

Índice

Calculo de Orden	2
Fuerza Bruta	2
Ordenar y Seleccionar	3
K-Selecciones.....	4
K-HeapSort	5

Calculo de Orden de los Algoritmos Solicitados

Fuerza Bruta

```

public boolean calcularPorFuerzaBruta(Integer pos, Integer valor){
    Integer aDevolver = null;          //1

    if (esKEsimoMenor(pos,valor)){      //1 + T(esKEsimoMenor)
        aDevolver = contenedorDatos.get(pos); //2
    }

    return aDevolver != null;           // 2
}

private boolean esKEsimoMenor(int k,int valor){

    int vecesMenor = 0;                // 1

    for(int nro : contenedorDatos){     // 1..N
        if (valor < nro){                //1
            vecesMenor++;                //1
        }
    }

    return (contenedorDatos.size() - vecesMenor) == k; // 3
    // devuelve true si es el k-menor elemento....
}

```

Orden

Para un arreglo de n elementos, este método realiza n iteraciones. En cada una de ellas elige un mínimo y lo compara contra los n elementos del arreglo. Por lo tanto, la cantidad de operaciones que requiere este elemento son

$$T(n) = 1 + 1 + 1 + \sum_{i=1}^n [1 + 1] + 3 = 5 + \sum_{i=1}^n (2) = 5 + 2n \in O(n)$$

Ordenar y Seleccionar

```

public boolean calcularPorOrdenarSeleccionar(Integer pos, Integer valor){
    ordenarPorBurbujeo(); T(ordenarPorBurbujeo)

    return verificarPos(pos-1,valor); T(verificarPos)
}

private void ordenarPorBurbujeo(){
    int i, j, aux;

    for(i=0;i<contenedorDatos.size()-1;i++) //n
        for(j=0;j<contenedorDatos.size()-i-1;j++) //n

            if(contenedorDatos.elementAt(j+1)<contenedorDatos.elementAt(j)){ //3
                aux=contenedorDatos.elementAt(j+1); //3

                contenedorDatos.set(j+1,contenedorDatos.elementAt(j)); //3

                contenedorDatos.set(j,aux); //1
            }
}

private boolean verificarPos(Integer pos, Integer valor){
    if(contenedorDatos.elementAt(pos) == valor) //2
        return true; //1

    return false; //1
}

```

Orden

Para un arreglo de n elementos, este método realiza $n*n$ iteraciones recorriendo en dos sentidos en vector intercambiando los mínimos que vayan surgiendo

$$T(n) = \sum_{i=1}^n \sum_{i=1}^n (3 + 3 + 3) + 2 + 1 = n(n + 9) = n^2 + 9n \in O(n^2)$$

K-Selecciones

```

public boolean calcularPorKSelecciones(Integer pos, Integer valor){
    Integer minimo = 2147483647 , indiceMin = 0;           //2

    for (int i = 0; i < pos; i++) {                          //n
        for (int j = 0; j < contenedorDatos.size(); j++) {  //n
            if(contenedorDatos.elementAt(j) < minimo){      //2
                minimo = contenedorDatos.elementAt(j);      //2
                indiceMin = j;                               //1
            }
        }
        selecciones.set(i,minimo);                          //1
        contenedorDatos.removeElementAt(indiceMin);         //1
        minimo = 2147483647 ; indiceMin = 0;               //2
    }
    return verificarPosPorSeleccion(pos-1,valor);           //1 + T(VerPos)
}

private boolean verificarPosPorSeleccion(Integer pos, Integer valor) {
    if(selecciones.elementAt(pos) == valor)                //1
        return true;                                       //1

    return false;                                           //1
}

```

Orden

Para un arreglo de n elementos, este método realiza pos iteraciones que son las K selecciones, en el peor caso K es máximo por lo que pos toma el valor de n , luego entra en otro for donde se operan n veces. Por lo tanto, la cantidad de operaciones que requiere este elemento son

$$\begin{aligned}
 T(n) &= 2 + \sum_{i=1}^n \sum_{j=1}^n (2 + 2 + 1) + 1 + 1 + 2 + 1 + 2 = 5 + n(n + 5) \\
 &= n^2 + 5n + 5 \in O(n^2)
 \end{aligned}$$

K-HeapSort

```

public boolean calcularPorKSeleccionesEnHeap(Integer pos, Integer valor){
    Integer minimo = 2147483647 , indiceMin = 0;           // 2

    for (int i = 0; i < pos; i++) {                        //n
        for (int j = 0; j < contenedorDatos.size(); j++) { //n
            if(contenedorDatos.elementAt(j) < minimo){      //2
                minimo = contenedorDatos.elementAt(j);    //2
                indiceMin = j;                             //1
            }
        }
        heap.agregar(minimo);                               //log2(n)
        contenedorDatos.removeElementAt(indiceMin);        //1
        minimo = 2147483647 ; indiceMin = 0;              //2
    }

    if( buscarRefEnHeap(pos,valor) )                       // T(buscarRefEnHeap)
        return true;                                       //1

    return false;                                          //1
}

private boolean buscarRefEnHeap(Integer pos, Integer valor) {
    --pos;                                                 //1
    while(pos > 0){                                         //n
        heap.eliminarMin();                                //log2(n)
        --pos;                                             //1
    }
    if (heap.obtenerMin() == valor)                       //2
        return true;                                       //1

    return false;                                          //1
}

```

Orden

Para un arreglo de n elementos, este método realiza pos iteraciones que son las K selecciones, en el peor caso K es máximo por lo que pos toma el valor de n , luego entra en otro for donde se operan n veces. Por lo tanto, la cantidad de operaciones que requiere este elemento son

$$T(n) = 2 + \sum_{i=1}^n \left[\sum_{j=1}^n (2 + 2 + 1) \right] \log_2(n) + 1 + 2 \Bigg] + 1 + n \cdot \log_2(n) =$$

$$T(n) = 2 + \sum_{i=1}^n [5n \log_2(n) + 1 + 2] + 1 + n \cdot \log_2(n) =$$

$$T(n) = 5n^2 \log_2(n) + 3n + n \cdot \log_2(n) + 3 \in \mathbf{O(n^2 \cdot \log_2(n))}$$