Tiempo de ejecución de *revertirAudio* $O(n)$, *limpiarAudio* $O(n^2)$, *maximosTemporales* $O(m \times n)$ justificados:

```
audio revertirAudio(audio a, int canal, int profundidad) {
audio b; ---------------------------------------------------------------------- O(1)
        for (int i=0; i<(a.size()/canal); i++) {
                revertirBloque(a, b, canal, i); ---------------------------------- O(c)
        } ---------------------------------------------------------------------- O(n/c) siendo n = a.size()
return b;                                                                          y c = canal
}


void revertirBloque(audio a, audio &b, int canal, int i) {
    for (int j=0; j<canal; ++j) {
        b.push_back(a[a.size() - canal*(i+1) + j]);
    } ---------------------------------------------------------------------- O(1)
} ------------------------------------------------------------------------- O(c)
```

Llamamos k a la suma de las operaciones $O(1)$. Para cada ejecución del primer ciclo se ejecuta el segundo (revertirBloque), entonces queda como $O((n/c) \times c + k)$ y simplifica $(n/c) \times c$ quedando entonces $O(n)$.

```
void limpiarAudio(audio &a, int profundidad, vector<int> &outliers) {
    int percentil95 = 0;---------------------------------------------------------- O(1)
    audio a0 = a;----------------------------------------------------------------- O(1)
    if (a.size()>1) {
        buscoOutliers(a, outliers, percentil95);
    } ---------------------------------------------------------------------- O(1)
    if (outliers.size()>0) {
        reemplazoOutliers(a0, outliers, percentil95);
    } ---------------------------------------------------------------------- O(1)
    a = a0;
}

void buscoOutliers (audio a, vector<int> &outliers, int &percentil95){
    audio a0 = a; ---------------------------------------------------------------- O(1)
    audio audioOrdenado = selectionSort(a0); ------------------------------------- O(n²) siendo n = a.size()
    percentil95 = audioOrdenado[(int) (floor(((a.size() * 95) / 100) - 1))]; ----- O(1)
    for (int i = 0; i < a.size(); ++i) {
        if (a[i] > percentil95) {
            outliers.push_back(i);
        }
    } ---------------------------------------------------------------------- O(n) siendo n = a.size()

}


void reemplazoOutliers (audio &a, vector<int> &outliers, int &percentil95){
    for (int i = 0; i < outliers.size(); ++i) {   //O(M) siendo M = la cantidad de outliers
        int noOutlierDerecha = buscarNoOutlierDerecha(a, outliers[i], percentil95); ---------O(n-i) con n=a.size()
        int noOutlierIzquierda = buscarNoOutlierIzquierda(a, outliers[i], percentil95); ----O(i) con i = la posición del
                                                                                                 outlier
        if ((noOutlierDerecha >= 0) && noOutlierIzquierda >= 0) {
            double b = (a[noOutlierDerecha] + a[noOutlierIzquierda]); ----------------------O(1)
```

```
            b = floor(b / 2);                                                    O(1)
            a[outliers[i]] = (int) b;                                            O(1)
        }                                                                        O(1)
        if ((noOutlierDerecha >= 0) && (noOutlierIzquierda == -1)) {
            a[outliers[i]] = a[noOutlierDerecha];
        }                                                                        O(1)
        if ((noOutlierDerecha == -1) && (noOutlierIzquierda >= 0)) {
            a[outliers[i]] = a[noOutlierIzquierda];
        }                                                                        O(1)
    }                                                                            O(M) con M = cantidad de
                                                                                      outliers
}

int buscarNoOutlierDerecha(audio a, int i, int percentil95) {
    int noHayNoOutlier = -1;                                                     O(1)
    for (int j = i + 1; j < a.size(); j++) {
        if (a[j] < percentil95) {
            return j;
        }                                                                        O(1)
    }                                                                            O(a.size() - I)
    return noHayNoOutlier;
}                                                                                O(1)


int buscarNoOutlierIzquierda(audio a, int i, int percentil95) {
    int noHayNoOutlier = -1;                                                     O(1)
    for (int j = i - 1; j >= 0; j--)
        if (a[j] < percentil95) {
            return j;
        }                                                                        O(1)
    }                                                                            O(a.size() - I)
    return noHayNoOutlier;//O(1)
}

audio selectionSort(audio &a) {
    int aux;
    for (int j = 0; j < a.size() - 1; ++j) {
        int min = a[j];
        aux = j;
        for (int i = j + 1; i < a.size(); ++i) {
            if (min > a[i]) {
                min = a[i];
                aux = i;
            }
        }
        swap(a[j], a[aux]);
    }
    return a;
}                                                                                O(n²) con n = a.size()
                                                                                 Complejidad demostrada en clase
```

Llamamos c a la suma de las operaciones $O(1)$. En el peor caso el tiempo de ejecución es $O(n^2 + n + m \times ((n-i) + i) + c)$. como al momento de calcular complejidad nos importa el grado mas grande, podemos acotar el termino menor a n^2 por un k, quedando así $O(n^2 \times k) = O(n^2)$

```
void maximosTemporales(audio a, int profundidad, vector<int> tiempos, vector<int> &maximos,
vector<pair<int, int> > &intervalos) {
    conseguirIntervalos (a, tiempos, intervalos);
    maximosDeLosIntervalos(a, maximos, intervalos);
}

void conseguirIntervalos(audio a, vector<int> tiempos, vector<pair<int, int> > &intervalos) {
    for (int j = 0; j < tiempos.size(); ++j) {
        for (int i = 0; i < a.size(); i += tiempos[j]) {
            pair<int, int> intervalo = {i, i + tiempos[j] - 1};---------- O(1)
            intervalos.push_back(intervalo); ---------- O(1)
        } ---------- Este for cicla n/m veces
                     Entonces es O(n) en el peor
                     caso
    } ---------- O(m) con m = tiempos.size()
} ---------- O(m * n )


void maximosDeLosIntervalos(audio a, vector<int> &maximos, vector<pair<int, int> > &intervalos) {
    for (int i = 0; i < intervalos.size(); ++i) {
        int max = 0
        for (int j = intervalos[i].first; j < a.size() && j <= intervalos[i].second; ++j) {
            if (a[j] > max) {
                max = a[j];
            } ---------- este for cicla t(i)
                         siendo cada tiempo particular

        }
        maximos.push_back(max);
    }
}
```

El ciclo *for* principal de *maximosDeLosIntervalos* cicla la cantidad de veces que es la suma de los tiempos de la lista de tiempos ($(a.size())/t_1 + (a.size())/t_2 + \dots (a.size())/t_n$). Por cada uno de estos intervalos, va a ciclar otra vez por el tiempo de ese intervalo, quedando la suma como $(a.size())/t_1 \times t_1 + (a.size())/t_2 \times t_2 + \dots (a.size())/t_n \times t_n$. Simplificando quedaría $n \times m$ veces.

Sumando las 2 funciónes, la complejidad total seria $O((n \times m) + (n \times m)) = O(2 \times (n \times m)) = O(n \times m)$

Tiempos de ejecución en el peor caso

```
void magnitudAbsolutaMaxima(audio a, int canal, int profundidad, vector<int> &maximos, vector<int> &posicionesMaximos) {
    for (int i = 0; i < canal; ++i) {
        int max = 0;                                                    O(1)
        int posMax = 0;                                                 O(1)
        maximoDelCanal(a, canal, i, max, posMax);                       O(n/c)
        maximos.push_back(max);                                         O(1)
        posicionesMaximos.push_back(posMax);                            O(1)
    }                                                                   con canal = c
                                                                        cicla c veces, es O(c)
}

void maximoDelCanal(audio a, int canal, int i, int &max, int &posMax) {
    for (int j = i; j < a.size(); j += canal)
        if (abs(a[j]) > max) {
            max = a[j];                                                 O(1)
            posMax = j;                                                 O(1)
        }
    }                                                                   Sea a.size() = n
                                                                        esto cicla n/c veces
}
```

Para cada ciclo $O(c)$ hay un ciclo $O(n/c)$, entonces la complejidad total es $O((n/c) \times c)$, simplificando es $O(n)$

```
Void audiosSoftYHard(vector<audio> as, int profundidad, int longitud, int umbral, vector<audio> &soft, vector<audio>
&hard) {
    for (int i = 0; i < as.size(); ++i) {
        int contador = 0;                                               O(1)
        bool esHard = esHardOSoft(as, longitud, umbral, contador, i);   O(L)
        if (esHard == true) {                                           O(1)
            hard.push_back(as[i]);                                      O(1)
        } else {
            soft.push_back(as[i]);                                      O(1)
        }
    }                                                                   O(n) con n = as.size()

}

bool esHardOSoft(vector<audio> as, int longitud, int umbral, int contador, int i) {
    bool esHard = false;                                                O(1)
    for (int j = 0; j < as[i].size(); ++j) {
        if (contador == longitud + 1) {                                 O(1)
            esHard = true;                                              O(1)
        }
        if (as[i][j] > umbral) {                                        O(1)
            contador++;                                                 O(1)
        } else {
            contador = 0;
        }                                                               O(1)
    }                                                                   O(L)
    return esHard;
}
```

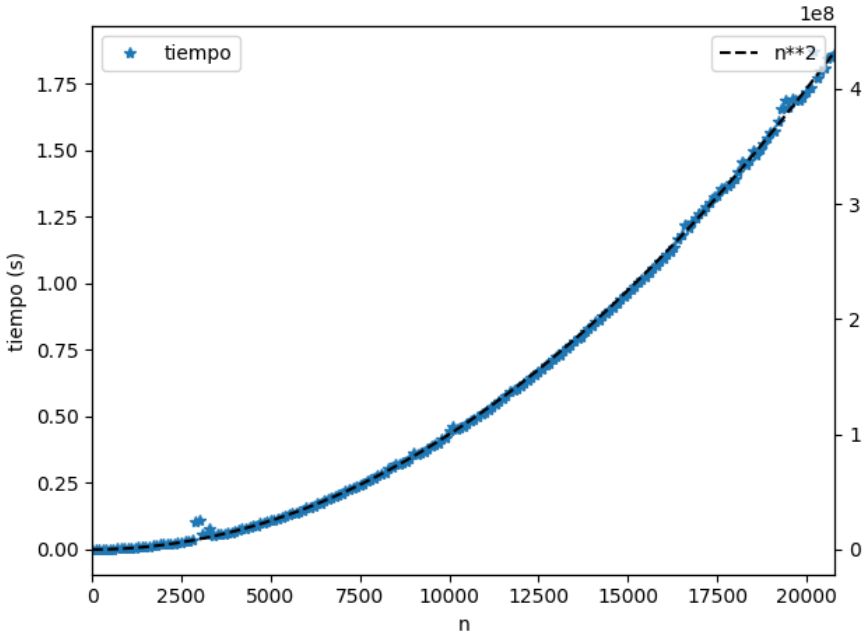
llamamos c a la suma de operaciones $O(1)$. Por cada ciclo for de audiosSoftYHard ($O(n)$) se ejecuta el ciclo for de esHardOSoft ($O(L)$ siendo L la longitud del audio más largo), la complejidad total es $O(n \times L + k)$, simplificando el termino k, queda $O(n \times L)$

```
void reemplazarSubAudio(audio &a, audio a1, audio a2, int profundidad) {
    int indiceDeAparicion = 0;                                              O(1)
    bool pertenece = false;                                                 O(1)
    buscoAparicion(a, a1, pertenece, indiceDeAparicion);                    O(a.size()^2)
    reemplazarAparicion(a, a1, a2, indiceDeAparicion, pertenece);           O(a.size() + a2.size())
}

void buscoAparicion(audio a, audio a1, bool &pertenece, int &indiceDeAparicion) {
    int contador = 0;                                                       O(1)
    for (int i = 0; i < a.size() && !pertenece; i++) {
        if (a[i] == a1[0]) {                                                O(1)
            contador = 1;                                                   O(1)
            for (int j = 1; j < a1.size() && (a[j + i] == a1[j]); j++) {
                contador++;                                                 O(1)
            }                                                               O(a1.size()-1)
            if (contador == a1.size()) {                                    O(1)
                pertenece = true;                                           O(1)
                indiceDeAparicion = i;                                      O(1)
            }
        }
    }                                                                       O((n*(n-1))/2) – n) con n = a.size()
}

void reemplazarAparicion(audio &a, audio a1, audio a2, int indiceDeAparicion, bool pertenece) {

    audio b;
    if (pertenece==true) {                                                  O(1)
        for (int i = 0; i < indiceDeAparicion; ++i) {
            b.push_back(a[i]);                                              O(1)
        }                                                                   O(a.size() - indiceDeAparicion)
        for (int j = 0; j < a2.size(); ++j) {                               O(a2.size())
            b.push_back(a2[j]);                                             O(1)
        }
        for (int k = indiceDeAparicion + a1.size(); k < a.size(); ++k) {
            b.push_back(a[k]);                                              O(1)
        }                                                                   O(a.size() – indiceDeAparicion – a1.size())
        a = b;                                                              O(1)
    }                                                                       O(a.size() + a2.size())
```

El peor caso de esta función, es el caso de que todos los elementos de a sean iguales menos el ultimo, y que a1 sea igual al audio original incluso con el ultimo elemento igual. Ya que en ese caso va a recorrer el audio de la siguiente manera: primero como va a detectar que empieza el audio como a1, entonces lo va a recorrer y cuando llegue a la ultima posición, va a ver que es distinto y saldrá del ciclo. En la siguiente iteración pasará la mismo pero terminara una posición antes. Así hasta la ante ultima posición. De esta manera, siendo a.size() = n, en la primera iteración recorrerá el audio n-1 veces, la segunda n-2, la tercera n – 3, y así hasta n = 1. Esto es la suma de gauss menos el ultimo elemento, quedando así (n*(n-1))/2) – n, llegando así a pertenecer a O(n^2).

Graficos de funciones

`limpiarAudio:`



`revertirAudio:`