HASHING

RENDIMIENTO	ORDENACIÓN	BÚSQUEDA
Normal	N^2	N
Bueno	$N \log N$	$\log N$
Óptimo	N	$O(1)$

CONSTRUCCIÓN DE FUNCIONES HASH

OBJETIVO: Construir función hash h tal que si $K \neq K'$ la probabilidad de que $h(K) = h(K')$ sea pequeña.

$$p(h(K) = h(K')) = \frac{1}{M} \leftarrow \text{tabla de M posiciones}$$

si la función hash verifica lo anterior se dice <u>función hash uniforme</u>.

Hash de división: fijamos un <u>número primo m</u> mayor que el número de datos.

$$\boxed{h_d(K) = K \% m}$$

Hash de multiplicación: escogemos <u>m no primo</u> mayor que el número de datos, y un número irracional:

$$\boxed{h_m(K) = \lfloor m.(K.\Phi)\rfloor} \quad \text{número irracional}$$

donde ( ) es la parte fraccionaria del resultado de multiplicar K por el número irracional.

MECANISMOS DE RESOLUCIÓN DE COLISIONES

HASH CON ENCADENAMIENTO

Sea h función hash uniforme, tabla de dimensión m y N datos, entonces:

$$\boxed{A^{f}_{HE}(N,m) = \lambda = \frac{N}{m}}$$

$\lambda \equiv$ factor de carga

$$\boxed{A^{e}_{HE}(N,m) = 1 + \frac{\lambda}{2} + O(1)}$$

Importante:

$$\boxed{n^{e}_{HE}(D_i, T) = 1 + n^{f}_{HE}(D_i, T_i) \simeq 1 + A^{f}_{HE}(i-1, m)}$$

$$A^{e}_{HE}(N,m) = \frac{1}{N}\sum_{i=1}^{N} n^{e}_{HE}(D_i, T) = \frac{1}{N}\sum_{i=1}^{N}\Big(1 + \overset{\rightarrow (i-1,m)}{n^{f}_{HE}(D_i, T_i)}\Big) =$$

MÉTODO INTEGRAL
(estimación)

$$= \frac{N}{N} + \frac{1}{N}\sum_{i=1}^{N} n^{f}_{HE}(D_i, T_i) \simeq$$

$$\simeq 1 + \frac{1}{\lambda}\int_0^{\lambda} \underbrace{\phi(u)\,du}_{\overset{\|}{A^{f}_{HE}(N,m)}}$$

MÉTODO SUMATORIO (más exacto)

$$= \frac{1}{N}\sum_{i=1}^{N}\Big(1 + \overset{\text{sup. } A^{f}_{HE}(N,m) = \lambda}{\frac{i-1}{m}}\Big) =$$

$$= 1 + \frac{1}{N.m}\sum_{i=1}^{N-1} i = 1 + \frac{1}{N.m}\cdot\frac{N(N-1)}{2}$$

$$= 1 + \frac{N}{2m} - \frac{1}{2m} = 1 + \frac{\lambda}{2} + O(1)$$

HASH CON DIRECCIONAMIENTO ABIERTO

- → SONDEOS LINEALES: $(p+1)\%m$, $(p+2)\%m$, ...
- → SONDEOS CUADRÁTICOS: $(p+1^2)\%m$, $(p+2^2)\%m$, $(p+3^2)\%$, ...
- → SONDEOS ALEATORIOS: random numbers

PROPOSICIÓN: Sea h una función hash uniforme y se usan <u>sondeos aleatorios</u> entonces:

$$A^{f}_{DA\_Aleat}(N,m) = \frac{1}{1-\lambda}$$

$$A^{e}_{DA\_Aleat}(N,m) = \frac{1}{2}\log\left(\frac{1}{1-\lambda}\right)$$

Importante:

$$\boxed{n^{e}_{DA}(D_i,T) = n^{f}_{DA}(D_i,T_i) \approx A^{f}_{DA}(i-1,m)}$$

$$A^{e}_{DA}(N,m) = \frac{1}{N}\sum_{i=1}^{N} n^{e}_{DA}(D_i,T) = \frac{1}{N}\sum_{i=1}^{N} n^{f}_{DA}(D_i,T_i) \approx \frac{1}{N}\sum_{i=1}^{N} A^{f}_{DA}(i-1,m)$$

$$\approx \frac{1}{N}\sum_{i=0}^{N-1} \phi\left(\frac{i}{m}\right) \approx \frac{1}{N}\int_0^N \phi\left(\frac{x}{m}\right)dx \underset{\downarrow}{=} \frac{1}{\lambda}\int_0^\lambda \phi(u)\,du$$

$u = \frac{x}{m}$; $m\,du = dx$

donde $\phi(\lambda) = A^{f}_{DA}(N,m)$

Por lo tanto, cuando $A^{f}_{\underbrace{DA}_{\to \text{ sondeos cualesquiera}}}(N,m) = \phi(\lambda)$

$$A^{e}_{DA}(N,m) \simeq \frac{1}{\lambda}\int_0^\lambda \phi(u)\,du$$

Ejemplificando, para sondeos lineales:

$$A^{f}_{DA\_SL}(N,m) = \frac{1}{2}\left(1+\frac{1}{(1-\lambda)^2}\right)$$

$$A^{e}_{DA\_SL}(N,m) \simeq \frac{1}{\lambda}\int_0^\lambda \frac{1}{2}\left(1+\frac{1}{(1+u)^2}\right)du = \frac{1}{2}\left(1+\frac{1}{1-\lambda}\right)$$

SELECTSORT

Pseudocódigo

```
void SelectSort(Tabla T, ind P, ind U)
    i=P;
    mientras i<U:
        min=i;
        para j de i+1 a U:
            si T[j]<T[min]
                min=j;
        swap(T[i],T[min]);
        i++;
```

Ejemplo:

Ordenar T = [4,3,2,1]

ter 1: i=0 | j=1 | min=4 → min=1 | swap(T[0], T[3])

T = [1,3,2,4]

ter 2: i=1 | j=2 | min=3 → min=2 | swap(T[1], T[2])

T = [1,2,3,4]

iter 3: i=2 | j=3 | min=3 → min=3 | swap(T[2], T[2])

La idea de SelectSort es recorrer la tabla, encontrar el mínimo y ponerl en la primera posición. Repetir con la subtabla desordenada (el primero ya está colocado); así hasta el final.

No es recursivo, se coge el primer elemento como mínimo y se percorre el resto de la tabla actualizando el mínimo cada vez que se encuentra uno menor para, al final, intercambia el mínimo por el primero de la tabla desordenada.

Cálculo de $n_{SS}(T,P,U)$

$$n_{SS}(T,1,N) = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} 1 = \sum_{i=1}^{N-1} (N-i) =$$

$$= \frac{N(N-1)}{2} = \frac{N^2}{2} - \frac{N}{2}$$

BUBLESORT

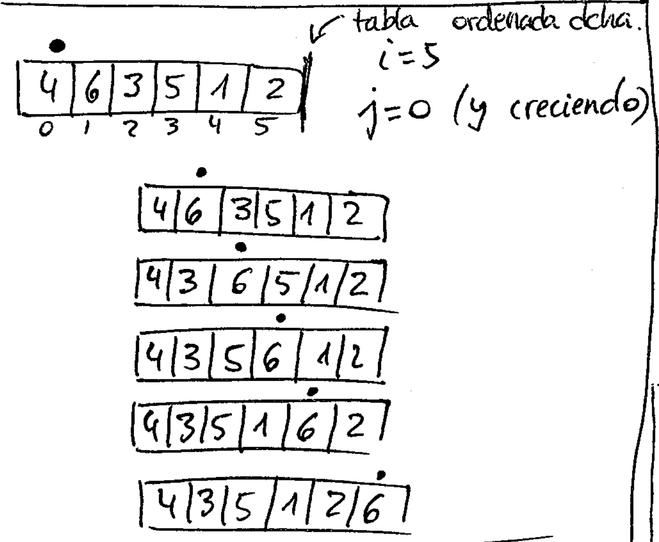
Pseudocódigo

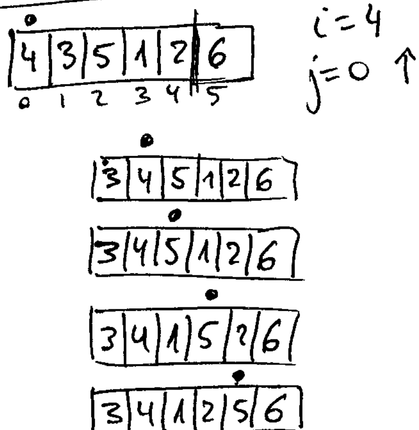
```
BubleSort_v1(Tabla T, ind P, ind U)
    para i de U a P+1:
        para j de P a i-1:
            si T[j] > T[j+1]:
                swap(T[j], T[j+1]);
```

La idea de Bublesort es seleccionar el primer elemento como "burbuja" y este intenta subir ($T[j] > T[j+1] \rightarrow$ swap(T[j], T[j+1])). Si no puede, se cambia la burbuja por el siguiente, así hasta el final de la tabla desordenada. El índice i marca la frontera entre la tabla desordenada y la ordenada.

Ejemplo

Ordenar $T = [4, 6, 3, 5, 1, 2]$
0 1 2 3 4 5





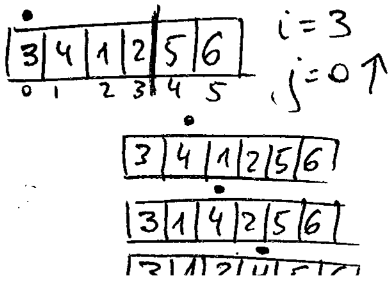
Cálculo de $n_{BS}(T, P, U)$

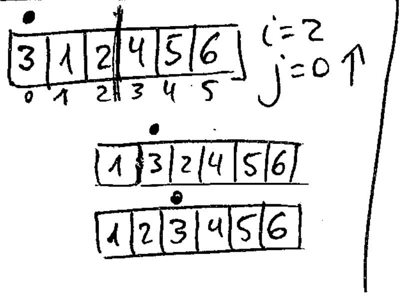
$$n_{BS}(T,1,N) = \sum_{i=2}^{N} \sum_{j=0}^{i-1} 1 = \sum_{i=2}^{N} (i-1) = \sum_{i=1}^{N-1} i = \frac{N^2}{N} - \frac{N}{2}$$

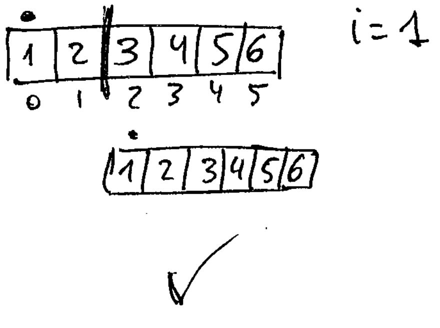
(*) MEJORA BUBLESORT

```
BubleSort_flag(Tabla T, ind P, ind U)
    flag = 1; i = U;
    mientras(flag == 1 AND i ≥ P+1):
        flag = 0
        para j de P a i-1:
            si T[j] > T[j+1]
                swap(T[j], T[j+1]); flag = 1;
        i--;
```

Ahora para el caso mejor $n_{BS\_flag} = N - 1$ pero el peor $n_{BS\_flag} \leq \frac{N^2}{\cancel{2}} - \frac{N}{2}$







INSERTSORT

Pseudocódigo

```
InsertSort(Tabla T, ind P, ind U)
    para i de P+1 a U
        A = T[i];
        j = i-1;
        mientras (j ≥ P && T[j]>A);
            T[j+1] = T[j];
            j--;
        T[j+1] = A;
```

La idea de InsertSort es seleccionar un elemento, copiarlo, y compararlo con todos los anteriores en una tabla, deteniénd cuando encuentre uno menor e insertándolo delante de él (habiendo desplazado los mayores previamente un lugar a la derech

Ejemplo

T = | 5 | 4 | 2 | 3 | 1 |

| 5 | (4) | 2 | 3 | 1 | (índices 0 1 2 3 4) i = 1, j = i-1 = 0

| 5 | 5 | 2 | 3 | 1 | j--

| 4 | 5 | 2 | 3 | 1 |

| 4 | 5 | (2) | 3 | 1 | (índices 0 1 2 3 4) i = 2, j = 1

| 4 | 5 | 5 | 3 | 1 | j--

| 4 | 4 | 5 | 3 | 1 | j--

| 2 | 4 | 5 | 3 | 1 |

| 2 | 4 | 5 | (3) | 1 | i = 3, j = 2

| 2 | 4 | 5 | 5 | 1 | j--

| 2 | 4 | 4 | 5 | 1 | j--

| 2 | 3 | 4 | 5 | 1 |

| 2 | 3 | 4 | 5 | (1) | i = 4, j = 3

| 2 | 3 | 4 | 5 | 5 | j--

| 2 | 3 | 4 | 4 | 5 | j--

| 2 | 3 | 3 | 4 | 5 | j--

| 2 | 2 | 3 | 4 | 5 | j--

| 2 | 3 | 4 | 5 |

CASO MEJOR Y PEOR

Tenemos que $1 \leq n_{is}(\sigma, i) \leq i-1$, por lo que: $\forall \sigma \in \Sigma_N$:

$$\sum_{i=2}^{N} 1 \leq \sum_{i=2}^{N} n_{is}(\sigma, i) \leq \sum_{i=2}^{N} (i-1) \Rightarrow N-1 \leq n_{is}(\sigma) \leq \frac{N(N-1)}{2}$$

- Caso peor $\left\{ \begin{array}{l} \forall \sigma \in \Sigma_N : n_{is}(\sigma) \leq \frac{N(N-1)}{2} \\ n_{is}([N, N-1, \dots, 2, 1]) = \frac{N(N-1)}{2} \end{array} \right\} \Rightarrow W_{is}(N) = \frac{N(N-1)}{2}$

- Caso mejor $\left\{ \begin{array}{l} \forall \sigma \in \Sigma_N : n_{is}(\sigma) \geq N-1 \\ n_{is}([1, 2, \dots, N-1, N]) = N-1 \end{array} \right\} \Rightarrow B_{is}(N) = N-1$

- Caso medio (demostración larga):

$$A_{is}(N) = \frac{N^2}{4} + O(N)$$

MERGESORT

Pseudocódigo

```
status Mergesort(tabla T, ind P, ind U)
    si P>U: devolver ERROR;
    si P==U: devolver OK;
    else:
        $M = \frac{(P+U)}{2}$;

        Mergesort(T,P,M);
        Mergesort(T,M+1, U);
    devolver Combinar(T,P,M,U);

status Combinar(Tabla T, ind P, ind U, ind M)
    T' = tablaAux(P,U);
    i=P; j=M+1 ; K=P;
    mientras i<=M && j<=U:
        si T[i] < T[j]: T'[K] = T[i]; i++;
        else :          T'[K] = T[j];
        K++;
        si i>M:
            mientras j<=U:
                T'[K] = T[j]; j++; K++;
        else si j>U:
            mientras i<=M:
                T'[K] = T[i]; i++; K++;

        Copiar(T', T, P, U);
        Free(T');
        devolver OK;
```

CASO PEOR: $N\log N + O(N)$

CASO MEDIO: $\theta(N\log N)$

CASO MEJOR: $\frac{N}{2}\log N + O(N$

QUICKSORT

Pseudocódigo

```
status Quicksort(Tabla T, ind P, ind U)
    si P>U : devolver ERROR;
    si P==U : devolver OK;
    else:
        M= partir(T,P,U)
        si P< M-1 :
            Quicksort(T,P,M-1);
        si M+1 < U :
            Quicksort(T,M+1,U);
    devolver OK;

ind Partir(Tabla T, ind P, ind U)
    M= medio (T,P,U);  <- devuelve la posición del pivote
    K= T[M];
    swap(T[P], T[M]);
    M=P;
    para i de P+1 a U:
        si T[i] < K :
            M++;
            swap(T[i],T[M]);
    swap(T[P], T[M]);
    devolver M;
```

CASO PEOR : $\frac{N^2}{2} - \frac{N}{2}$

CASO MEDIO : $2N\log N + O(N)$

CASO MEJOR : desconocido

HEAPSORT

Pseudocódigo

```
HeapSort(Tabla T, int N)
    CrearHeap(T,N);
    OrdenarHeap(T,N);

CrearHeap(Tabla T, int N)
    si N<2: return;
    para i de $\frac{N-2}{2}$ a 0:
        heapify(T,N,i);

OrdenarHeap(Tabla T, int N)
    para i de N-1 a 1:
        swap(T[0], T[i])
        heapify(T, N, 0)

Heapify(Tabla T, int N, ind i)
    mientras (2*i +1 <=N):
        ind = max(T,N, i, izq(i), der(i));
        si ind != i :
            swap(T[i], T[ind]);
            i=ind;
        else:
            return;
```

CASO PEOR: $\Theta(N \log N)$

CASO MEDIO: $\Theta(N \log N)$