Eficiencia teórica y empírica de los siguientes métodos.

Método INSERT

```
bool conjunto::insert( const conjunto::value_type & e){
	pair<conjunto::value_type, bool> par(find(e));
	bool insertado = par.second;

	if(!insertado){
		vm.push_back(e); O(1)
	}

	return insertado;
}
```

La eficiencia analizando el peor caso sería igualmente de O(1) ya que no hay bucle alguno.

Eficiencia empírica

Tiempo: 4.161e-06 s

Método FIND

```
pair<conjunto::value_type,bool>  conjunto::find (const string & chr, const unsigned int
& pos) const{
	bool encontrado = false;
	conjunto::value_type aux;

	pair<conjunto::value_type, bool> par(aux, encontrado);

	for(unsigned int i = 0; i < vm.size() && !encontrado; i++){ O(n)
		if(vm[i].getChr() == chr || vm[i].getPos() == pos) {
			par.first = vm[i];
			par.second = true;
		}
	}

	return par;
}
```

Este método tiene una eficiencia de **O(n)** ya que el peor caso sería que no está tal mutación y por tanto se recorrería todo el vector de mutaciones.

Eficiencia empírica

Tiempo: 7.2145e-05 s

Método ERASE

```
bool conjunto::erase(const string & chr, const unsigned int & pos){
	bool correcto = false;

	for(unsigned int i = 0; i < vm.size(); i++){  O(n)
		if(vm[i].getChr() == chr && vm[i].getPos() == pos){
			vm.erase(vm.begin() + i);
			correcto = true;
		}
	}

	return correcto;
}
```

```
bool conjunto::erase(const string & ID){
	bool correcto = false;

	for (unsigned int i = 0; i < vm.size(); i++){  O(n)
		if(vm[i].getID() == ID){
			vm.erase( vm.begin() + i );
			correcto = true;
		}
	}

	return correcto;
}
```

```
bool conjunto::erase(const conjunto::value_type & e){
	return erase(e.getID());    O(n)
}
```

En los tres métodos de borrado la eficiencia será de **O(n)** debido a que en el peor caso, que sería que no estuviera la buscada mutación, recorrería todo el vector, y por tanto, el mayor número de iteracciones. El tercer método tiene la misma eficiencia ya que llama al segundo, y este tiene O(n).

Eficiencia empírica

Tiempo: 1.0313e-05 s