# Ayudantía 8 - Algoritmos y Complejidad
# Analisis Amortizado

Amortized complexers

## 1. Analisis Amortizado

Cuando se hace el analisis asintotico de alguna operacion, aveces no resulta una vision ”realista” puesto que siempre se esta poniendo en el peor caso, pero generalmente las operaciones pesadas solo se realizan cada cierto tiempo. Aqui es donde entra el analisis amortizado que nos dice que evaluemos las operaciones en secuencia, puesto que una operacion cara viene luego de varias operaciones baratas.

**El costo *amortizado*** por operación en una secuencia de $n$ operaciones es el costo total de la secuencia divididio por $n$.

**Ojo** que *amortizado* y *promedio* son **cosas diferentes**. Considere el caso del arreglo dinamico.

### 1.1. Arreglo (Pila) dinamico(a)

Listing 1: Las operaciones basicas de una pila....

```
typedef T ...;
T A[SIZE];
static int top = 0;

void push(T *A, T v){
    A[top++] = v;
}

T pop(T *A){
    return A[--top];
}
```

¿Que hace si se llena? *realloc()*, por cuanto y cuando?
Considere que el costo de un PUSH, POP es simplemente el costo de copiar y el costo de expandir un arreglo de largo $n$ es copiar esos $n$ elementos a un nuevo espacio de memoria.
Expandimos en 1 cada vez que se llena? Si queremos llenar un Stack con $n$, hay que hacer $n$ push partiendo desde vacio, si el stack tiene tamaño $k$ hay que hacer $k+1$ copias ($k$ reallocs $+1$) por push:

$$\sum_{k=0}^{n-1}(k+1) = \frac{n(n+1)}{2}$$

Que es un costo amortizado de $(n+1)/2$ por push.
Y si vamos duplicando la capacidad del arreglo cada vez que expandemos?

$$1+(1+1)+(2+1)+1+(4+1)+1+1+1+\ldots+\left(2^k+1\right)+1+1+\ldots$$

Un push es una copia, y cada vez que pasamos de una potencia de dos se copian los elementos que llevamos. Tenemos 1 copia por push y $2^k$ cada vez que pasamos esa potencia, para llegar a $n$:

$$\begin{aligned}\sum_{k=0}^{n} 1+\sum_{k=0}^{\log_2(n-1)} 2^k &= n+2^{\lfloor \log_2(n-1)\rfloor+1}-1\\ &\leq n+2^{\log_2(n-1)+1}-1\\ &= n+2(n-1)-1\\ &= 3n-3\\ &< 3n\end{aligned}$$

Costo amortizado de 3 por push, mas bonito.

**Esto fue ANALISIS *AGREGADO***

## 1.2. Metodos de Analisis

Existen 3 metodos de analisis amortizado, el que vimos recien se llama analisis *Agregado* (summation method), existen tambien el "metodo contable"(taxation/accounting), la "funcion potencial"(potential method), veamoslo con el ejemplo clasico:

## 1.3. Ejemplo clasico: Binary Counter[1]

Tiene un contador binario (un numero binario) grande, inicializado en 0, en cada "tick" se incrementa en 1. Nuestro modelo de costos carga 1 unidad cada vez que un bit cambia. Despues de $n$ ticks el peor caso es $\lfloor \log_2 n\rfloor$, que es el numero de bits que cambian de 1 a 0.

### 1.3.1. Summation Method

Observe que "INCREMENT" no flipea $\log_2 n$ cada vez que se llama. El MSB $B[0]$ flipea en cada operacion, $B[1]$ flipea operacion por medio, $B[2]$ flipea cada 4 operaciones, $B[k]$ flipea cada $2^k$ ticks. Despues de una secuencia de $n$ INCREMENTs, $B[i]$ flipea $\lfloor \frac{n}{2^i} \rfloor$ veces. El total de flips de la secuencia de INCREMENTs:

$$\sum_{k=0}^{\lfloor \log_2 n\rfloor} \lfloor \frac{n}{2^k} \rfloor < \sum_{k=0}^{\infty} \frac{n}{2^k} = 2n$$

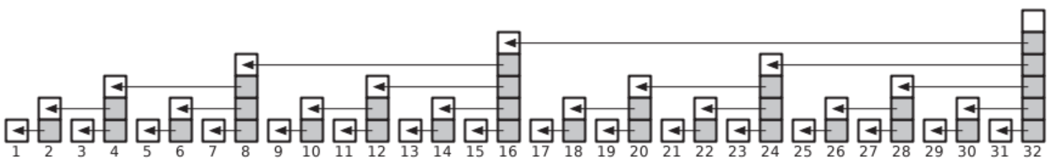
Por analisis amortizado el INCREMENT corre en tiempo constante 2

**Aggregate analysis** : Sea T(n) el tiempo de ejecucion del peor caso de una secuencia de n operaciones, el costo amortizado de cada operacion es T(n)/n.

[1] Para ejemplos solo busque en youtube binary counter, omita los videos de minecraft

### 1.3.2. Taxation/Accounting/charging method

Suponga que flipear un bit nos cuesta \$1 *VonBrand Coin*, asi que podemos medir el tiempo de ejecucion en dinero!
Cargue 2 unidades al incremento. Si cambia de $0 \to 1$ gasta 1 y ahorra 1, si cambia de $1 \to 0$ ocupa lo ahorrado para pagar los cambios (Osea, le estamos cargando el costo de limpiar un bit a las operaciones anteriores que lo pintaron). Si flipeamoos $k$ bits durante un incremento, cargamos $k-1$ a las operaciones anteriores (consumimos lo que ahorramos). Entonces, para la operacion inicial que pinta un unico bit, recibe a lo mas 1 cargo por la siguiente vez que el bit es limpiado, entonces en vez de pagar por $k$ flips, pagamos por a lo mas dos: el costo de pintar, y lo que dejamos guardado para pagar la limpieza. Entonces el costo amortizado es a lo mas 2




### 1.3.3. Potential Method

El mas poderoso y dificil, imita la idea de ”energia potencial”. En vez de asociar costos e impuestos a operaciones particulares, les asignamos una energia potencial que podemos gastar despues.
Supongamos que hay una secuencia de operaciones $o_1, o_2, o_3, \ldots$ que llevan a la estructura de datos desde el esto inicial $S_0$ hasta un estado $S_i$. Sea $\Phi(S_i)$ el potencial del estado $S_i$ y sea $c_i$ el costo de la operacion $o_i$. El costo amortizado de la operación $o_i$ es:

$$a_i = c_i + \Phi(S_i) - \Phi(S_{i-1})$$

Es decir:

$$\text{costo amortizado} = \text{costo real} + \text{cambio en potencial}$$

Entonces la suma de $n$ operaciones es el costo actual mas el incremento total en potencial:

$$\sum_{i=1}^{n} a_i = \sum_{i=1}^{n} (c_i + \Phi(S_i) - \Phi(S_{i-1})) = \sum_{i=1}^{n} c_i + (\Phi(S_n) - \Phi(S_0))$$

Una funcion potencial es valida si $\Phi(S_i) - \Phi(S_0) \geq 0 \forall i$, el potencial siempre crece respecto al inicio.
Para nuestro caso nos definimos el potencial $\Phi(S_i)$ como la cantidad de bits con valor 1. Notamos que $\Phi(S_0) = 0$ y $\Phi(S_i) > 0 \forall i > 0$, asi que es una funcion potencial valida. Podemos describir el costo real de un INCREMENT y el cambio de potencial en terminos de la cantidad de bits que cambian:

$$\begin{aligned} c_i &= \text{Numero de bits que cambian de 0 a 1 + numero de bits que cambian de 1 a 0} \\ \Phi(S_i) - \Phi(S_i - 1) &= \text{Numero de bits que cambian de 0 a 1 - numero de bits que cambian de 1 a 0} \end{aligned}$$

Entonces el costo amortizado es:

$$a_i = c_i + \Phi(S_i) - \Phi(S_{i-1}) = 2 \text{ veces el numero de bits que cambian de 0 a 1}$$

Como INCREMENT solo cambia un bit de 0 a 1, el costo amortizado de INCREMENT es 2.
Esto de la funcion potencial es bien magico y no hay una receta general para diseñar funciones de potencial, solo necesita imaginacion.

# 2. Ejercicios

## 2.1. Amort1

Una estructura de datos da un costo máximo total de $O(n \log n)$ al efectuar $n$ operaciones

1. ¿cual es el costo amortizado por operacion?

   **R:**

   $$\frac{O(n \log n)}{n} = O(logn)$$

2. ¿cual es el costo maximo por operacion?

   **R:** $O(n \log n)$, si son todas $O(1)$ la ultima tiene que ser $O(n \log n)$

3. ¿cual es el costo minimo por operacion?

   **R:** $O(1)$

4. ¿En que situaciones deberia aplicarse analisis asintotico tradicional peor caso y no amortizado?

   **R:** Al análisis amortizado es aplicable cuando lo relevante es el costo total de una secuencia de operaciones. Si lo que importa es el costo de una operación individual (como en sistemas de tiempo real o sistemas interactivos), debe considerarse el costo máximo.

## 2.2. Amort2

Dado un arreglo no ordenado de $n$ elementos diferentes, se procesan en orden $m$ consultas sobre si está o no presente un elemento. Dependiendo de $n$ y $m$, ¿cómo determina cuándo conviene buscar en el arreglo desordenado o primero ordenarlo y usar búsqueda binaria? Suponga que ordenar el arreglo toma tiempo $2(n+1)\log n$, búsqueda binaria toma $2\log n$, búsqueda lineal $n$ si falla y $n/2$ si tiene éxito (suponga 60 % éxitos y 40 % fallas).

**R:** El costo total de la secuencia de $m$ operaciones, si no se ordena, es:

$$m \cdot \left(0{,}6 \cdot \frac{n}{2} + 0{,}4n\right) = 0{,}7 \cdot m \cdot n$$

Si se ordena el costo total es:

$$2(n+1)\log n + m \cdot 2\log n = 2m\log n + 2(n+1)\log n$$

Estas son ecuaciones lineales en $m$, su intersección define el punto en que deja de ser ventajoso búsqueda lineal:

$$\begin{aligned} 0,7m^{\star} n &= 2m^{\star} \log n + 2(n+1)\log n \\ m^{\star} &= \frac{2(n+1)\log n}{0,7n - 2\log n} \end{aligned}$$

Un Grafico de $m$(eje y) vs $n$(eje x) muestra que $m$ es bastante pequeño sin importar el $n$, por lo que nunca es conveniente busqueda binaria. https://www.desmos.com/calculator/fehbewqhru