

**INSTITUTO POLITÉCNICO NACIONAL**

**ESCUELA SUPERIOR DE CÓMPUTO**

**ANÁLISIS DE ALGORITMOS**

**PROFESORA: LUZ MARÍA SÁNCHEZ GARCÍA**

**ALUMNO: VÁZQUEZ MORENO MARCOS OSWALDO**

**2016601777**

**PRÁCTICA 1 ANÁLISIS TEMPORAL DE UNA BÚSQUEDA LÍNEAL**

**3CM2**

**23 DE FEBRERO DE 2019**

**Introducción**

A continuación, se presenta un reporte de la práctica número 2 la cual consiste en recorrer y examinar cada uno de los elementos del arreglo o en este caso de tres estructuras distintas, hasta encontrar el o los elementos buscados, o hasta que se han mirado todos los elementos de las estructuras.

Este es el método de búsqueda más lento, pero si nuestra información se encuentra completamente desordenada es el único que nos podrá ayudar a encontrar el dato que buscamos. El siguiente algoritmo ilustra un esquema de implementación del algoritmo de búsqueda secuencial:

**for(i=j=0;i<N;i++)  
  if(array[i]==elemento)  
   {  
     solucion[j]=i;  
     j++;  
   }**

Existen distintas complejidades de la búsqueda lineal

(A) MEJOR CASO: El algoritmo de búsqueda lineal termina tan pronto como encuentra el elemento buscado en el array. Si tenemos suerte, puede ser que la primera posición examinada contenga el elemento que buscamos, en cuyo caso el algoritmo informará que tuvo éxito después de una sola comparación. Por tanto, la complejidad en este caso será O(1).

(B) PEOR CASO: Sucede cuando encontramos X en la última posición del array. Como se requieren n ejecuciones del bucle mientras, la cantidad de tiempo es proporcional a la longitud del array n, más un cierto tiempo para realizar las instrucciones del bucle mientras y para la llamada al método. Por lo tanto, la cantidad de tiempo es de la forma an + b (instrucciones del mientras \* tamaño del arreglo + llamada al método) para ciertas constantes a y b, que representan el coste del bucle mientras y el costo de llamar el método respectivamente. Representando esto en notación O, O(an+b) = O(n).

(C) CASO MEDIO: Supongamos que cada elemento almacenado en el array es igualmente probable de ser buscado. La media puede calcularse tomando el tiempo total de encontrar todos los elementos y dividiéndolo por n:

Total = a (1 + 2 + ...+n) + bn = a (n(n+1) / 2) + bn, a representa el costo constante asociado a la ejecución del ciclo y b el costo constante asociado a la evaluación de la condición. 1, 2 , ..n, representan el costo de encontrar el elemento en la primera, segunda, ..., enesima posición dentro del arreglo.  
Media = (Total / n) = a((n+1) / 2) + b que es O(n).

Este es el algoritmo de más simple implementación, pero no el más efectivo. En el peor de los casos se recorre el array completo y el valor no se encuentra o se recorre el array completo si el valor buscado está en la última posición del array. La ventaja es su implementación sencilla y rápida, la desventaja, su ineficiencia. (Díaz, 2006)

**Planteamiento del problema**

Hacer un programa que realice la búsqueda lineal de un elemento dentro de 3 estructuras de datos distintas considerando que los n datos no se encuentran ordenados (peor caso o caso promedio).

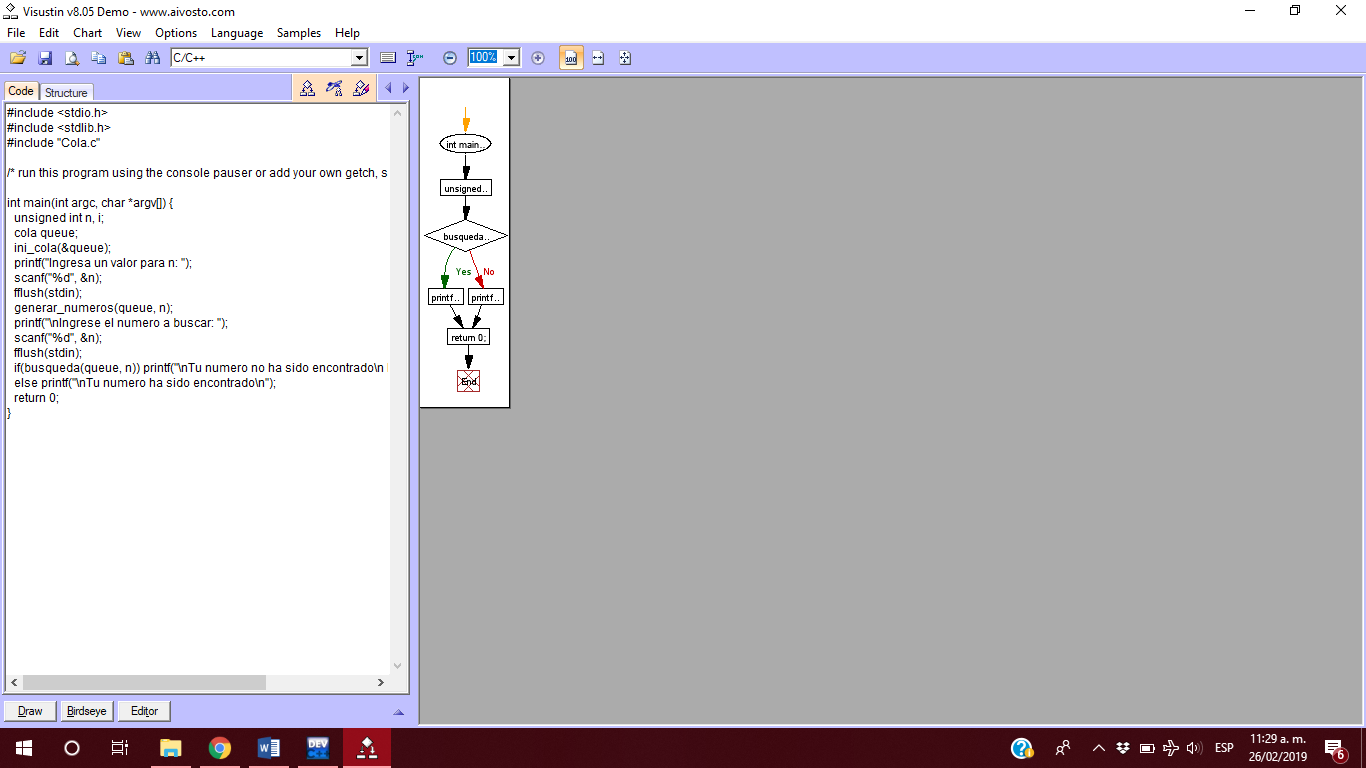
Diseñar los algoritmos para calcular de cada una de las estructuras de datos:

* La función de complejidad temporal: *f(n).*
* La función de complejidad espacial: *f(n).*

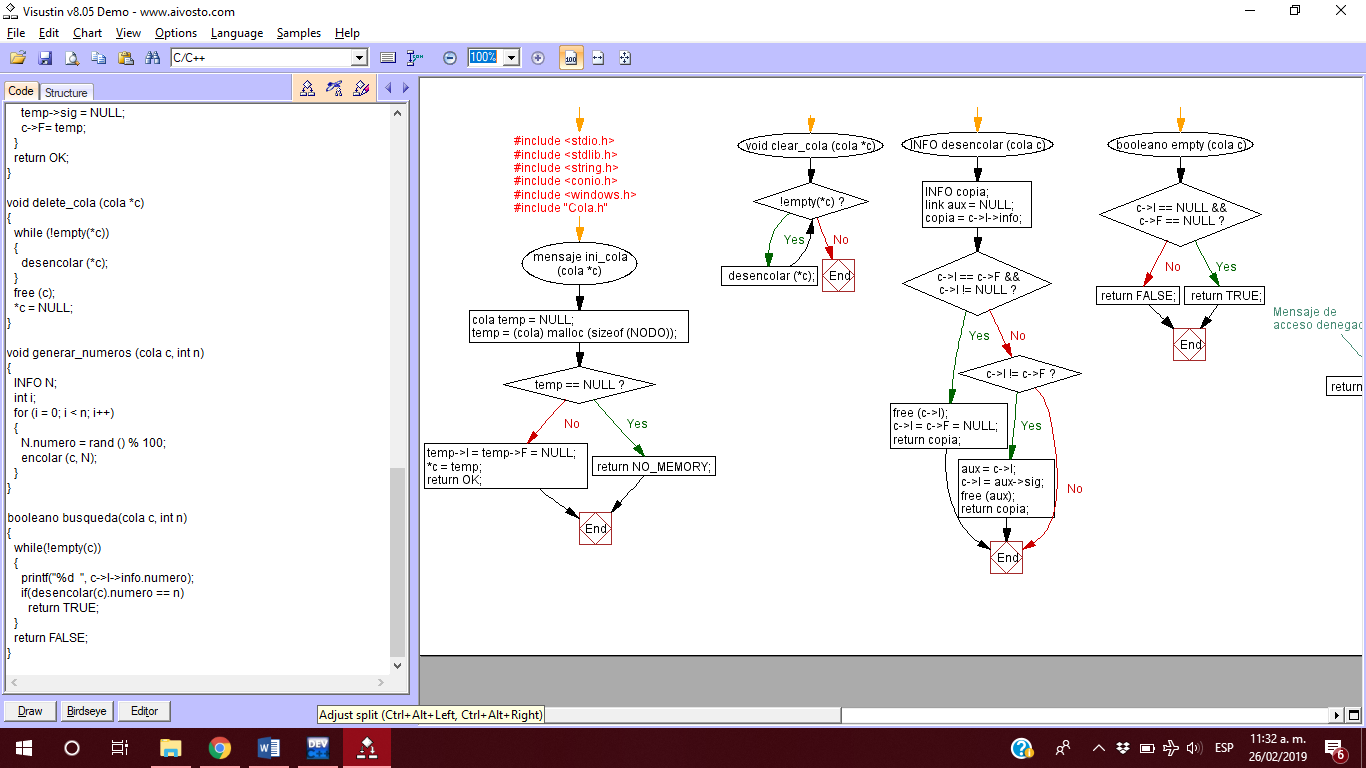
**Diseño de la solución**

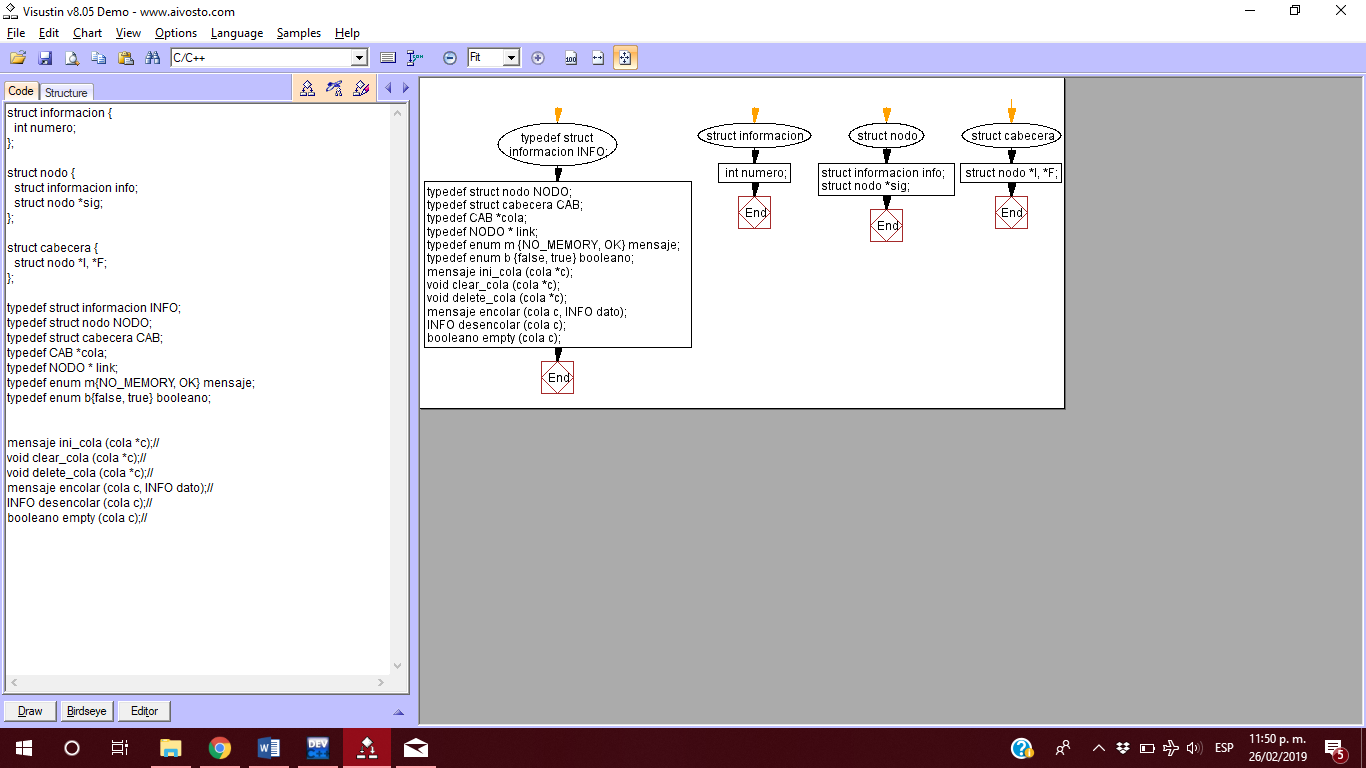
A continuación, se muestran los diagramas de flujo de la búsqueda lineal de las tres estructuras de datos en las cuales se realizará la misma búsqueda.

Primeramente, se muestra en el diagrama 1.1 el diseño de la solución de la búsqueda lineal en la cola.c

  
Diagrama 1.1 Cola.c

A continuación, se muestra la implementación de la cola, misma que fue implementada en clase de estructuras de datos con el profesor Yaxkin. (Yaxkin, 2017)

   
Diagrama 1.2 Cola implementación

  
Diagrama 1.3 Cola implementación (continuación)

A continuación, se muestra la implementación de la pila, misma que también fue implementada en clase de estructuras de datos con el profesor Yaxkin. (Yaxkin, 2017)

Primeramente, se muestra en el diagrama 1.4 el diseño de la solución de la búsqueda lineal en la pila.c.

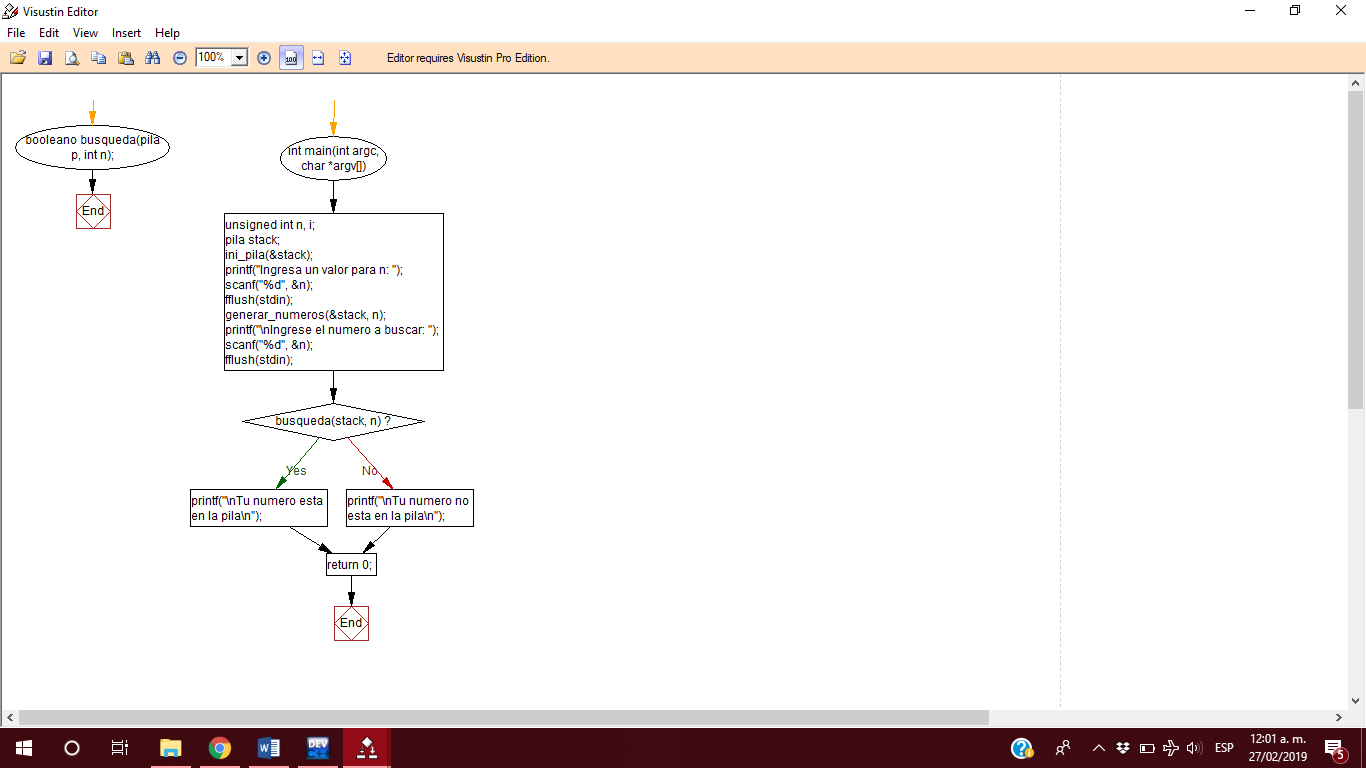
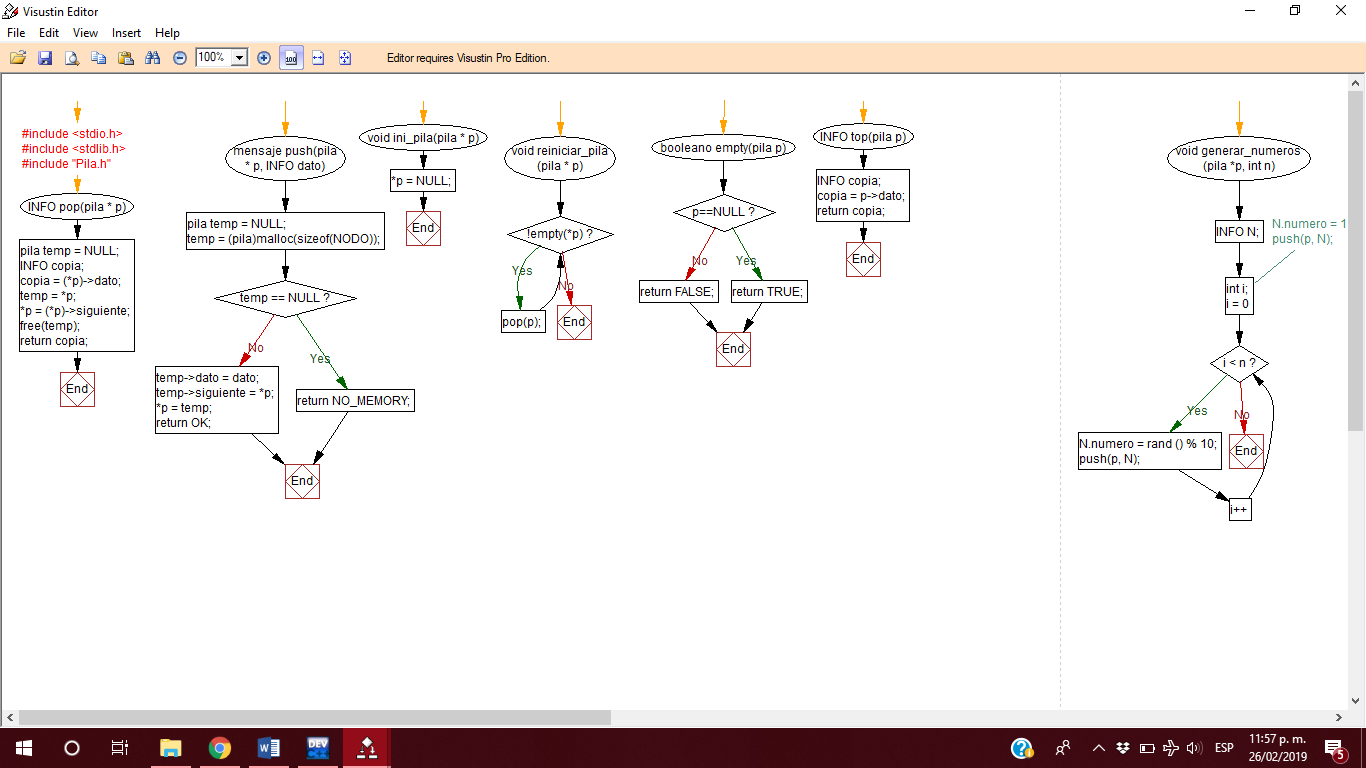
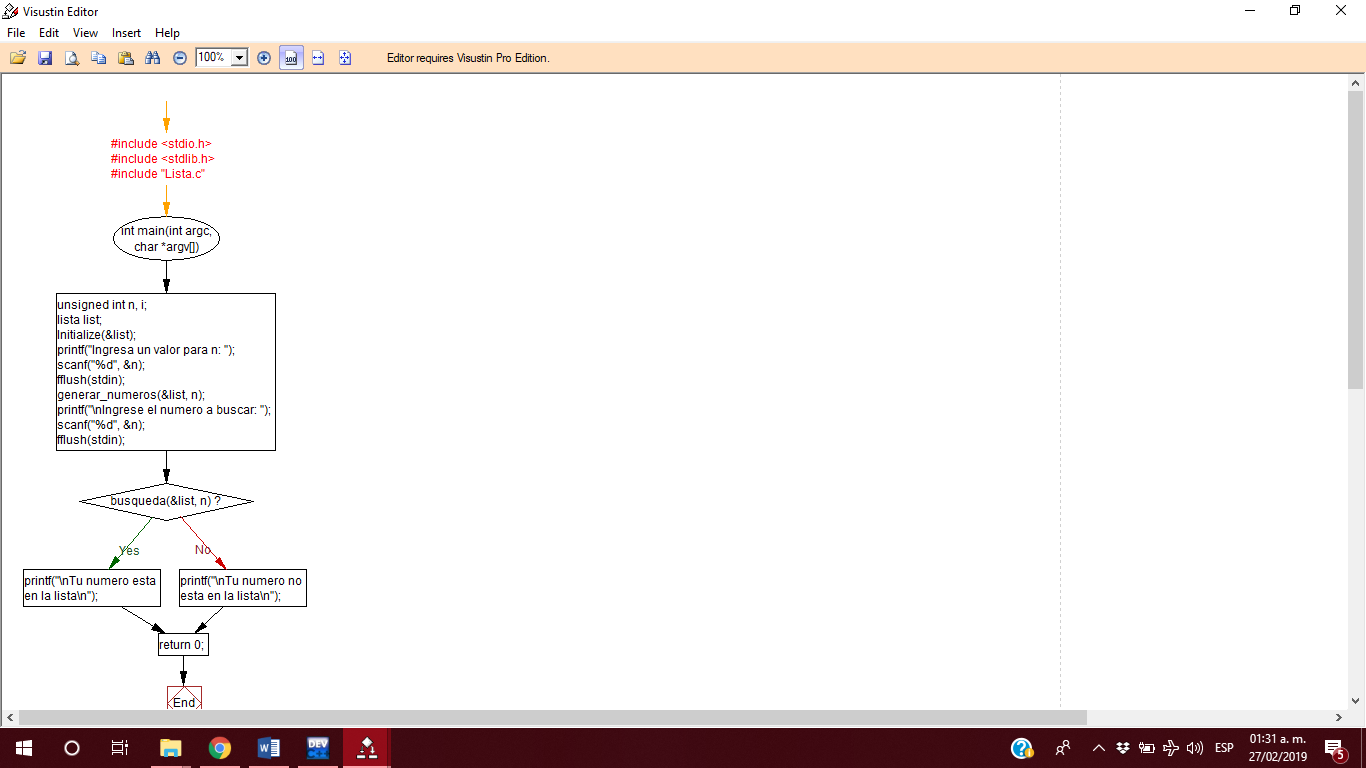


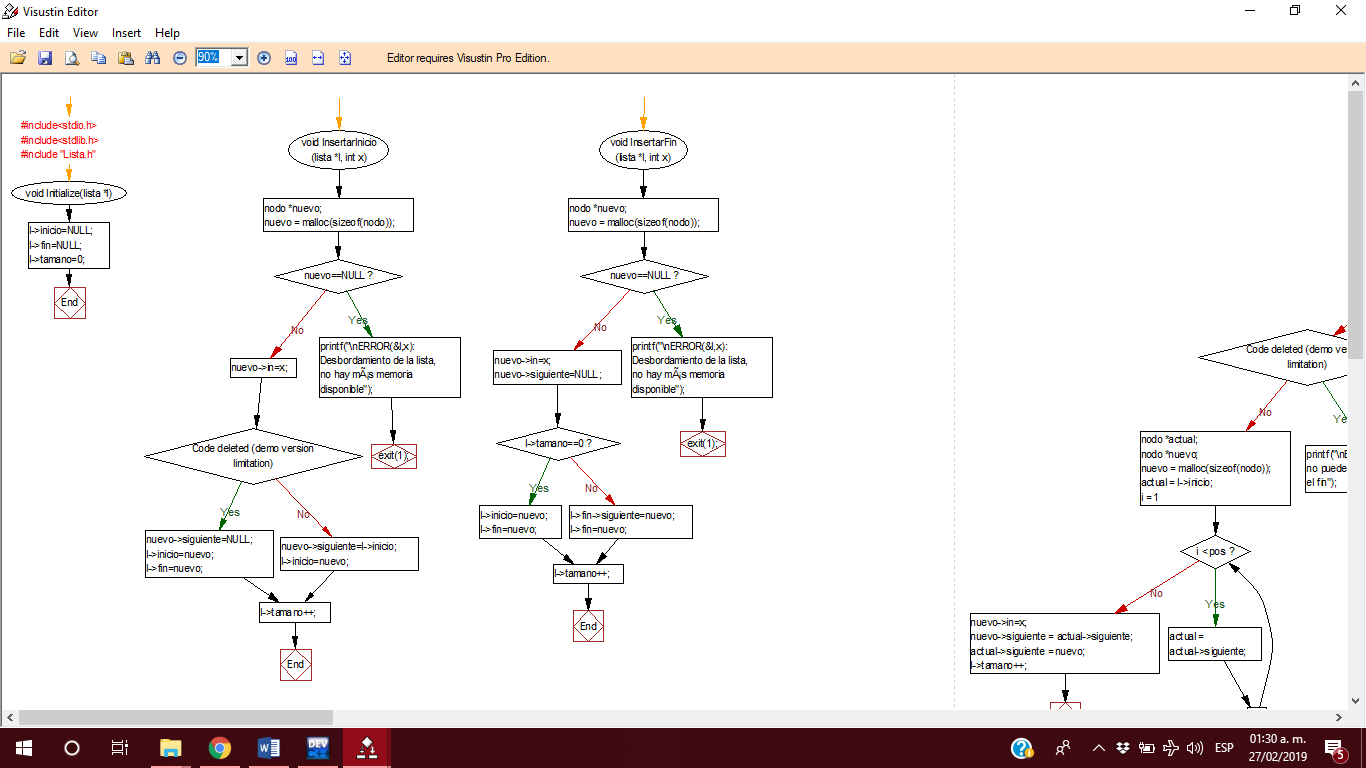
Diagrama 1.4 Pila solución

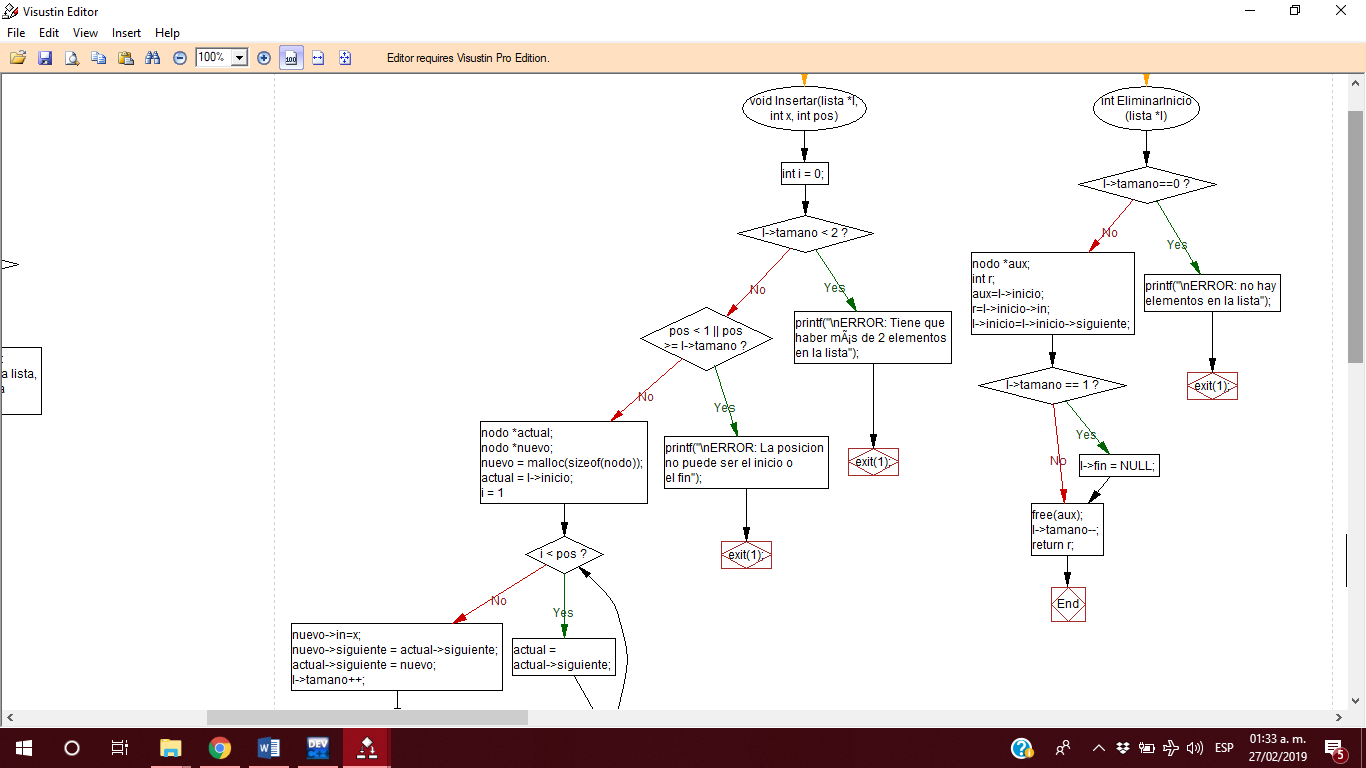
  
Diagrama 1.5 Pila implementación

A continuación, se muestra la implementación de una lista simplemente ligada, la cual fue creada en clase de estructuras de datos con el profesor Edgardo Adrián Franco. (Martínez)

Se muestra en el diagrama 1.6 el diseño de la solución de la búsqueda lineal en la lista.c.

  
Diagrama 1.6 Lista Main

  
Diagrama 1.7 Lista Implementación

  
Diagrama 1.8 Lista Implementación (continuación)

**Implementación de la solución**

1. /\*
2. PRACTICA 2
3. ALGORITMOS
4. PROFRA LUZ MARIA
5. MARCOS OSWALDO VAZQUEZ MORENO
6. ESCUELA SUPERIOR DE COMPUTO
7. 2016601777
9. \*/
10. #include <stdio.h>
11. #include <stdlib.h>
12. #include <string.h>
13. #include <conio.h>
14. #include <windows.h>
15. #include "Cola.h"
17. //COLA IMPLEMENTACION
19. mensaje ini\_cola (cola \*c)
20. {
21. cola temp = NULL;
22. temp = (cola) malloc (**sizeof** (NODO));
23. **if** (temp == NULL)
24. {
25. **return** NO\_MEMORY;
26. }
27. temp->I = temp->F = NULL;
28. \*c = temp;
29. **return** OK;
30. };
32. **void** clear\_cola (cola \*c)
33. {
34. **while** (!empty(\*c))
35. {
36. desencolar (\*c);
37. }
38. };
40. INFO desencolar (cola c)
41. {
42. INFO copia;
43. link aux = NULL;
44. copia = c->I->info;
45. **if** (c->I == c->F && c->I != NULL)
46. {
47. free (c->I);
48. c->I = c->F = NULL;
49. **return** copia;
50. }
51. **else** **if** (c->I != c->F)
52. {
53. aux = c->I;
54. c->I = aux->sig;
55. free (aux);
56. **return** copia;
57. }
58. }
60. booleano empty (cola c)
61. {
62. **if** (c->I == NULL && c->F == NULL)
63. **return** TRUE;
64. **return** FALSE;
65. }
67. mensaje encolar(cola c, INFO dato){
68. link temp = NULL;
69. temp = (link)malloc(**sizeof**(NODO)); //Solicitamos el nuevo nodo
70. **if**(temp==NULL) //Si el recurso fue denegado
71. **return** NO\_MEMORY; //Mensaje de acceso denegado
72. temp->info = dato; //Asignamos la información a guardar
73. **if**(empty(c)){ //Cola vacía
74. temp->sig = NULL;
75. c->I = c->F = temp;
76. }
77. **else**
78. {
79. c->F->sig = temp;
80. temp->sig = NULL;
81. c->F= temp;
82. }
83. **return** OK;
84. }
86. **void** delete\_cola (cola \*c)
87. {
88. **while** (!empty(\*c))
89. {
90. desencolar (\*c);
91. }
92. free (c);
93. \*c = NULL;
94. }
96. **void** generar\_numeros (cola c, **int** n)
97. {
98. INFO N;
99. **int** i;
100. **for** (i = 0; i < n; i++)
101. {
102. N.numero = rand () % 100;
103. encolar (c, N);
104. }
105. }
107. booleano busqueda(cola c, **int** n)
108. {
109. **while**(!empty(c))
110. {
111. printf("%d  ", c->I->info.numero);
112. **if**(desencolar(c).numero == n)
113. **return** TRUE;
114. }
115. **return** FALSE;
116. }
117. //MAIN COLA

120. #include <stdio.h>
121. #include <stdlib.h>
122. #include "Cola.c"

125. **int** main(**int** argc, **char** \*argv[]) {
126. unsigned **int** n, i;
127. cola queue;
128. ini\_cola(&queue);
129. printf("Ingresa un valor para n: ");
130. scanf("%d", &n);
131. fflush(stdin);
132. generar\_numeros(queue, n);
133. printf("\nIngrese el numero a buscar: ");
134. scanf("%d", &n);
135. fflush(stdin);
136. **if**(busqueda(queue, n)) printf("\nTu numero esta en la cola\n");
137. **else** printf("\nTu numero no esta en la cola\n");
138. **return** 0;
139. }
140. **struct** informacion {
141. **int** numero;
142. };
144. **struct** nodo {
145. **struct** informacion info;
146. **struct** nodo \*sig;
147. };
149. **struct** cabecera {
150. **struct** nodo \*I, \*F;
151. };
153. **typedef** **struct** informacion INFO;
154. **typedef** **struct** nodo NODO;
155. **typedef** **struct** cabecera CAB;
156. **typedef** CAB \*cola;
157. **typedef** NODO \* link;
158. **typedef** **enum** m{NO\_MEMORY, OK} mensaje;
159. **typedef** **enum** b{**false**, **true**} booleano;

162. mensaje ini\_cola (cola \*c);//
163. **void** clear\_cola (cola \*c);//
164. **void** delete\_cola (cola \*c);//
165. mensaje encolar (cola c, INFO dato);//
166. INFO desencolar (cola c);//
167. booleano empty (cola c);//

//Pila Implementación

1. /\*
2. PRACTICA 2
3. ALGORITMOS
4. PROFRA LUZ MARIA
5. MARCOS OSWALDO VAZQUEZ MORENO
6. ESCUELA SUPERIOR DE COMPUTO
7. 2016601777
9. \*/
10. #include <stdio.h>
11. #include <stdlib.h>
12. #include "Pila.h"
14. INFO pop(pila \* p)
15. {
16. pila temp = NULL;
17. INFO copia;
18. copia = (\*p)->dato;
19. temp = \*p;
20. \*p = (\*p)->siguiente;
21. free(temp);
22. **return** copia;
23. };
25. mensaje push(pila \* p, INFO dato)
26. {
27. pila temp = NULL;
28. temp = (pila)malloc(**sizeof**(NODO));
29. **if**(temp == NULL)
30. {
31. **return** NO\_MEMORY;
32. }
33. temp->dato = dato;
34. temp->siguiente = \*p;
35. \*p = temp;
36. **return** OK;
37. };
39. **void** ini\_pila(pila \* p)
40. {
41. \*p = NULL;
42. };
44. **void** reiniciar\_pila(pila \* p)
45. {
46. **while**(!empty(\*p))
47. pop(p);
48. };
50. booleano empty(pila p){
51. **if**(p==NULL)
52. {
53. **return** TRUE;
54. }
55. **return** FALSE;
56. };
58. INFO top(pila p)
59. {
60. INFO copia;
61. copia = p->dato;
62. **return** copia;
63. };
65. **void** generar\_numeros (pila \*p, **int** n)
66. {
67. INFO N;
68. //N.numero = 11;
69. //push(p, N);
70. **int** i;
71. **for** (i = 0; i < n; i++)
72. {
73. N.numero = rand () % 10;
74. push(p, N);
75. }
77. }
79. booleano busqueda(pila p, **int** n)
80. {
81. **while**(!empty(p))
82. {
83. printf("%d  ", top(p).numero);
84. **if**(pop(&p).numero == n)
85. **return** TRUE;
86. }
87. **return** FALSE;
88. }
89. //MAIN PILA
91. #include <stdio.h>
92. #include <stdlib.h>
93. #include "Pila.c"

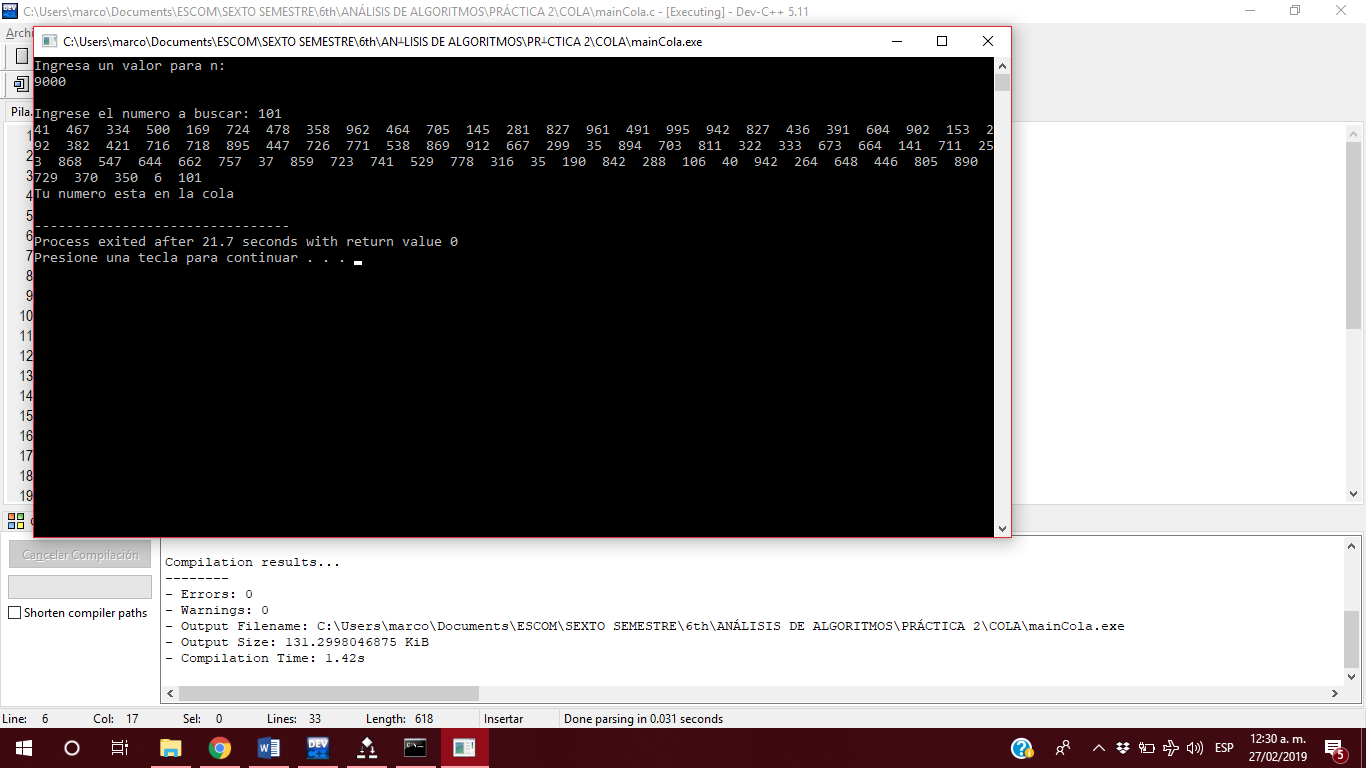
96. booleano busqueda(pila p, **int** n);
98. **int** main(**int** argc, **char** \*argv[]) {
99. unsigned **int** n, i;
100. pila stack;
101. ini\_pila(&stack);
102. printf("Ingresa un valor para n: ");
103. scanf("%d", &n);
104. fflush(stdin);
105. generar\_numeros(&stack, n);
106. printf("\nIngrese el numero a buscar: ");
107. scanf("%d", &n);
108. fflush(stdin);
109. **if**(busqueda(stack, n)) printf("\nTu numero esta en la pila\n");
110. **else** printf("\nTu numero no esta en la pila\n");
111. **return** 0;
112. }
113. /\*
114. PRACTICA 2
115. ALGORITMOS
116. PROFRA LUZ MARIA
117. MARCOS OSWALDO VAZQUEZ MORENO
118. ESCUELA SUPERIOR DE COMPUTO
119. 2016601777
121. \*/
122. **struct** informacion
123. {
124. **int** numero;
125. };
127. **struct** nodo
128. {
129. **struct** informacion dato;
130. **struct** nodo \* siguiente;
131. };
133. **typedef** **struct** nodo \* pila;
134. **typedef** **struct** informacion INFO;
135. **typedef** **struct** nodo NODO;
136. **typedef** **enum** B{FALSE,TRUE} booleano;
137. **typedef** **enum** M{NO\_MEMORY,OK} mensaje;
139. **void** ini\_pila(pila \* p);
140. **void** reiniciar\_pila(pila \* p);
141. INFO pop(pila \* p); //Último elemento guardado
142. mensaje push(pila \* p, INFO dato); //Confirma el último elemento leído
143. booleano empty(pila p); //Vacía la pila
144. INFO top(pila p); //Obtiene una copia del top de la pila

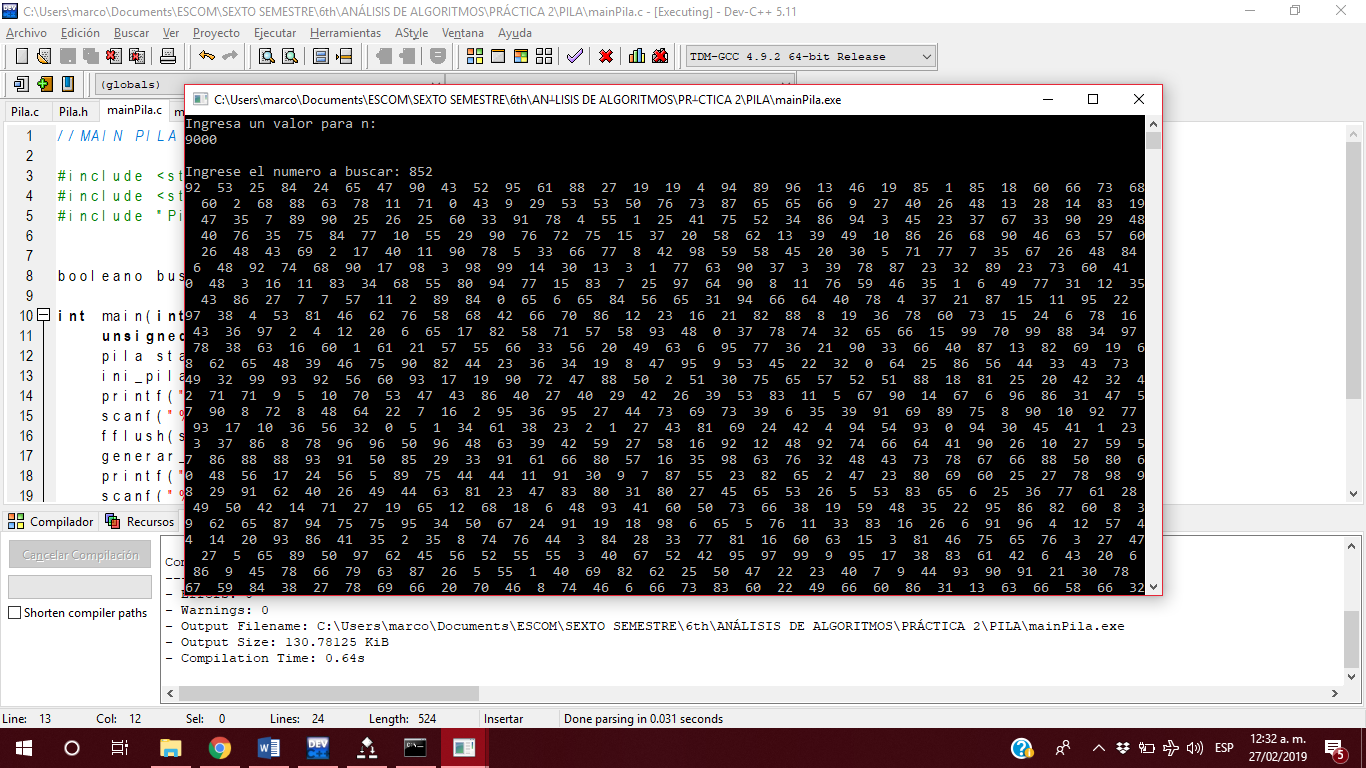
//Lista implementación

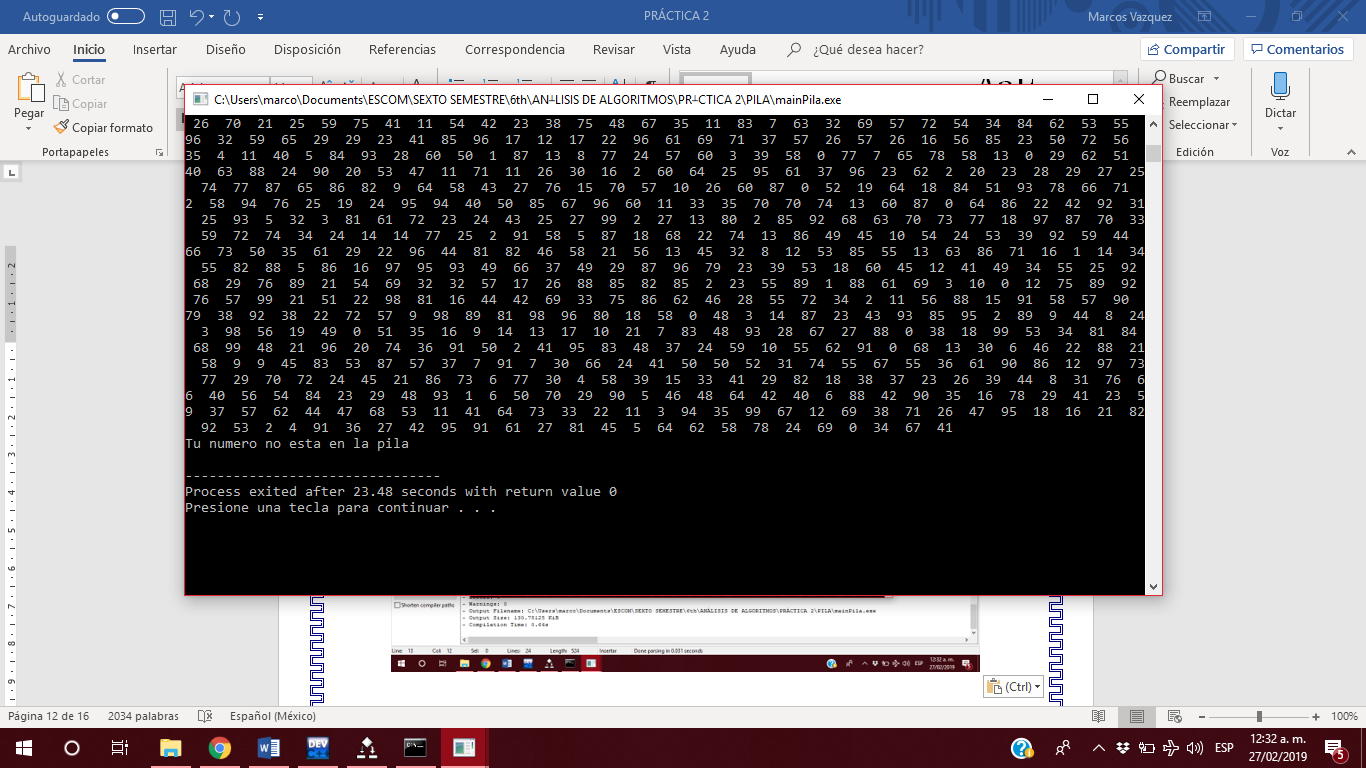
1. #include<stdio.h>
2. #include<stdlib.h>
3. #include "Lista.h"

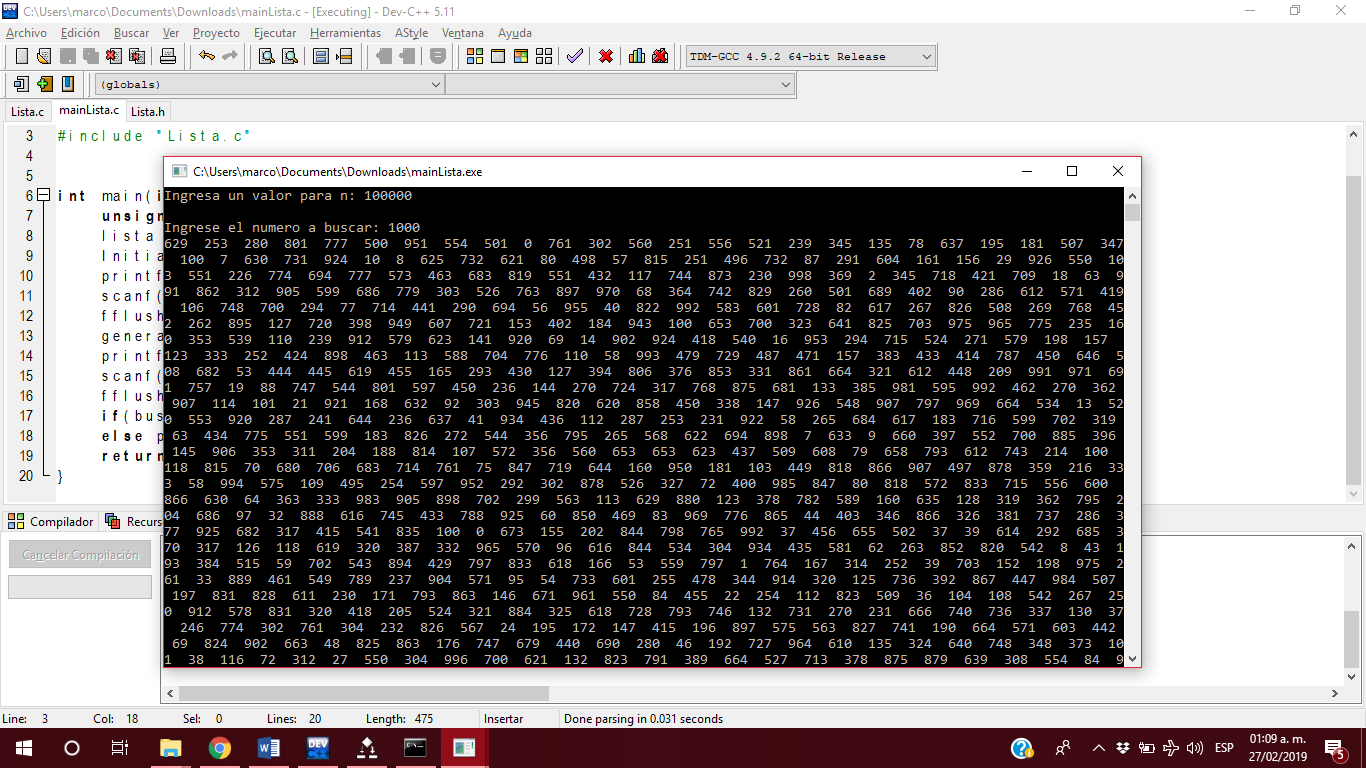
6. **void** Initialize(lista \*l){
7. l->inicio=NULL;
8. l->fin=NULL;
9. l->tamano=0;
10. **return**;
11. }
13. **void** InsertarInicio (lista \*l, **int** x){
14. nodo \*nuevo;
15. nuevo = malloc(**sizeof**(nodo));
16. **if**(nuevo==NULL){
17. printf("\nERROR(&l,x): Desbordamiento de la lista, no hay mÃ¡s memoria disponible");
18. exit(1);
19. }
20. nuevo->in=x;
21. **if**(l->tamano==0){
22. nuevo->siguiente=NULL;
23. l->inicio=nuevo;
24. l->fin=nuevo;
26. }
27. **else**{
28. nuevo->siguiente=l->inicio;
29. l->inicio=nuevo;
30. }
31. l->tamano++;
32. **return**;
33. }
35. **void** InsertarFin (lista \*l, **int** x){
36. nodo \*nuevo;
37. nuevo = malloc(**sizeof**(nodo));
38. **if**(nuevo==NULL){
39. printf("\nERROR(&l,x): Desbordamiento de la lista, no hay mÃ¡s memoria disponible");
40. exit(1);
41. }
42. nuevo->in=x;
43. nuevo->siguiente=NULL;
44. **if**(l->tamano==0){
45. l->inicio=nuevo;
46. l->fin=nuevo;
47. }
48. **else**{
49. l->fin->siguiente=nuevo;
50. l->fin=nuevo;
51. }
52. l->tamano++;
53. **return**;
54. }
56. **void** Insertar(lista \*l, **int** x, **int** pos){
57. **int** i = 0;
58. **if** (l->tamano < 2){
59. printf("\nERROR: Tiene que haber mÃ¡s de 2 elementos en la lista");
60. exit(1);
61. }
62. **if** (pos < 1 || pos >= l->tamano){
63. printf("\nERROR: La posicion no puede ser el inicio o el fin");
64. exit(1);
65. }
67. nodo \*actual;
68. nodo \*nuevo;
70. nuevo = malloc(**sizeof**(nodo));
71. actual = l->inicio;
72. **for** (i = 1; i < pos; ++i)
73. actual = actual->siguiente;
74. nuevo->in=x;
75. nuevo->siguiente = actual->siguiente;
76. actual->siguiente = nuevo;
77. l->tamano++;
78. **return**;
79. }
81. **int** EliminarInicio(lista \*l){
82. **if**(l->tamano==0){
83. printf("\nERROR: no hay elementos en la lista");
84. exit(1);
85. }
86. nodo \*aux;
87. **int** r;
88. aux=l->inicio;
89. r=l->inicio->in;
90. l->inicio=l->inicio->siguiente;
92. **if** (l->tamano == 1)
93. l->fin = NULL;
94. free(aux);
95. l->tamano--;
96. **return** r;
97. }
99. **int** Eliminar (lista \*l, **int** pos){
100. **int** i = 0;
101. **if** (l->tamano <= 1 || pos < 1 || pos >= l->tamano){
102. printf("\nERROR: Posicion no valida");
103. exit(1);
104. }
105. **int** r;
106. nodo \*aux;
107. nodo \*actual;
109. actual=l->inicio;
110. **for** (i = 1; i < pos; ++i)
111. actual = actual->siguiente;
113. aux=actual->siguiente;
114. r=aux->in;
115. actual->siguiente=actual->siguiente->siguiente;
117. **if**(actual->siguiente == NULL)
118. l->fin = actual;
120. free(aux);
121. l->tamano--;
122. **return** r;
123. }
125. **int** Ver (lista \*l, **int** pos){
126. **int** r, i = 0;
127. nodo \*actual;
128. **if**(pos>=l->tamano){
129. printf("\nERROR: posicion no valida");
130. exit(1);
131. }
132. actual=l->inicio;
133. **for** (i = 1; i < pos; ++i)
134. actual = actual->siguiente;
135. **if**(pos==0)
136. r=actual->in;
137. **else**
138. r=actual->siguiente->in;
140. **return** r;
141. }
143. **void** Destroy (lista \*l){
144. **while** (l->tamano > 0){
145. EliminarInicio (l);
146. }
147. }
149. **int** Size(lista \*l){
150. **return** l->tamano;
151. }
153. **int** empty(lista \*l)
154. {
155. **return** l->tamano == 0;
156. }
158. **int** busqueda(lista \*l, **int** n)
159. {
160. **int** tam = Size(l);
161. **int** pos = 0;
162. **while**(!empty(l))
163. {
164. printf("%d  ", Ver(l, pos));
165. **if**(l->inicio->in == n)
166. **return** TRUE;
167. EliminarInicio(l);
168. }
169. **return** FALSE;
170. }
172. **void** generar\_numeros (lista \*l, **int** n)
173. {
174. **int** i, num;
175. **for** (i = 0; i < n; i++)
176. {
177. num = rand () % 1000;
178. InsertarInicio(l, num);
179. }
181. }

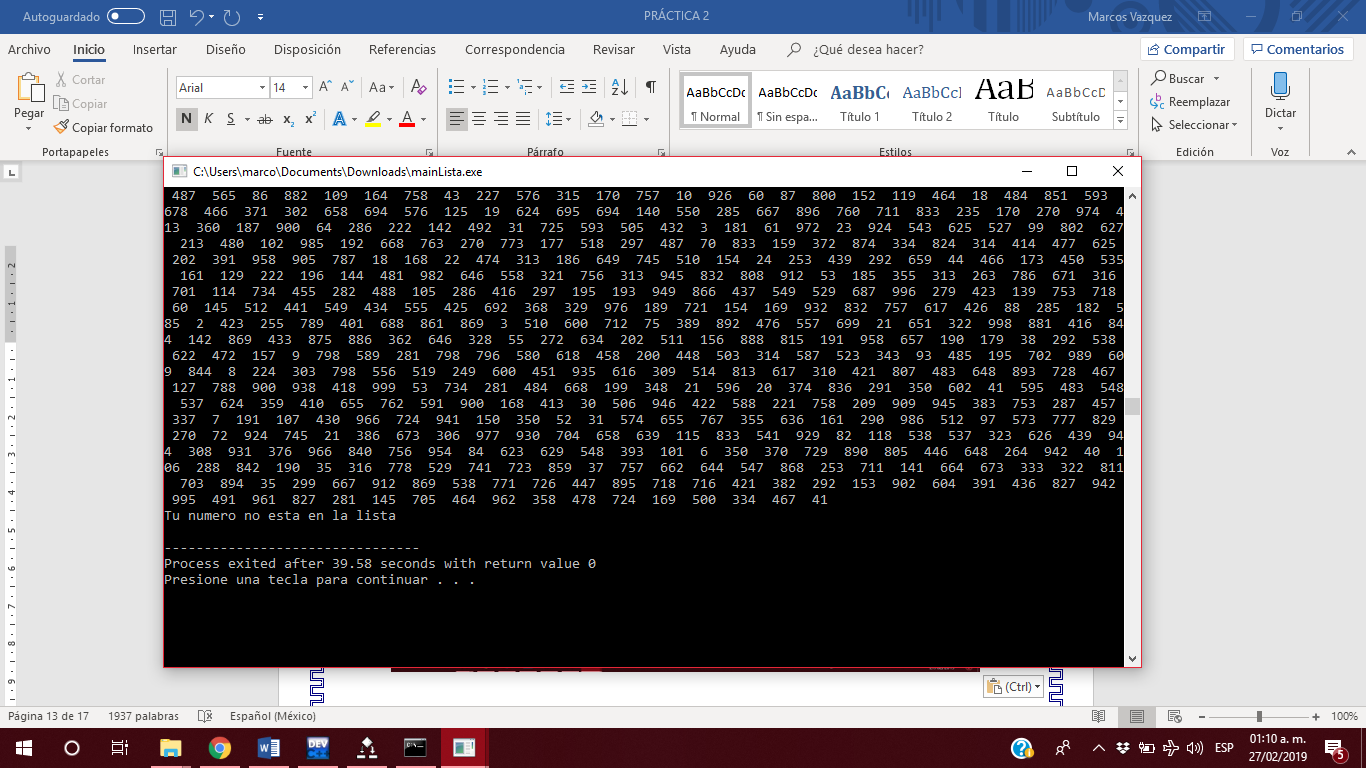
**Funcionamiento**

  
Imagen 2.1 Prueba Cola

  
Imagen 2.2 Prueba Pila

Imagen 2.3 Prueba Pila

  
Imagen 2.4 Prueba Lista

  
Imagen 2.5 prueba Lista

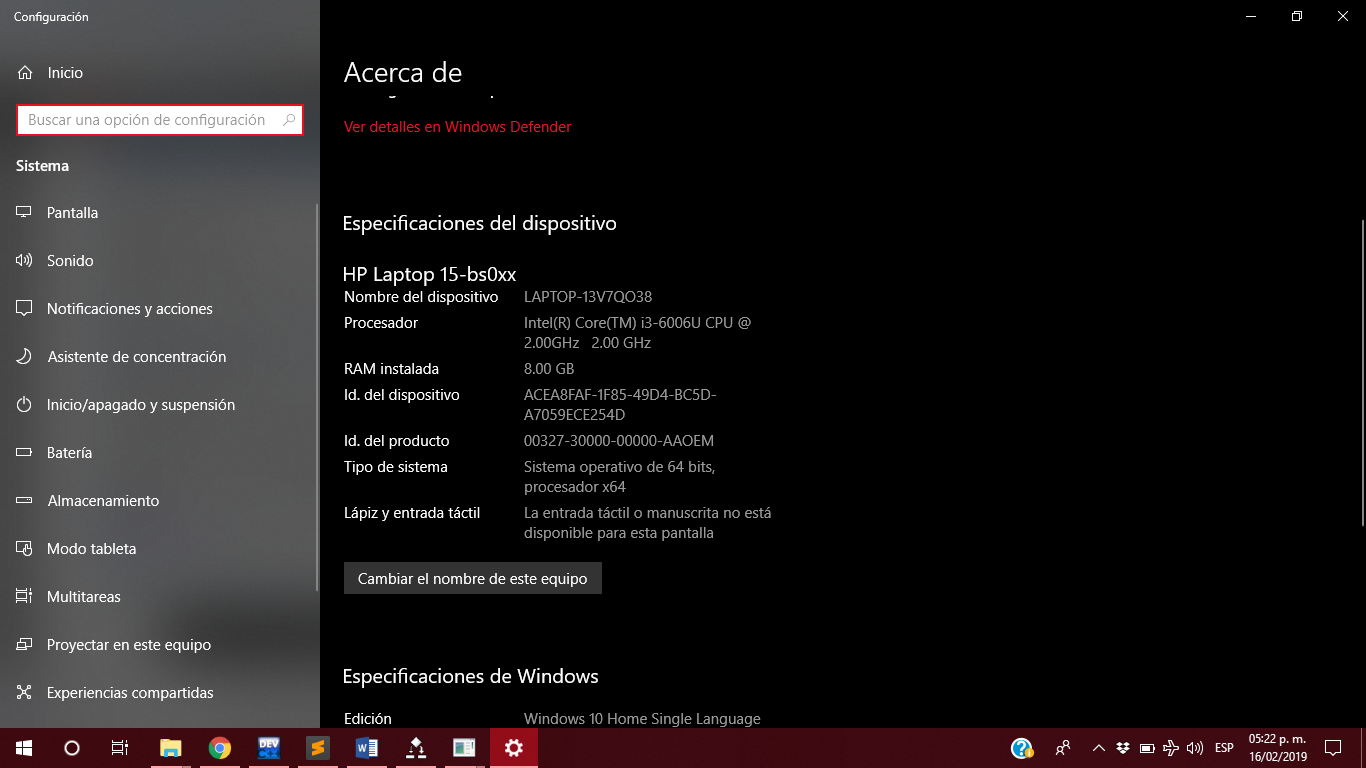
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **n** | **f(n) temporal** | **f(n) espacial** | **Mejor caso**  **PILA** | **Mejor caso**  **COLA** | **Mejor caso**  **LISTA** |
| **1** | n+2 | n | 0.00023s | 0.00015s | 0.00010s |
| **10** | n+2 | n | 0.00045s | 0.00019s | 0.00027s |
| **100** | n+2 | n | 0.00078s | 0.00085s | 0.00068s |
| **1000** | n+2 | n | 0.0015s | 0.00362s | 0.00125s |
| **10000** | n+2 | n | 0.089s | 0.019s | 0.025s |
| **100000** | n+2 | n | 1.99s | 1.75s | 1.0723s |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **n** | **f(n) temporal** | **f(n) espacial** | **Caso promedio**  **PILA** | **Caso promedio**  **COLA** | **Caso promedio**  **LISTA** |
| **1** | n+2 | n | 2.45s | 1.85s | 1.10s |
| **10** | n+2 | n | 4.96s | 2.45s | 1.576s |
| **100** | n+2 | n | 6.11s | 5.89s | 2.21s |
| **1000** | n+2 | n | 8.97s | 6.10s | 3.23s |
| **10000** | n+2 | n | 12.78s | 8.12s | 5.90s |
| **100000** | n+2 | n | 13.56s | 9.73s | 7.67s |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **n** | **f(n) temporal** | **f(n) espacial** | **Peor caso**  **PILA** | **Peor caso**  **COLA** | **Peor caso**  **LISTA** |
| **1** | n+2 | n | .89s | 1.76s | 2.12s |
| **10** | n+2 | n | 3.74s | 8.12s | 4.56s |
| **100** | n+2 | n | 8.45s | 15.32s | 7.89s |
| **1000** | n+2 | n | 12.87s | 17.90s | 24.7s |
| **10000** | n+2 | n | 18.23s | 20.89s | 31.12s |
| **100000** | n+2 | n | 23.489s | 29.15s | 39.58s |

**Plataforma experimental**

La ejecución de los algoritmos anteriores se llevó a cabo en una computadora personal que se describe en la siguiente imagen.

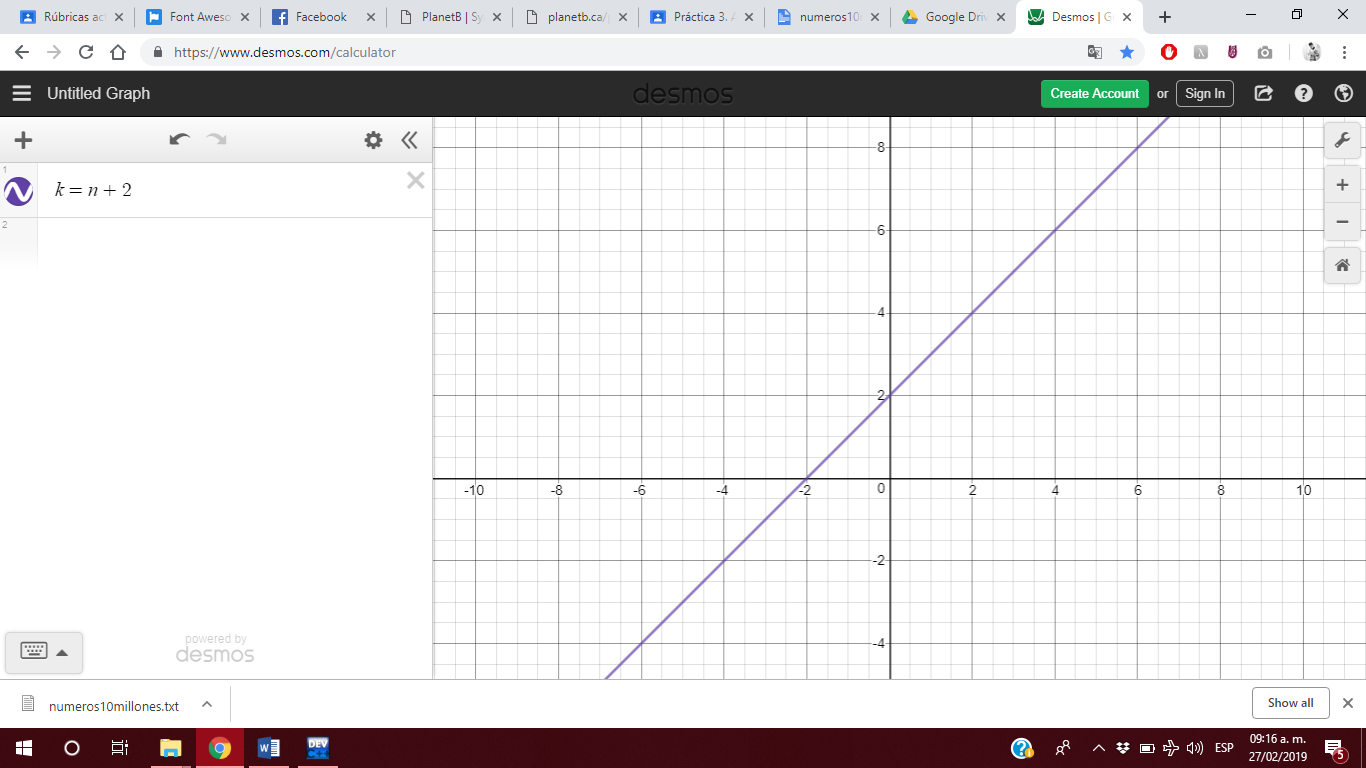
  
Imagen 3.1 Plataforma Experimental

El compilador utilizado fue gcc integrado dentro del IDE DevC en un sistema operativo de 64 bits Windows 10.

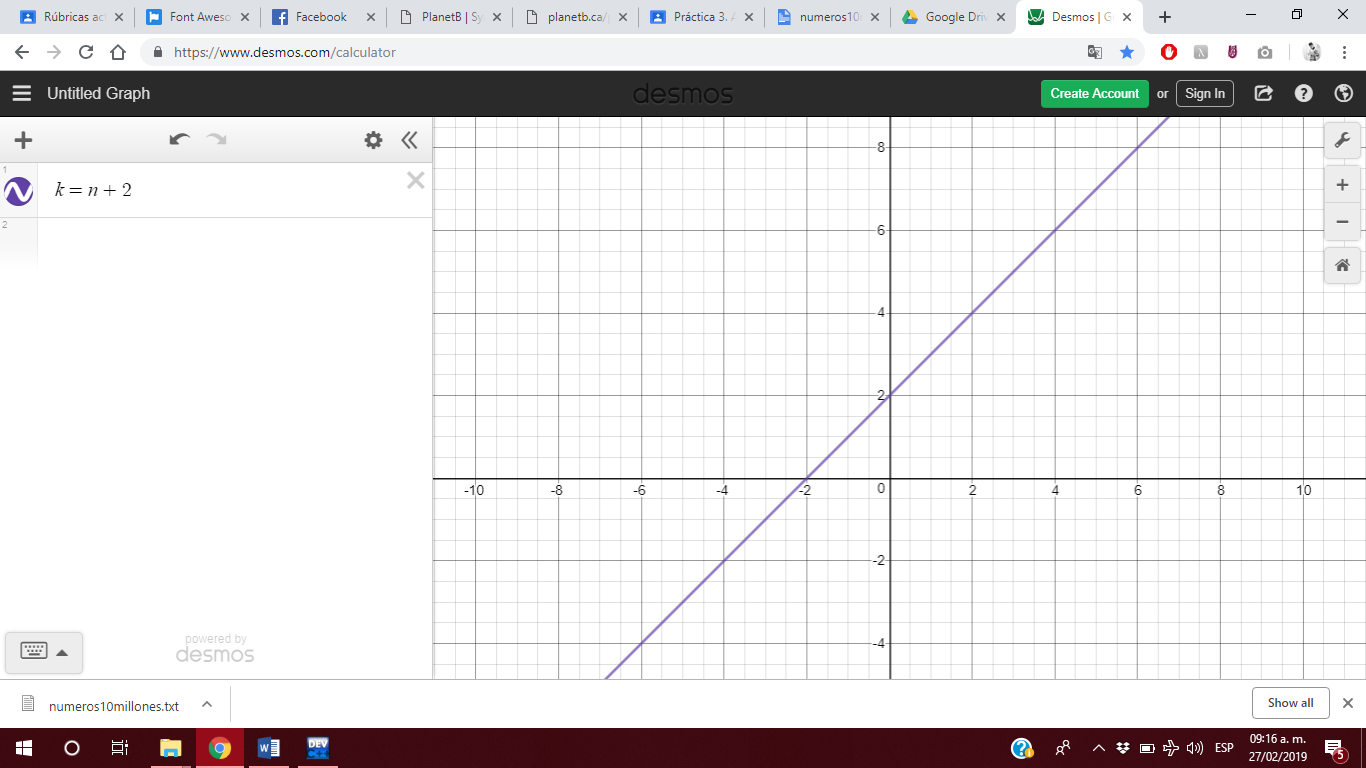
**Graficas de funciones**

Para la cola, la pila y la lista la gráfica es la misma ya que todas son de función x=n+2.

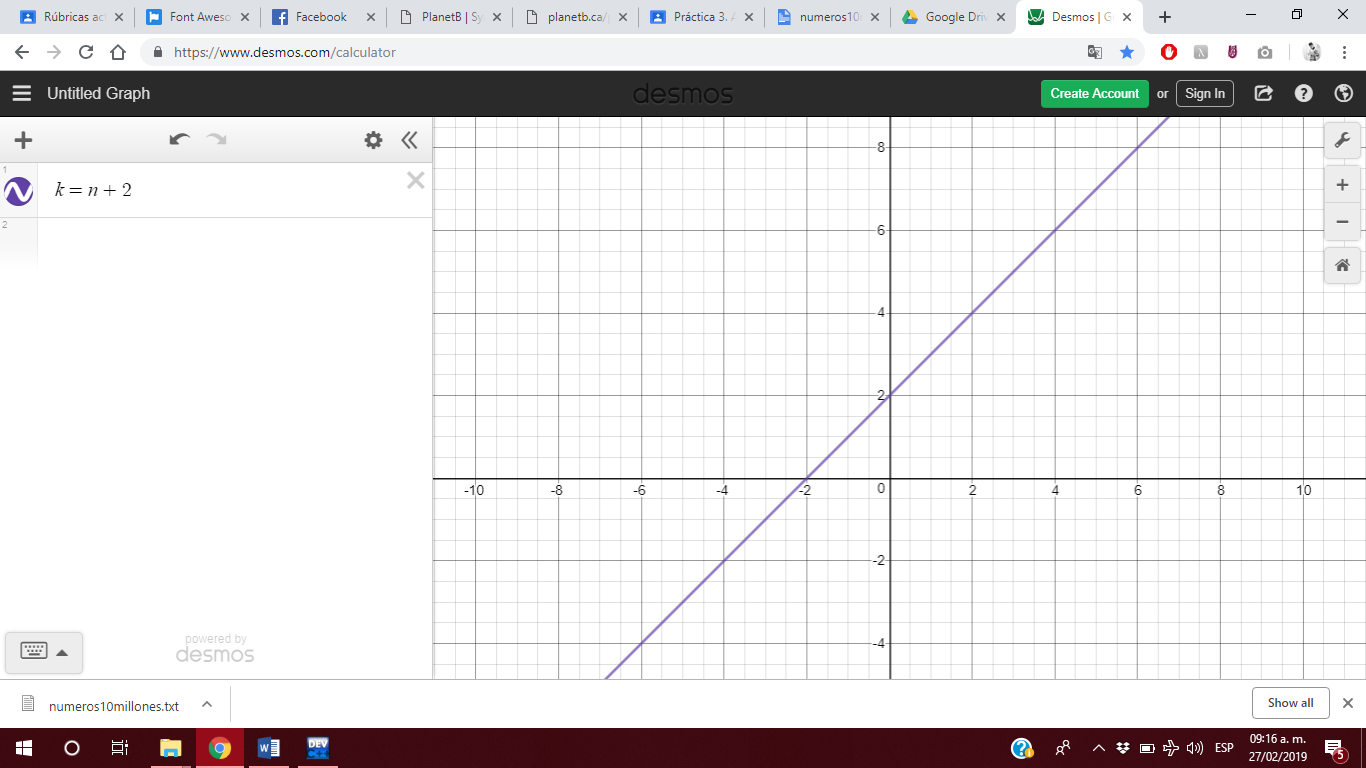
Cola

  
Gráfica 1.1 Cola

Pila

  
Gráfica 1.1 Pila

Lista

  
Gráfica 1.1 Lista

**Conclusiones**

En conclusión se puede decir que lo primordial en el tiempo de demora cuando se hace una búsqueda lineal dentro de una estructura influye demasiado cuántos números aleatorios se piden a generar y dentro de la función que genera aleatorios, el margen de números en los que se va a generar, por ejemplo imagina que se deben 10000 de generar en un rango de 0 a 1000, tardará más que solo generar números de 0 a 9, por otro lado, es importante decir que lo encuentra más fácil en una lista simplemente ligada que en una pila pues se tienen que hacen push y pop, operaciones de la pila que son más costosas en tiempo.

También, es importante mencionar que el hecho de que se busque un numero y no se encuentre en la estructura, es decir, el algoritmo encuentre su peor caso hace que el algoritmo vaya hasta el final de los números generados, generando así un tiempo mayor de ejecución.

Es importante mencionar, que lo que hemos visto en clase lo estamos llevando a la utilidad en su mayor grado de intensidad, pues es algo que quizá es muy utilizado en materias de Diseño de Sistemas Digitales o Arquitectura de Computadoras pero aquí con números más grandes nos damos cuenta de lo quizá tardado que esto puede llegar a ser.

**Bibliografía**

# Bibliografía

Martínez, E. A. (s.f.). *www.eafranco.com.* Recuperado el 2017, de http://www.eafranco.com/docencia/estructurasdedatos

Visustin v8.05

Syntax Highlight Code In Word Documents *http://planetb.ca/syntax-highlight-word*

Desmos Graphic *https://www.desmos.com/calculator*

Yaxkin F.M. (2017). Cola Implementada. ESCOM. CDMX.