;
        delete head->siguiente;
        head->siguiente=aux->siguiente;
    }
    mysize=0;
    delete head;
}
void LinkedList::insert(int x){      //O(1)
    struct nodo *aux;
    aux = new nodo();
    if(mysize>0){
        aux->n = x;
        aux->siguiente = head->siguiente;
        head->siguiente = aux;
        mysize+=1;
    }
    if(mysize==0){
        aux->n = x;
        head->siguiente = aux;
        mysize+=1;
    }
}
```

```cpp
void LinkedList::pop(){            //O(n)
    struct nodo *aux;
    aux = head;
    if(mysize==1){
        delete head->siguiente;
    }
    for(int i = 1; i<=mysize-1; i++){
        aux=aux->siguiente;
        if(i==mysize-1){
            delete aux->siguiente;
            break;
        }
    }
    mysize-=1;

}
int LinkedList::at(int pos){         //O(n)
    if(pos>mysize){
        return -1;
    }
    struct nodo * aux;
    aux = head->siguiente;
    int i=1;
    while(i<pos){
        aux=aux->siguiente;
        i++;
    }
    return aux->n;
}
int LinkedList::size(){
    return mysize;
}

void LinkedList::print(){
    struct nodo *aux;
    aux = head->siguiente;
    for(int i = 0; i<mysize; i++){
        int x = aux->n;
        cout<<x<<" ";
        aux=aux->siguiente;

    }
}
```

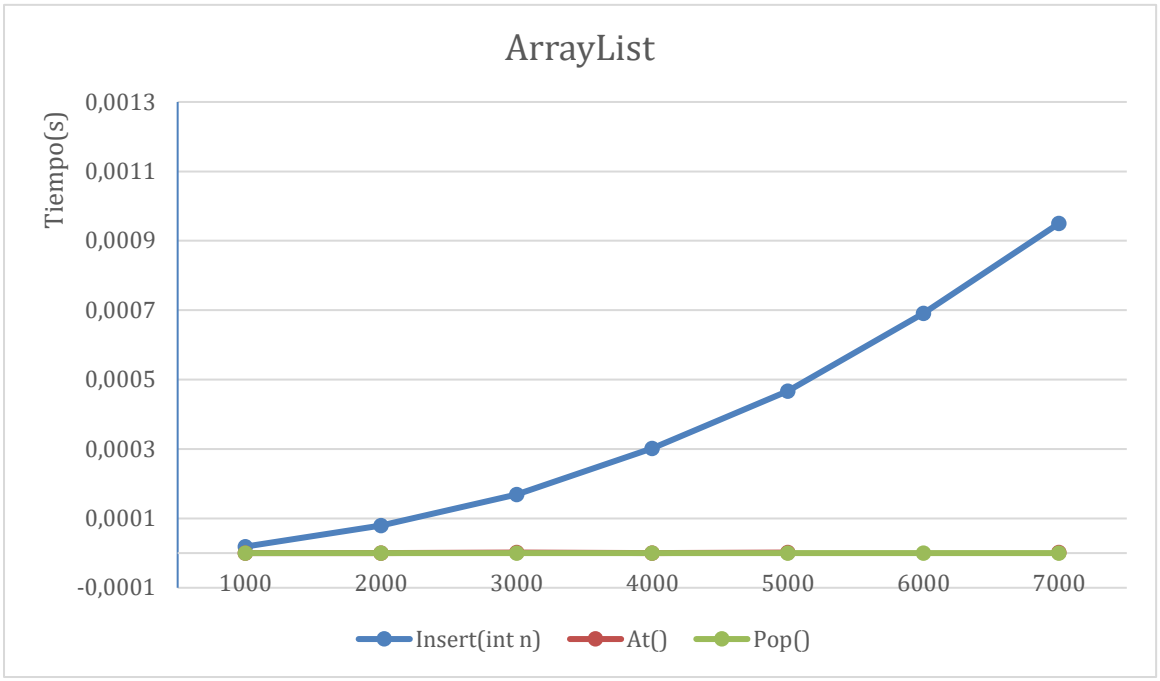
4.

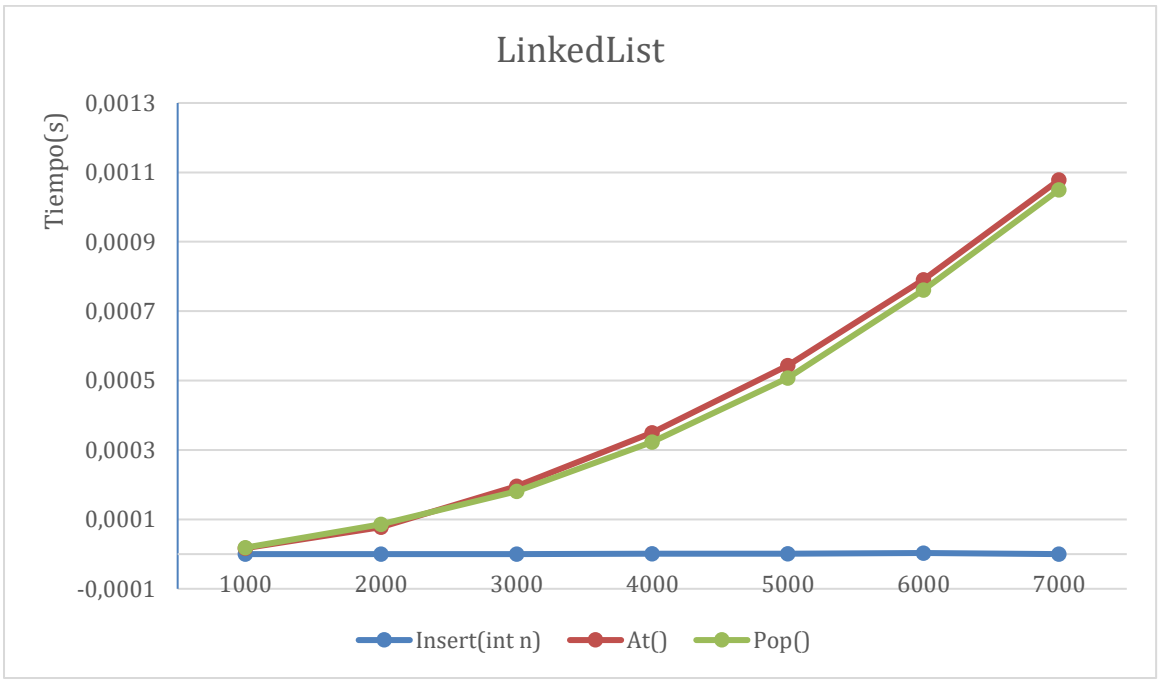
ArrayList

N	TIEMPO DE EJECUCION		
	Insert(int n)	At()	Pop()
1000	0.0018750000	0.0	0.0
2000	0.0079687500	0.0	0.0
3000	0.0168750000	0.0001562500	0.0
4000	0.0301562500	0.0	0.0
5000	0.0467187500	0.0001562500	0.0
6000	0.0690625000	0.0	0.0
7000	0.0950000000	0.0001562500	0.0

LinkedList

N	TIEMPO DE EJECUCION		
	Insert(int n)	At()	Pop()
1000	0.0	0.0017187500	0.0018750000
2000	0.0	0.0078125000	0.0085937500
3000	0.0	0.0195312500	0.0181250000
4000	0.0001562500	0.0350000000	0.0323437500
5000	0.0001562500	0.0543750000	0.0507812500
6000	0.0003125000	0.0790625000	0.0760937500
7000	0.0	0.1078125000	0.1050000000


ArrayList
Tiempo(s)
0,0013
0,0011
0,0009
0,0007
0,0005
0,0003
0,0001
-0,0001
1000
2000
3000
4000
5000
6000
7000
Insert(int n)
At()
Pop()


LinkedList
Tiempo(s)
0,0013
0,0011
0,0009
0,0007
0,0005
0,0003
0,0001
-0,0001
1000
2000
3000
4000
5000
6000
7000
Insert(int n)
At()
Pop()

5. ¿Cuál crees que es la mejor implementación para la ADT List? ¿Por qué?

A mi parecer se implementa mejor con la LinkedList, ya que hacer Insert() en ArrayList es muy costoso, ya que si o si recorre el array completo. Y si bien los otros métodos de ArrayList son menos costosos, Insert() es el más relevante a mi gusto y en el caso de LinkedList su costo es bajo y constante O(1) y bueno, hacer pop() si es muy costoso, aunque se podría solucionar teniendo un puntero al ultimo elemento. Y hacer at() en el peor de los casos al igual que como ocurre con ArrayList es O(n). Yo en lo particular prefiero la implementación ADT List en LinkedList básicamente por el método Insert()