UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Estructura de Datos y Análisis de Algoritmos

Laboratorio N°1

Alumno: Israel Arias Panez.

Profesor: Mario Inostroza.
Ayudante: Esteban Silva.

Santiago - Chile
2-2020

TABLA DE CONTENIDOS

Índice de Figuras

CAPÍTULO 1. INTRODUCCIÓN

Una de las muchas definiciones de lo que es un ingeniero y la actividad en la que se desempeña dice “La actividad del ingeniero supone la concreción de una idea en la realidad. Esto quiere decir que, a través de técnicas, diseños y modelos, y con el conocimiento proveniente de las ciencias, la ingeniería puede resolver problemas y satisfacer necesidades humanas.” [1] tomando en cuenta la definición anterior en la asignatura Análisis de algoritmos y estructura de datos se nos presenta el caso del laboratorio n°1 “Salvación espacial”, en el cual se presenta una situación en la cual por motivo de la expansión del covid-19 la Doctora Ximi ha tomado la decisión de emigrar a otro planeta, para ello seleccionara a otras personas que puedan abordar la nave de acuerdo a ciertos criterios medidos en un examen, el cual le otorga una calificación a cada postulante, la doctora trabaja junto a un equipo de reclutamiento llamado ReclutadorAhorro, pero no confía en sus criterios de selección de personal, por ello nos contacta para que elaboremos una solución que permita elegir quienes serán la tripulación que emprenderá el viaje basándonos en la calificación que consiguieron en la prueba aplicada para ello se nos entrega un archivo de texto plano llamado “entrada.in” el cual en su primera línea indica la cantidad de personas que postulan para subir a la nave y a continuación luego de un espacio indica el peso máximo soportado por la nave luego de un salto de línea se ira presentando la información de los postulantes indicando “postulanteX” “peso” “calificación”. En este laboratorio se busca que hagamos la labor de un ingeniero, que apliquemos técnicas, diseños y modelos con el conocimiento proveniente de las ciencias de la computación y de los algoritmos y estructura de datos, para idear una solución, para ello se pondrá en uso lo adquirido en las clases de la asignatura, se hará uso de algoritmos de ordenamiento, estructura de datos como son los arreglos, uso de punteros y direcciones de memoria, etc. También se hará el análisis del algoritmo implementado en base al tiempo y su complejidad.

En esta introducción del informe se ha contextualizado el problema a resolver en este laboratorio, en la próxima sección se hará el análisis del problema, haciendo énfasis en la metodología que se siguió para resolverlo, estructuras de datos usadas, entre otros. Luego en la sección de Algoritmos y estructura de datos se presentarán los algoritmos usados para el desarrollo de la solución, detallando su funcionamiento y tiempo T(n) y O( ). A continuación, en la sección de Análisis de resultados, se describirá los resultados obtenidos de la solución implementada, se analizarán también las falencias detectadas y propuestas de cómo mejorar estas mismas y otros ámbitos de la solución. Finalmente, en la sección de Conclusión se indicará el grado de logro de los objetivos iniciales y una conclusión respecto al trabajo en este laboratorio.

CAPÍTULO 2. DESCRIPCIÓN DE LA SOLUCIÓN

2.1.1 Marco teórico

Para el mejor entendimiento de la solución que será planteada y del informe en general, conviene definir algunos conceptos que serán nombrados.

- Algoritmo: Conjunto de instrucciones o reglas definidas no ambiguas que permite solucionar un problema u otras tareas.
- Metodología: Conjunto de procedimientos racionales para alcanzar un objetivo.
- Archivo: Conjunto de bytes que son almacenados en un dispositivo, por ejemplo, un archivo de texto plano.
- Función: Pequeña parte de un programa que realiza una tarea en particular, recibiendo una entrada, procesándola y devolviendo una salida.
- Arreglo: Conjunto de datos homogéneos que se encuentran ubicados en forma consecutiva en la memoria ram y sirve para almacenar datos en forma temporal.
- Memoria Ram: Memoria principal de la computadora, donde residen programas y datos, sobre la que se pueden efectuar operaciones de lectura y escritura.
- Struct: Declaración de un tipo de dato compuesto que permite almacenar conjuntos de datos en un bloque de memoria simulando una “estructura”.
- Tiempo de ejecución: Es el intervalo de tiempo en el que un programa de computadora se ejecuta en un sistema operativo.
- Complejidad: Es una descripción de la cantidad de tiempo que lleva ejecutar un algoritmo.
- Iteración: Es la acción de repetir varias veces un proceso con la intención de alcanzar una meta u objetivo.
- Estructura de datos: Es una forma particular de organizar datos en una computadora para que puedan ser utilizados de manera eficiente.
- CPU: Es el hardware dentro de un ordenador u otros dispositivos programables, que interpreta las instrucciones de un programa informático mediante la realización de las operaciones básicas aritméticas, lógicas y de entrada/salida del sistema.

2.1.2 Metodología de solución y herramientas utilizadas.

La metodología por seguir para la resolución de este laboratorio es el uso de una programación modular en el lenguaje de programación C o también llamada como “divide y vencerás”, la cual consiste en dividir el problema en subproblemas a fin de ir solucionándolos poco a poco.
Por ende, el programa seguirá las siguientes etapas:
1) Leer el archivo de entrada.
2) Recolectar la información del archivo en arreglos.
3) Generar todas las posibles combinaciones de dos tripulantes que cumplan los requerimientos.
4) Generar todas las posibles combinaciones de tripulantes restantes que cumplan los requerimientos.
5) Generar las combinaciones de tripulantes individuales que cumplan las condiciones.
6) Ordenar todas las combinaciones generadas de mayor a menor según su calificación total.
7) Escribir todas las combinaciones ya ordenadas en un archivo de salida según el formato solicitado.

Para el primer punto se usara la función fopen() con el modo de lectura.
Para el segundo punto se hará uso de la estructura de datos “arrays” o arreglos, se crearán arreglos dinámicos, a los cuales se les asignara memoria usando malloc para que puedan almacenar datos, luego usando un algoritmo iterativo, se ira recorriendo todo el archivo de principio a fin, almacenando todos los datos necesarios, como lo son nombre/identificador del postulante, su peso y calificación, además de la cantidad de postulantes y peso máximo que puede llevar la nave, los valores se conseguirán con el uso de la función fscanf().
Para el tercer, cuarto y quinto punto se hará uso de un struct “combinación” el cual contiene todos los datos que una combinación de tripulantes tiene, por ejemplo, el peso total del tripulante X e Y, y su calificación total, la cual es la suma de su calificación individual de ambos, para crear las combinaciones se usaran múltiples algoritmos iterativos.
Para el quinto punto se hará uso del algoritmo iterativo visto en catedra “Bubble Sort”, sin embargo, fue ligeramente modificado para que ordenara de mayor a menor, en base a la calificación de los tripulantes.
Para el sexto punto se hará uso de algoritmos iterativos, además se debe velar respetar la estructura del formato de escritura del archivo de salida, el cual se puede observar en la figura 1.

```
postulante2, postulante3, 176, 28
postulante1, postulante2, 189, 25
postulante1, postulante3, 183, 23
postulante2 91 15
postulante3 85 13
postulante1 98 10
```

Figura 1: Ejemplo formato de salida.

2.1.3 Algoritmos y estructuras de datos

Empezando por el punto uno de la metodología de solución presentada en el punto 2.1.2 "Leer el archivo de entrada". Para eso se ha implementado la siguiente función:

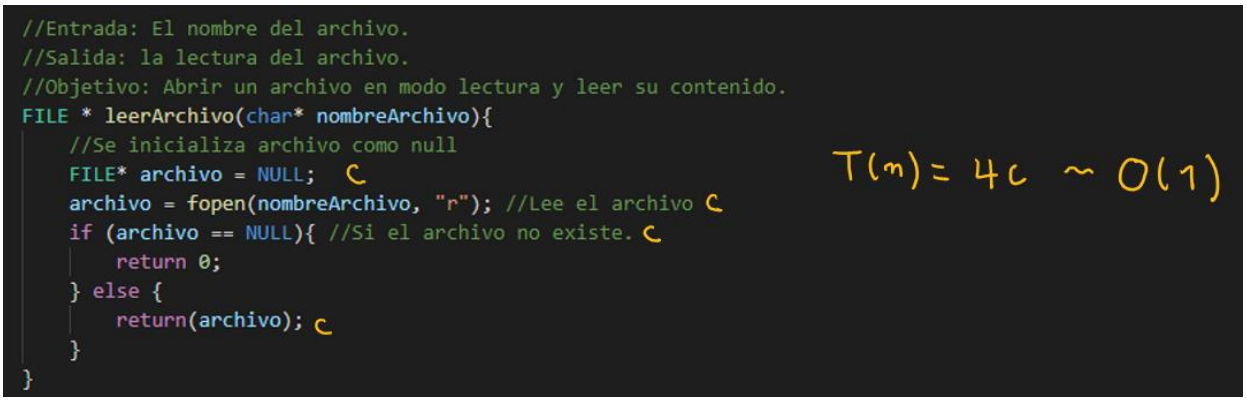

Figura 2: Función leerArchivo.

La función leerArchivo tiene como misión el abrir un archivo en modo lectura para poder leer su contenido, recibe como entrada el nombre del archivo y su salida en caso de que se haya encontrado el archivo por su nombre es el resultado de la función fopen(), en caso contrario retorna un cero.
La función tiene un T(n) = c + c +c +c -> t(n) = 4c ~ O(1).
La función siempre se llama con el nombre de entrada "entrada.in", ya que es lo que se solicita en el planteamiento del laboratorio.

Para el punto número dos de la metodología, se hizo uso del siguiente algoritmo iterativo:

```
//Se crea Una variable iteradora
int i = 0;
//Ahora se ejecutara un ciclo para guardar todo el documento en los 4 arreglos creados anteriormente
for(i = 0; i < cantidadPostulantes; i++){
    //Se lee y almacena el numero del tripulante en el arreglo.
    fscanf(archivo,"%s",&nombreTripulante);
    strcpy(tripulantes[i], nombreTripulante);
    //Se lee y almacena el peso del tripulante en el arreglo.
    fscanf(archivo,"%d",&pesoTripulante);
    pesoTripulantes[i] = pesoTripulante;
    //Se lee y almacena la calificacion del tripulante en el arreglo.
    fscanf(archivo,"%d",&puntuacionTripulante);
    calificacionTripulantes[i] = puntuacionTripulante;
    //Finalmente se agrega el id del tripulante al arreglo, siendo este la posicion dentro del arreglo
    idTripulantes[i] = i;
}
//Una vez almacenada toda la informacion que contiene el archivo, se cierra ya que no se ocupara más
fclose(archivo);
```

T(n) = c + 8cn + c → T(n) = 2c + 8cn ~ O(n)

Figura 3: Algoritmo para guardar la información.

El algoritmo recorre todo el documento haciendo uso de fscanf() para leer la información dentro del archivo, para el caso de los nombres además se hace uso de strcpy() para almacenar en el arreglo que contiene los nombres de los postulantes se almacenan los datos en cuatro arreglos: tripulantes, pesoTripulantes, calificacionTripulantes, idTripulantes, el algoritmo tiene una complejidad de O(n). Una vez almacenada toda la información necesaria, se cierra el documento abierto.

Para el tercer punto presentado en la metodología fue implementada la struct combinaciones

```
typedef struct combinacion combinacion;

struct combinacion{
    int cantidadElementos;
    int pesoTotal;
    int calificacionTotal;
    int *arregloCombinacion;
};
```

Figura 4: Struct combinación.

La struct en si es una combinación de distintos postulantes, la cantidadElementos refleja de cuantos tripulantes es la combinación, pesoTotal refleja la suma de todos los pesos de los tripulantes de la combinación, calificacionTotal es la sumatoria de todas las calificaciones de los tripulantes que componen la combinación y por ultimo arregloCombinación es un arreglo el cual contendrá todos los id de los tripulantes que componen la combinación, para así saber quiénes componen esta combinación, por ejemplo una struct compuesta por el tripulante 1 y el tripulante 2 tendría como valor cantidadElementos = 2, pesoTotal = pesoTripulante1 + pesoTripulante2, calificacionTotal = calificacionTripulante1 + calificacionTripulante2, arregloCombinacion = {0 ,1}.

La struct antes presentada servirá para ir creando las distintas combinaciones validas que se

vayan encontrando, ahora para encontrar las combinaciones de 2 tripulantes fue implementado el siguiente algoritmo:

```
//Tambien se creara un arreglo que contendra "Combinaciones"
int cantidadCombinaciones = 0;
//Se le asigna memoria y se crea
combinacion *combinaciones = (combinacion *)malloc(sizeof(combinacion)*cantidadCombinaciones);
// A continuacion se efectuara la combinatoria para ir creando las distintas combinaciones de tripulantes que puedan abordar
// Sin embargo esto agregara solo todas las combinaciones de 2 tripulantes al arreglo combinaciones
for (int i = 0; i < cantidadPostulantes; i++){
    for(int j = i+1; j < cantidadPostulantes; j++){
        if(pesoTripulantes[i]+pesoTripulantes[j] <= pesoMaximo){
            int * arregloCombinacion = malloc(sizeof(int)*2);
            //Se agregan los id de los tripulantes
            arregloCombinacion[0] = i;
            arregloCombinacion[1] = j;
            combinacion nuevaCombinacion = crearCombinacion(2,arregloCombinacion,pesoTripulantes[i]+pesoTripulantes[j],calificacionTripulantes[i]+calificacionTripulantes[j]);
            //Se agrega la combinacion encontrada al arreglo de combinaciones
            combinaciones = agregarCombinacion(combinaciones,&cantidadCombinaciones,nuevaCombinacion);
            free(arregloCombinacion);
        }
    }
}
```

Figura 5: Algoritmo generador de combinaciones duales.

El algoritmo como se puede ver funciona con un arreglo que ira almacenando combinaciones que usan la struct anteriormente presentada, almacenando las combinaciones en un arreglo llamado combinaciones.
El algoritmo funciona de la siguiente manera: se recorrerán los arreglos creados anteriormente que contienen toda la información de los tripulantes, con dos ciclos for anidados, que tomaran por así decirlo un tripulante y lo comparara al resto de los tripulantes, se verificara que la suma del peso de ambos no supere el máximo pero permitido por la nave, en caso de que no superen el peso máximo se creara una nueva combinación (struct) en la cual se almacenara todos los datos correspondientes a esa combinación entre esos dos tripulantes, para ello se a implementado otra función llamada crearCombinacion(), finalmente la combinación creada se agregara al arreglo de combinaciones haciendo uso de otra función implementada llamada agregarCombinacion(). Al final de estos dos ciclos anidados, el arreglo combinaciones contendrá todas las combinaciones posibles validas de dos tripulantes presentes en el archivo de entrada.

La función crearCombinación, usada en el algoritmo se detalla a continuación:

```
//Entrada: La cantidad de elementos de la nueva combinacion, el arreglo combinacion, el peso total de  la combinacion, la calificacion total.
//Salida: una combinacion.
//Objetivo: Crea una nueva combinacion con todos los elementos de la struct combinacion
combinacion crearCombinacion(int cantidadElementos,int * arregloCombinacion,int pesoTotal,int calificacionTotal){
    combinacion nuevaCombinacion; c
    nuevaCombinacion.cantidadElementos = cantidadElementos; c
    nuevaCombinacion.pesoTotal = pesoTotal; c
    nuevaCombinacion.calificacionTotal = calificacionTotal; c
    nuevaCombinacion.arregloCombinacion = malloc(sizeof(int)*cantidadElementos); c
    for(int i = 0; i < cantidadElementos; i++){
        nuevaCombinacion.arregloCombinacion[i] = arregloCombinacion[i];   c c  2cm
    }
    return (nuevaCombinacion);   c
}
T(m) = 4c + 2cm + c → T(m) = 5c + 2cm ~ O(m)
```

Figura 6: Función crearCombinacion.

La función tiene como objetivo crear una nueva combinación, para ello recibe como entrada la cantidad de elementos que es la cantidad de tripulantes de la cual se compone la combinación a crear, también recibe el arreglo con los id de los tripulantes, recibe también el peso total de los tripulantes y su calificaciónTotal, hace uso de la struct combinación, agrega todos los elementos a una nuevaCombinación y finalmente con un ciclo for copia el arreglo

con el id de los tripulantes dentro de la nuevaCombinación, finalmente retorna la nueva combinacion creada. El algoritmo tiene un T(n) = 5c+2n y un O(n).

La función agregarCombinación usada en el algoritmo se detalla a continuación:

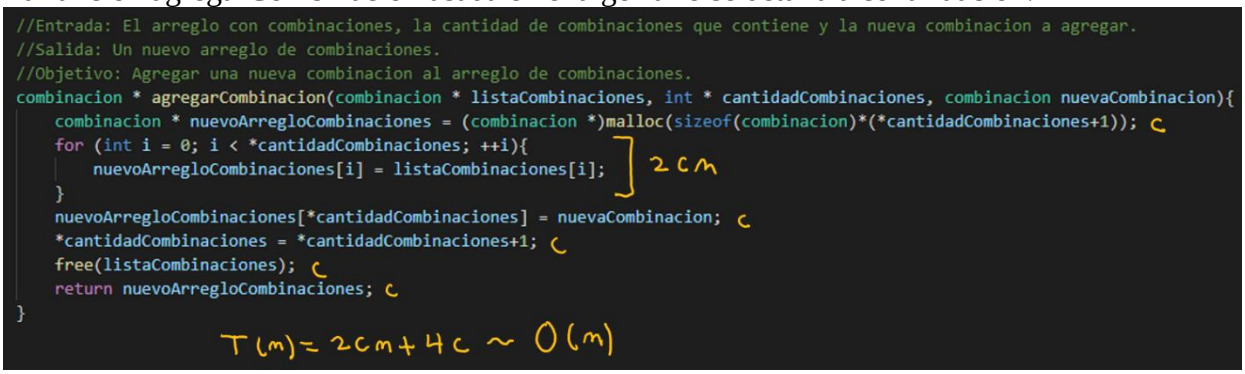

Figura 7: Función agregarCombinacion.

La función tiene como objetivo agregar una combinación al arreglo de combinaciones, para esto recibe como entrada el arreglo que contiene todas las combinaciones, la cantidad de combinaciones que ya tiene dentro y por último la combinación que se quiere agregar, su funcionamiento es crear un nuevo arreglo y asignarle memoria a la misma cantidad de elementos que ya tenía +1. Luego se copiara todas las combinaciones ya existentes a este nuevo arreglo, y finalmente se agregara la nueva combinación al final del arreglo, se liberará la memoria del arreglo antiguo y se retornará el nuevo arreglo actualizado, también se hace uso de punteros para actualizar desde dentro de la función la cantidad de combinaciones.
La función tiene un tiempo T(n) = 2cn + 4c y un O(n).

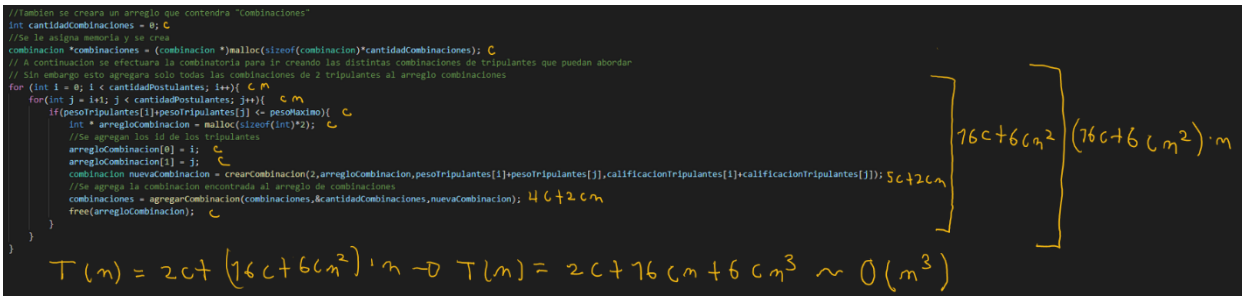

Figura 8: T(n) y O() del algoritmo generador de combinaciones duales.

Finalmente, el algoritmo que genera todas las posibles combinaciones de 2 tripulantes tiene un T(n) = $2c+16cn+6cn^3$ lo cual se puede apreciar en las funciones, ya que tiene un doble for anidado el cual se transforma en tres for anidados por los ciclos for que usan las funciones crearCombinación y agregarCombinación.

Luego para el cuarto punto presentado en la metodología “Generar todas las posibles combinaciones de tripulantes restantes que cumplan los requerimientos” se implementó el siguiente algoritmo:

```
//Ahora para añadir las combinaciones restantes se iterara sobre la cantidad de combinaciones
//Se ira viendo combinacion por combinacion de la lista de combinaciones
//el ciclo se rompera cuando se hayan recorrido todas las combinaciones y no sea posible crear mas, incluso las que este mismo ciclo crea
for(int  i = 0; i < cantidadCombinaciones; i++){
    //Se vera si la combinacion seleccionada puede agregar a algun tripulante mas dentro de ella, para eso se recorre todo el arreglo de tripulantes
    for(int j = 0; j < cantidadPostulantes; j ++){
        int sePuedeCombinar = esPosibleCombinar(combinaciones[i],pesoTripulantes[j],idTripulantes[j],pesoMaximo);
        //Si es posible combinar
        if(sePuedeCombinar == 1){
            //Se crea la combinacion con el tripulante añadido a la antigua combinacion
            combinacion nuevaCombinacion = juntarCombinaciones(combinaciones[i],pesoTripulantes[j],idTripulantes[j],calificacionTripulantes[j]);
            //Luego se verifica si la combinacion creada no se encuentra dentro del arreglo de combinaciones
            if(estaCombinacion(combinaciones,cantidadCombinaciones,nuevaCombinacion) == 0){
                //En caso de no estarlo,se actualiza el arreglo de combinaciones
                combinaciones = agregarCombinacion(combinaciones,&cantidadCombinaciones,nuevaCombinacion);
            }
        }
    }
}
```

Figura 9: Algoritmo generador de combinaciones.

El algoritmo necesita como funcionamiento el que existan combinaciones previas ya generadas, por eso primero se implemento un algoritmo que genere todas las combinaciones de dos elementos, el algoritmo implementa la siguiente lógica:

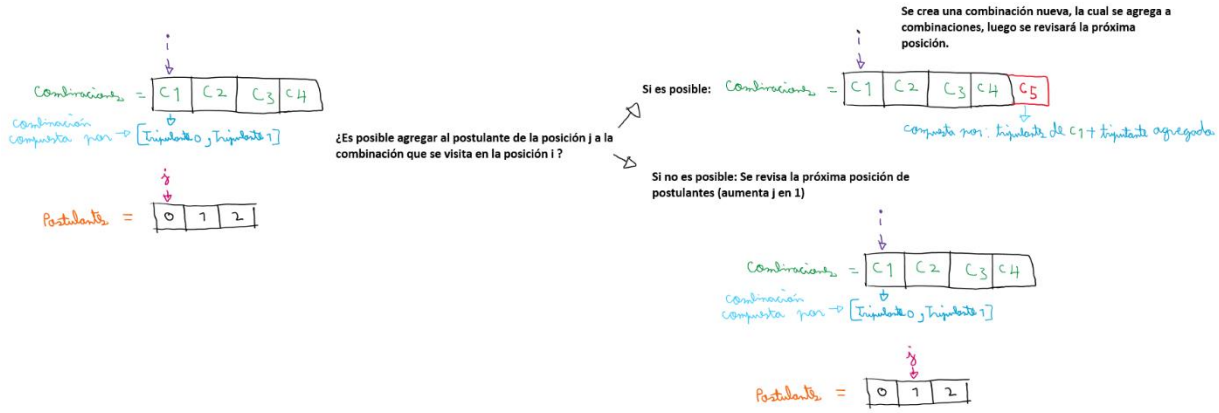

Figura 10: Lógica del algoritmo generador de combinaciones restantes.

El algoritmo ira iterando sobre el arreglo de combinaciones ya generadas, revisara por ejemplo la primera combinación, revisará de que postulantes esta generada esa combinación, luego revisará el arreglo de postulantes, y chequeara si es posible incorporar al postulante[j] dentro de la combinación que se esta revisando, en caso de que eso sea posible, crea una nueva combinación la cual se agrega al final del arreglo de combinaciones, la cual por ejemplo si la combinación c1 estaba formada de los tripulantes 0 y 1, pero a esta combinación puede entrar el tripulante 2, luego de haber verificado el peso de la nave y de los tripulantes, se crea una nueva combinación por ejemplo la combinación 5 que se ve en la figura 10, la cual se añade al final del arreglo combinación, la combinación 5 entonces esta formada por el tripulante 0, 1 y 2. En caso de no ser posible agregar al tripulante, se revisará la siguiente posición de postulantes (j+1), se hará esta verificación para cada combinación la

cual verificará si puede añadir a cada postulante dentro de ella, para evitar combinaciones repetidas, se verificará si la combinación que se quiere agregar ya se encuentra o no dentro del arreglo de combinaciones, para ello se implementó la función estaCombinación(), para verificar si es posible generar una nueva combinación se creó el algoritmo esPosibleCombinar(), para crear la nueva combinación entre una combinación y un tripulante se creo la función juntarCombinaciones() y finalmente para agregar la nueva combinación al arreglo se hace uso de la función presentada anteriormente agregarCombinacion().
La función esPosibleCombinar() es la siguiente:

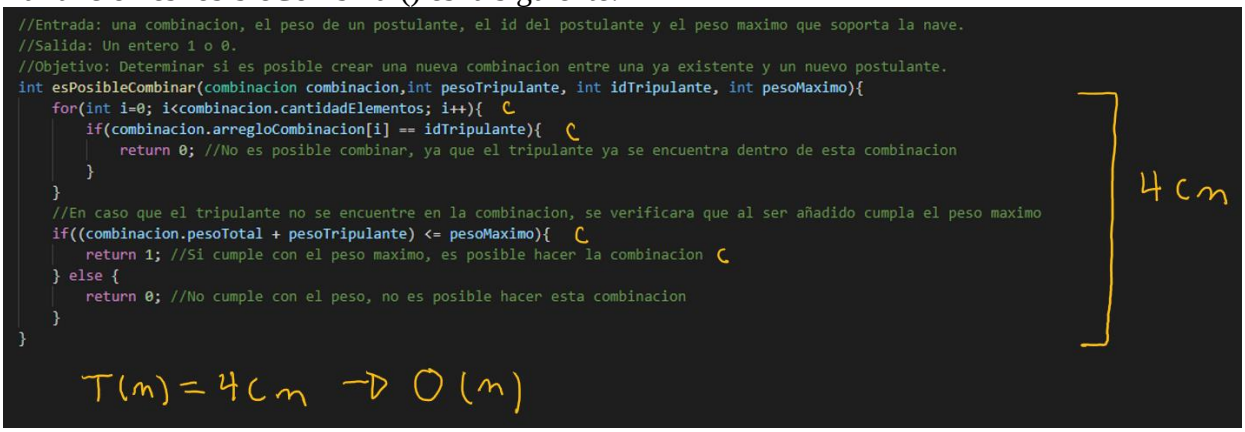

Figura 11: Función esPosibleCombinar().

La función tiene como objetivo el determinar si es posible crear una combinación entre una combinación ya existente y un tripulante de la lista de tripulantes, para ello en primer lugar verifica si el tripulante que se quiere agregar se encuentra o no dentro de la combinación, en caso de estarlo retorna un 0, lo que significa que no es posible, en caso que no se encuentre dentro de la combinación, se verificara si al agregar a la combinación la suma del peso de todos los tripulantes no sumen más que el limite del peso de la nave, en caso de que no lo superen retorna 1, lo que significa que es posible generar la combinación entre la combinación previa y el tripulante a sumar. En caso de que, si se supere el peso máximo, retorna 0. La función tiene un T(n) = 4cn y un O(n).

La función juntarCombinaciones() es la siguiente:

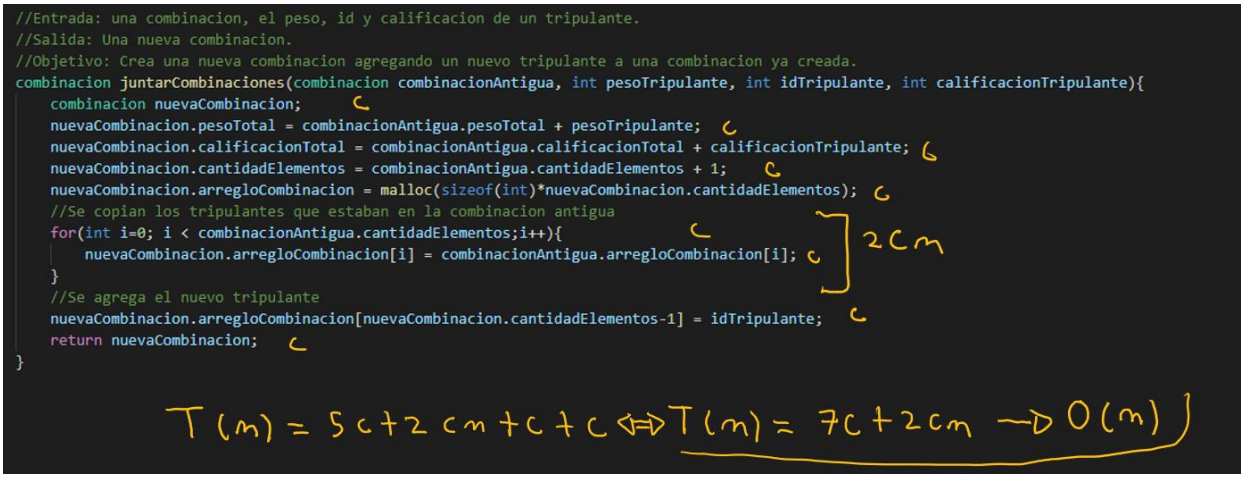

Figura 12: Función juntarCombinaciones()

La función tiene como objetivo el generar una nueva combinación, tomando como base una combinación ya creada dentro del arreglo y un tripulante que se "unirá" a esa combinación, en resumen, se crea una nueva combinación que contiene toda la información de los tripulantes de la combinación antigua a la cual se le ingresa el nuevo tripulante, retornando así una nueva combinación tiene como entradas una combinación y los datos del tripulante (peso, id calificación) que entrará. La función tiene un T(n) = 7c +2cn y un O(n).

Luego nos encontramos con la función verificadora estaCombinacion():

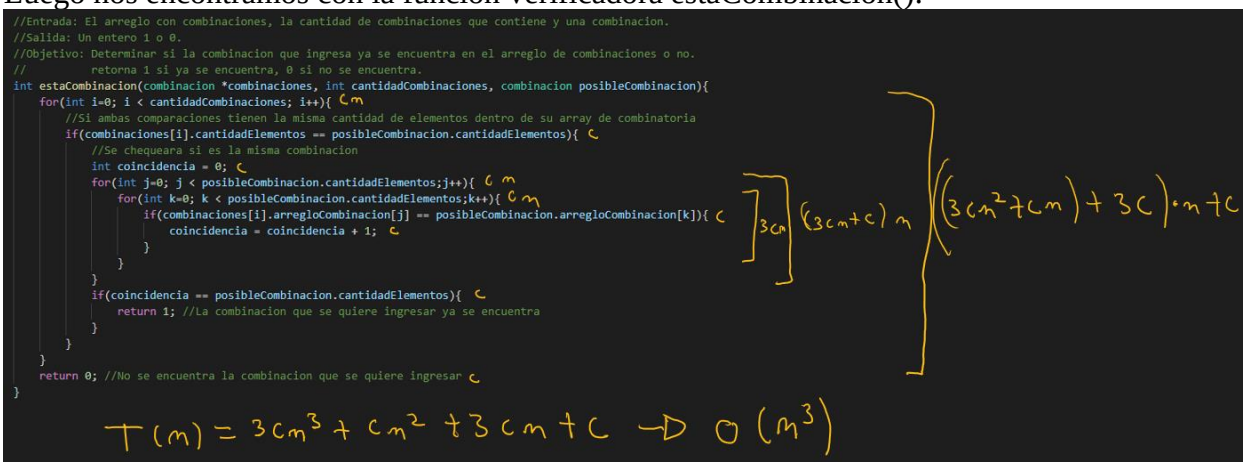

Figura 13: Función estaCombinacion().

Esta función tiene como objetivo el verificar si una combinación ya se encuentra dentro del arreglo de combinaciones o no. Para ello recibe como objetivo el arreglo de combinaciones, la cantidad de combinaciones que tiene el arreglo y la combinación que se quiere comprobar. Para realizar la comprobación se hará uso de tres ciclos for anidados, en primer lugar se verificara si la combinación revisada del arreglo tiene la misma cantidad de elementos(tripulantes) que la combinación ingresada a verificar, en caso de que sea la misma se procederá a verificar con un doble ciclo for anidado si cada tripulante presente en una

combinación se encuentra en la otra combinación, en caso de que todos los tripulantes sean los mismos en ambas partes, se retornara un uno, lo que significa que la combinación ya se encuentra en el arreglo de combinaciones, en caso de que no sea el caso retornara un cero, lo que significa que la combinación no se encuentra en el arreglo de combinaciones.

Finalmente, el algoritmo del punto 4 de la metodología presentada, considerando las funciones presentadas:

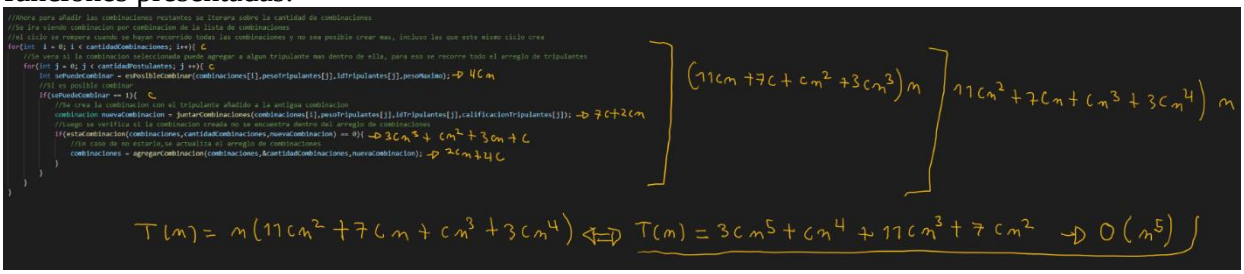

Figura 14: T(n) y O() del algoritmo para el cuarto punto de la metodología.

El algoritmo tiene un $T(n) = 3cn^5+cn^4+11cn^3+7cn^2$ y un $O(n^5)$, lo que lo hace un algoritmo que consume una gran cantidad de recursos y memoria del computador.

Hasta ahora ya se tienen todas las combinaciones, sin embargo, faltan las combinaciones individuales de tripulantes, por lo tanto, para el punto 5 de la metodología se hace uso del siguiente algoritmo:

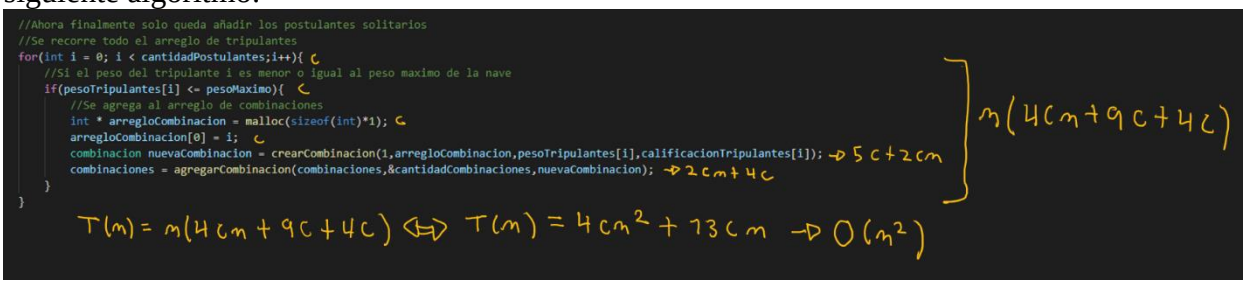

Figura 15: Algoritmo para agregar combinaciones individuales.

El algoritmo hace uso de funciones ya implementadas, funciona iterando sobre el arreglo de postulantes, determinando si están bajo el peso máximo o no, en caso de estar bajo el peso máximo, llama a la función crearCombinacion() y por ultimo lo agrega al arreglo de combinaciones haciendo uso de la función agregarCombinacion(). El algoritmo tiene un $T(n) = 4cn^2+13cn$ y un $O(n^2)$.

Una vez cumplidos los puntos uno, dos, tres, cuatro y cinco. El arreglo de combinación tiene absolutamente todas las combinaciones validas de posible tripulación que puede abordar la nave. Por ende, ahora se procederá al punto seis de la metodología propuesta en el punto 2.1.2 del informe “Ordenar las combinaciones de acuerdo a su calificación”, para ello se hará uso de un algoritmo visto en Catedra “Bubble Sort”:

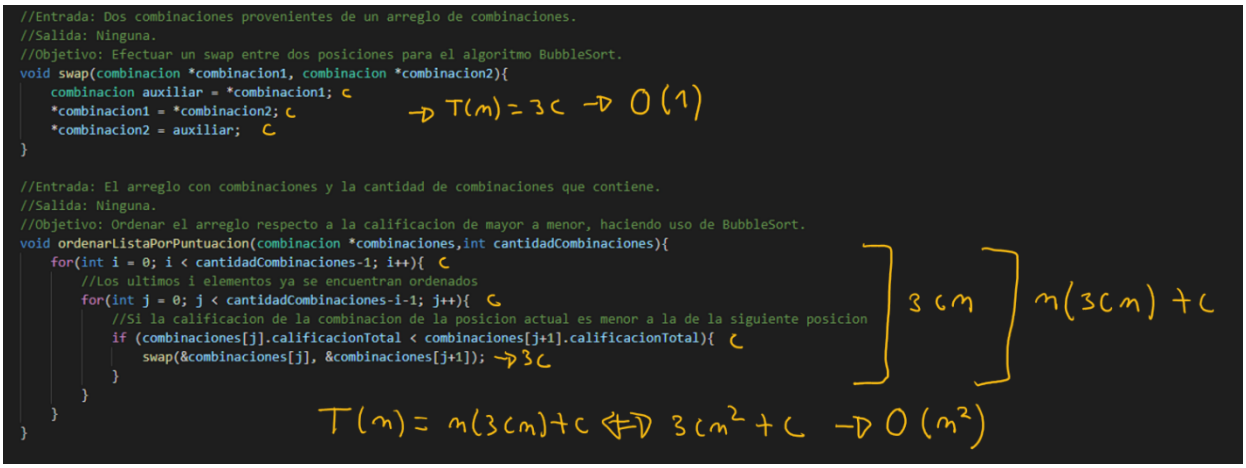

Figura 16: Bubble Sort.

El Bubble sort es un algoritmo de ordenamiento, que funciona comparando una posición junto a la posición que le sigue, en este caso verifica si la posición actual es menor a la siguiente posición, en caso de ser menor las intercambia, llamando a la función swap, de esa manera al terminar los dos ciclos for anidados, se tendrá una lista ordenada de mayor a menor. En este caso la función bubble sort recibirá como entrada la lista de combinaciones y la lista de combinaciones, y hará la comparación comparando la calificación total de la combinación a la combinación de la posición siguiente, de esa manera hará los intercambios cuando la calificación actual de la combinación de la posición actual sea menor a la siguiente, de esa manera al finalizar el algoritmo, el arreglo de combinaciones se encontrara ordenado desde la combinación que tiene la mayor calificación hasta la combinación que tiene la calificación más baja, no existen retornos ya que modifica directamente el arreglo mediante punteros.

Finalmente, para llevar a cabo el punto 7 de la metodología propuesta, se hace uso del siguiente algoritmo:

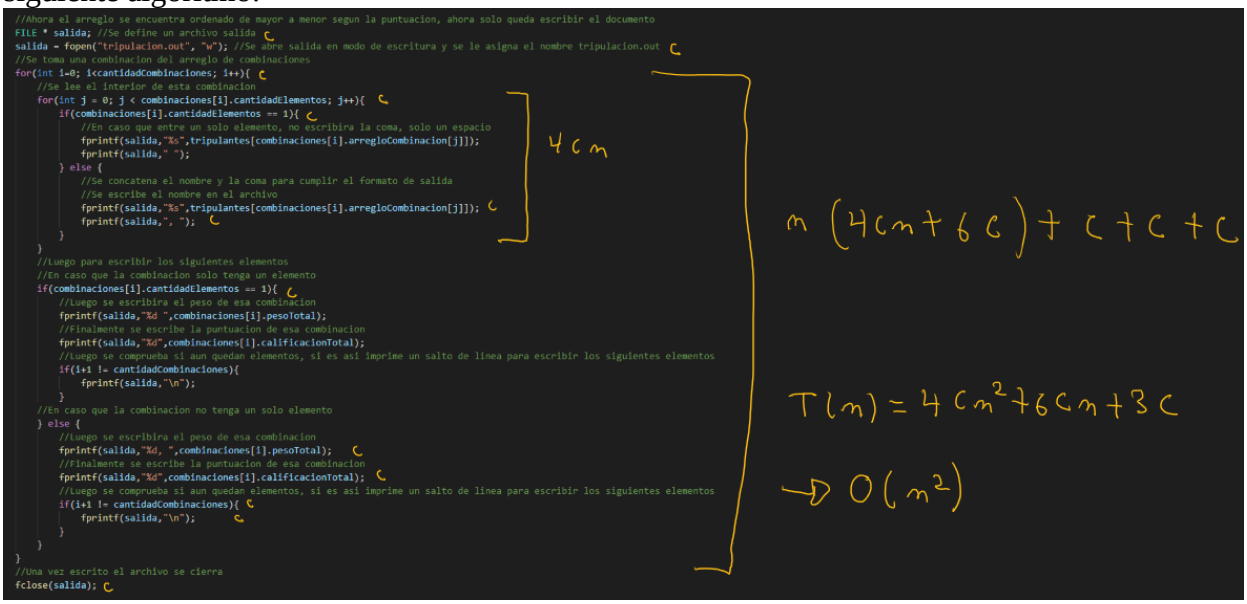

Figura 17: Algoritmo de escritura de archivo.

El algoritmo leerá iterativamente el arreglo de combinaciones, tomando cada elemento desde el principio al final y los escribirá en un archivo de salida llamado “tripulación.out”, el cual será creado por la función fopen() con modo escritura, luego se escribirá en el documento creado usando la función fprintf(), también se verificara dentro del algoritmo los casos cuando se esté escribiendo una combinación de un solo tripulante, ya que en estos casos según el formato solicitado, no debe haber comas que separen el nombre de su peso y puntuación, también se verificara cuando quede una sola combinación, para no escribir un salto de línea innecesario finalmente cierra el documento al terminar de escribir todos los datos del arreglo de combinaciones. El algoritmo de escritura de archivo tiene un $T(n) = 4cn^2+6cn+3c$ y un $O(n^2)$. Con esto se terminaría toda la subdivisión de problemas propuesta con la cual se resolvió el problema propuesto en este laboratorio.

2.2 Análisis de los resultados

En distintos casos probados el programa funciona de buena manera y cumple con la generación archivo de salida especificado, a continuación, un ejemplo.

```
postulante1, postulante2, postulante4, postulante5, 530, 285
postulante1, postulante3, postulante4, 770, 258
postulante2, postulante3, postulante4, 740, 243
postulante1, postulante3, postulante5, 790, 238
postulante1, postulante2, postulante4, 440, 235
postulante2, postulante3, postulante5, 760, 223
postulante3, postulante4, postulante5, 660, 218
postulante1, postulante2, postulante5, 460, 215
postulante1, postulante4, postulante5, 360, 210
postulante2, postulante4, postulante5, 330, 195
postulante1, postulante3, 700, 188
postulante2, postulante3, 670, 173
postulante3, postulante4, 570, 168
postulante1, postulante2, 370, 165
postulante1, postulante4, 270, 160
postulante3, postulante5, 590, 148
postulante2, postulante4, 240, 145
postulante1, postulante5, 290, 140
postulante2, postulante5, 260, 125
postulante4, postulante5, 160, 120
postulante3 500 98
postulante1 200 90
postulante2 170 75
postulante4 70 70
postulante5 90 50
```

Figura 18: Ejemplo de salida del archivo de prueba de 5 tripulantes.

Respecto a los tiempos de ejecución, tomando en cuenta todos los algoritmos presentados en la sección 2.1.3 el programa en su completitud tendría una complejidad $O(n^5)$, que es la complejidad más alta de todos los algoritmos implementados, la cual definitivamente podría ser más eficiente y mejorarse, ya sea con el refinamiento de los mismos algoritmos implementados, o el uso de distintos algoritmos y estructuras de datos. Considerando la

complejidad del programa, considero esa la principal falencia de la solución implementada, debido a los altos tiempos de ejecución que podrían presentarse con grandes cantidades de datos. Haciendo uso de la librería time y la función clock_t se ha medido el tiempo para el archivo de ejemplo disponible en uvirtual el cual contempla cinco tripulantes, se obtuvo el siguiente tiempo de ejecución real:

```
C:\Users\Israel\Desktop\Codigos\C\EDA\Lab1>lab1.exe
Archivo de salida generado !
Tiempo transcurrido: 0.019000
C:\Users\Israel\Desktop\Codigos\C\EDA\Lab1>
```

Figura 19: Tiempo real de ejecución.

El tiempo es respecto al tiempo en el cual la CPU del equipo estuvo ocupado ejecutando el programa, por ende, este tiempo puede ser mayor o menor dependiendo de las características de cada equipo.

Para el siguiente archivo de prueba, de 25 tripulantes:

```
25 256
postulante1 98 08
postulante2 91 15
postulante3 85 13
postulante4 98 17
postulante5 91 15
postulante6 85 13
postulante7 98 24
postulante8 91 15
postulante9 85 13
postulante10 98 10
postulante11 91 15
postulante12 85 13
postulante13 72 10
postulante14 91 15
postulante15 85 13
postulante16 98 23
postulante17 91 15
postulante18 85 13
postulante19 98 10
postulante20 40 15
postulante21 85 13
postulante22 98 16
postulante23 91 15
postulante24 85 13
postulante25 85 13
```

Figura 20: Archivo de prueba.

El tiempo de ejecución fue de 0.044 segundos, medido al igual que el ejemplo anterior.

Al ejecutar distintas mediciones con archivos de prueba con el mismo formato del de la figura 20, se obtuvieron los siguientes resultados:

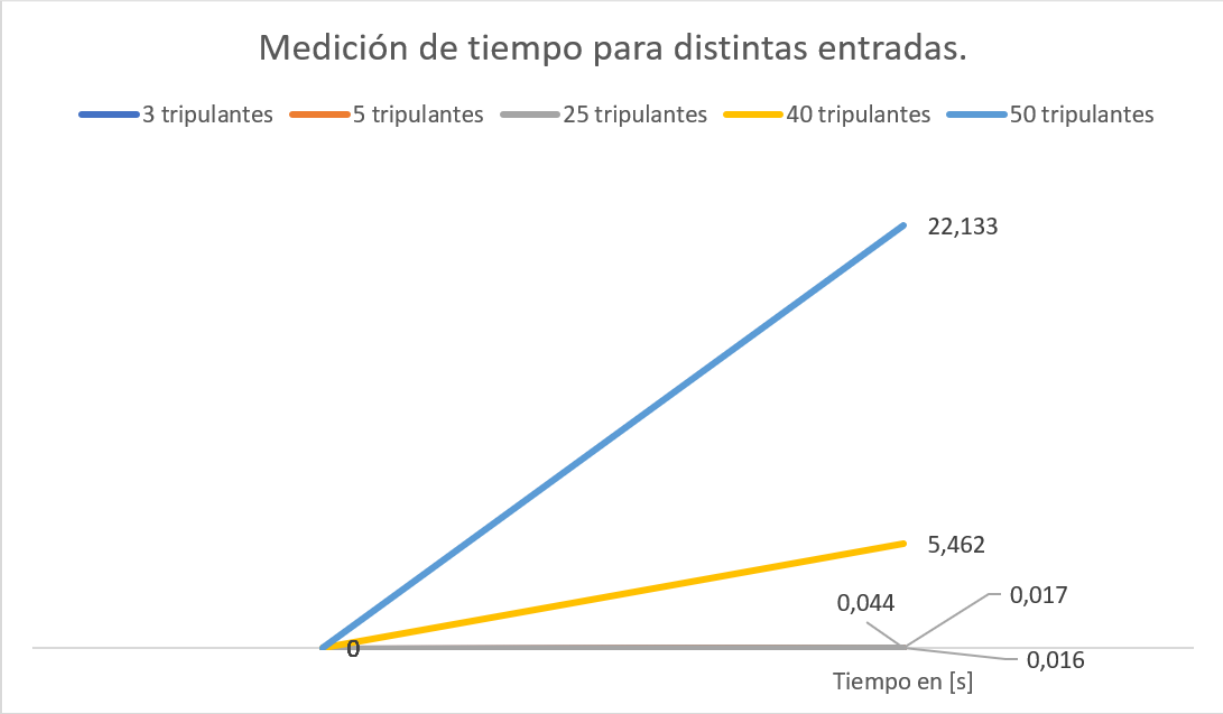

Figura 21: Grafico de mediciones de tiempo.

Haciendo un análisis del grafico y considerando que el orden del algoritmo del programa calculado es de $O(n^5)$, es posible determinar que efectivamente existe una fuerte correlación del tiempo de ejecución real con el tiempo de ejecución teórico calculado y como se había inferido anteriormente, el algoritmo es poco efectivo mientras más cantidad de datos deba analizar.

2.3 Conclusión

Para concluir, se logro a cabalidad el objetivo principal de este laboratorio, que era el crear una solución que lograra generar las posibles tripulaciones que abordaran la nave y clasificarlas de acuerdo con su calificación en la prueba. Sin embargo, la solución creada presenta el problema que se aprecia en el gráfico de la figura 21 del punto anterior: es ineficiente para cantidades de datos grandes, por ende, si se intentara generar una tripulación de por ejemplo 500 personas, el algoritmo tomaría una enorme cantidad de tiempo en lograr generar las distintas combinaciones de tripulación. La principal mejora que se le puede hacer al programa creado es el de optimizar mejor los algoritmos usados para poder trabajar con grandes cantidades de datos sin que el orden del algoritmo sea de $O(n^5)$ y por tanto tenga altos tiempos de ejecución.

Para finalizar, esta experiencia de trabajo de laboratorio sirvió para darse cuenta de la importancia de los distintos algoritmos y estructura de datos, en especial cuando fue necesario el determinar el orden del algoritmo creado y medir el tiempo de ejecución, es muy probable que por ejemplo con el uso de una estructura de datos como lo son las listas enlazadas y algoritmos de búsqueda optimizados como es el quickSort o Heapsort, sea posible trabajar de manera más fácil problemas como este laboratorio o cualquier otro desafío que se presente.

2.4 Referencias

[1] Pérez J y Merino M. (2009). *Definición de ingeniería.* (Recuperado 08/11/2020). https://definicion.de/ingenieria/