



**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 = $a(1 + 2 + \dots + n) + bn = a(n(n+1) / 2) + bn$, 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, ..., enésima 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

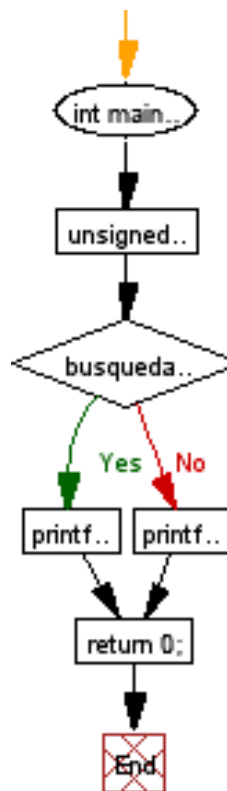


Diagrama 1.1 Cola.c

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)

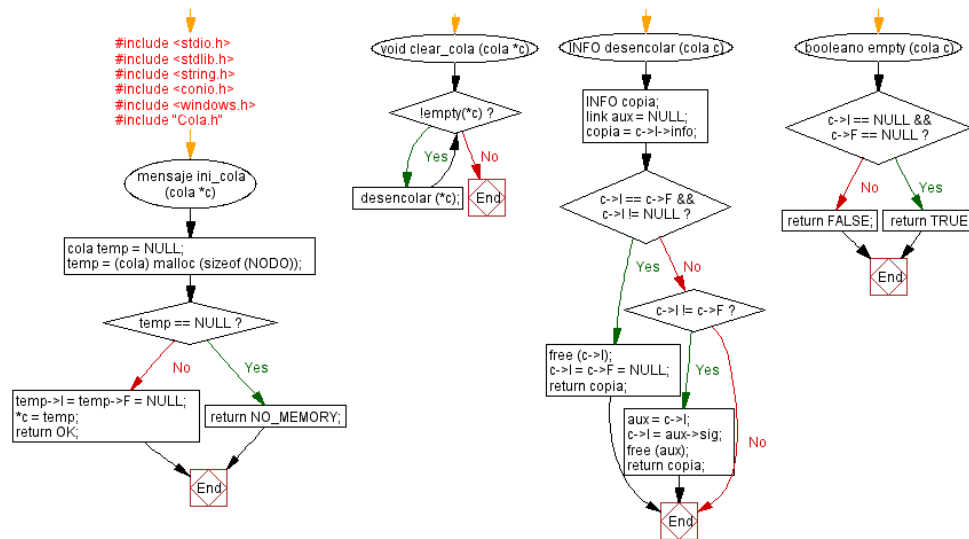


Diagrama 1.2 Cola implementación

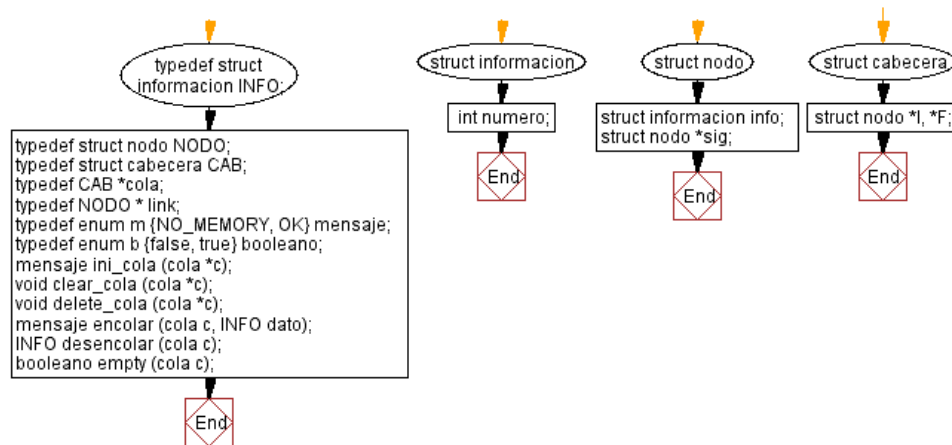


Diagrama 1.3 Cola implementación (continuación)

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.

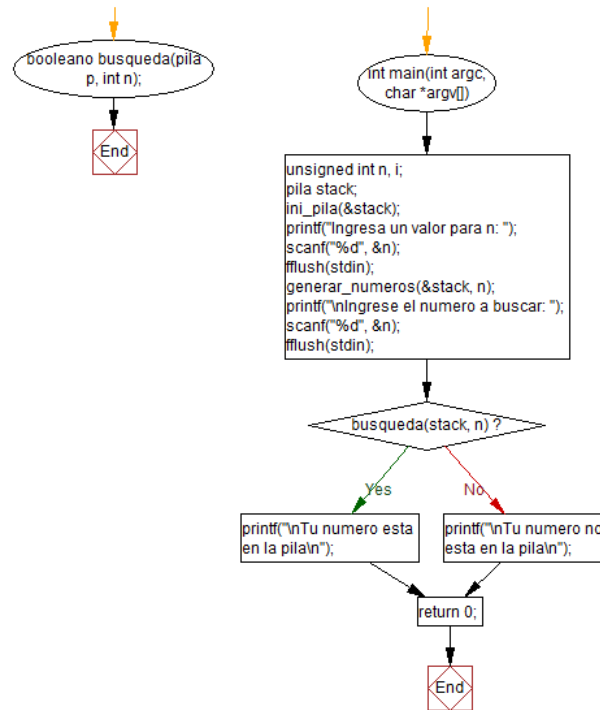


Diagrama 1.4 Pila solución

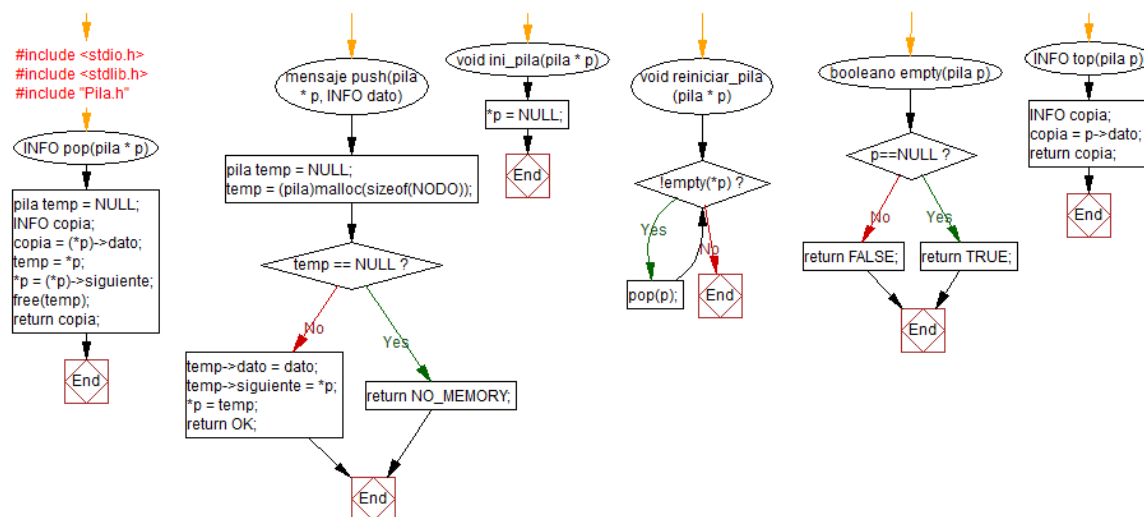


Diagrama 1.5 Pila implementación

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.

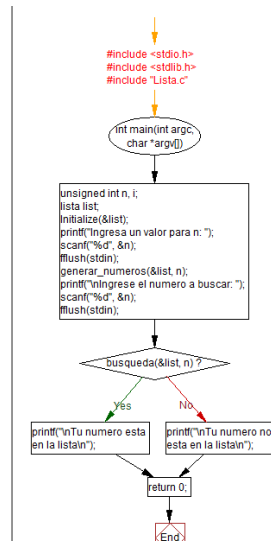


Diagrama 1.6 Lista Main

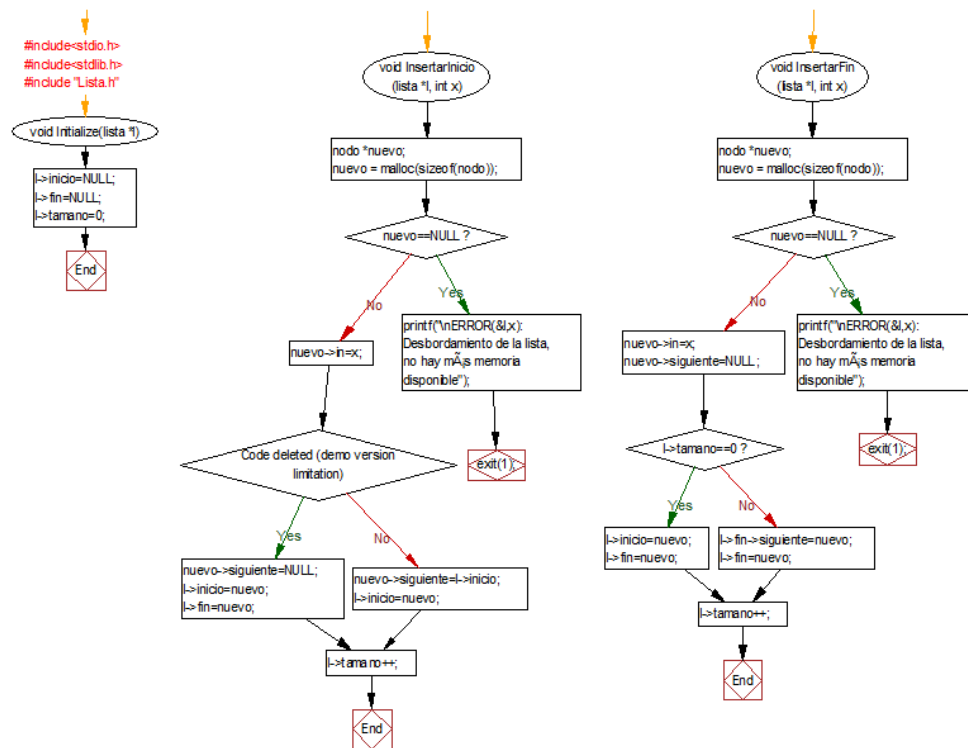


Diagrama 1.7 Lista Implementación

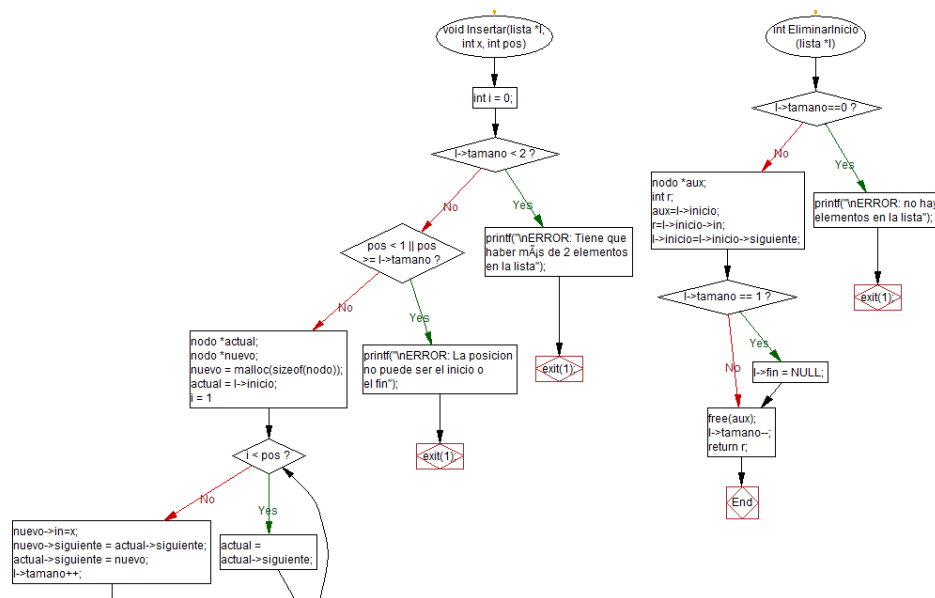


Diagrama 1.8 Lista Implementación (continuación)

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
8.
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"
16.
17. //COLA IMPLEMENTACION
18.
19. mensaje iniCola (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. };
31.
32. void clearCola (cola *c)
33. {
34.     while (!empty(*c))
35.     {
36.         desencolar (*c);
37.     }
38. };
39.
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. }
59.
60. booleano empty (cola c)
61. {
62.     if (c->I == NULL && c->F == NULL)
63.         return TRUE;
64.     return FALSE;
65. }
66.
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. }
85.
86. void deleteCola (cola *c)
87. {
88.     while (!empty(*c))
89.     {
90.         desencolar (*c);
91.     }

```



```

92.     free (c);
93.     *c = NULL;
94. }
95.
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. }
106.
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
118.
119.
120. #include <stdio.h>
121. #include <stdlib.h>
122. #include "Cola.c"
123.
124.
125. int main(int argc, char *argv[]) {
126.     unsigned int n, i;
127.     cola queue;
128.     iniCola(&queue);
129.     printf("Ingresa un valor para n: ");
130.     scanf("%d", &n);
131.     fflush(stdin);
132.     generar_numeros(queue, n);
133.     printf("\nIngresa 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. }

```

```

1. struct informacion {
2.     int numero;
3. };
4.
5. struct nodo {
6.     struct informacion info;
7.     struct nodo *sig;
8. };
9.
10. struct cabecera {

```

```

11.     struct nodo *I, *F;
12. };
13.
14. typedef struct informacion INFO;
15. typedef struct nodo NODO;
16. typedef struct cabecera CAB;
17. typedef CAB *cola;
18. typedef NODO * link;
19. typedef enum m{NO_MEMORY, OK} mensaje;
20. typedef enum b{false, true} booleano;
21.
22.
23. mensaje ini_cola (cola *c);//
24. void clear_cola (cola *c);//
25. void delete_cola (cola *c);//
26. mensaje encolar (cola c, INFO dato);//
27. INFO desencolar (cola c);//
28. 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
8.
9.  */
10. #include <stdio.h>
11. #include <stdlib.h>
12. #include "Pila.h"
13.
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. };
24.
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. };
38.

```

```

39. void ini_pila(pila * p)
40. {
41.     *p = NULL;
42. };
43.
44. void reiniciar_pila(pila * p)
45. {
46.     while(!empty(*p))
47.         pop(p);
48. };
49.
50. booleano empty(pila p){
51.     if(p==NULL)
52.     {
53.         return TRUE;
54.     }
55.     return FALSE;
56. };
57.
58. INFO top(pila p)
59. {
60.     INFO copia;
61.     copia = p->dato;
62.     return copia;
63. };
64.
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.     }
76.
77. }
78.
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
90.
91. #include <stdio.h>
92. #include <stdlib.h>
93. #include "Pila.c"
94.
95.
96. booleano busqueda(pila p, int n);
97.
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("\nIngresa el numero a buscar: ");
107.     scanf("%d", &n);
108.     fflush(stdin);
109.     if(búsqueda(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.

```

```

1.  /*
2.  PRACTICA 2
3.  ALGORITMOS
4.  PROFRA LUZ MARIA
5.  MARCOS OSWALDO VAZQUEZ MORENO
6.  ESCUELA SUPERIOR DE COMPUTO
7.  2016601777
8.
9.  */
10. struct informacion
11. {
12.     int numero;
13. };
14.
15. struct nodo
16. {
17.     struct informacion dato;
18.     struct nodo * siguiente;
19. };
20.
21. typedef struct nodo * pila;
22. typedef struct informacion INFO;
23. typedef struct nodo NODO;
24. typedef enum B{FALSE,TRUE} booleano;
25. typedef enum M{NO_MEMORY,OK} mensaje;
26.
27. void ini_pila(pila * p);
28. void reiniciar_pila(pila * p);
29. INFO pop(pila * p); //Último elemento guardado
30. mensaje push(pila * p, INFO dato); //Confirma el último elemento leído
31. booleano empty(pila p); //Vacía la pila
32. 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"
4.
5.
6.  void Initialize(lista *l){
7.     l->inicio=NULL;
8.     l->fin=NULL;
9.     l->tamano=0;

```

```

10.     return;
11. }
12.
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 más 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;
25.     }
26.     else{
27.         nuevo->siguiente=l->inicio;
28.         l->inicio=nuevo;
29.     }
30.     l->tamano++;
31.     return;
32. }
33.
34.
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 más 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. }
55.
56. void Insertar(lista *l, int x, int pos){
57.     int i = 0;
58.     if (l->tamano < 2){
59.         printf("\nERROR: Tiene que haber más 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.     }
66.
67.     nodo *actual;
68.     nodo *nuevo;
69.

```

```

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. }
80.
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;
91.
92.     if (l->tamano == 1)
93.         l->fin = NULL;
94.     free(aux);
95.     l->tamano--;
96.     return r;
97. }
98.
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;
108.
109.     actual=l->inicio;
110.     for (i = 1; i < pos; ++i)
111.         actual = actual->siguiente;
112.
113.     aux=actual->siguiente;
114.     r=aux->in;
115.     actual->siguiente=actual->siguiente->siguiente;
116.
117.     if(actual->siguiente == NULL)
118.         l->fin = actual;
119.
120.     free(aux);
121.     l->tamano--;
122.     return r;
123. }
124.
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.     }

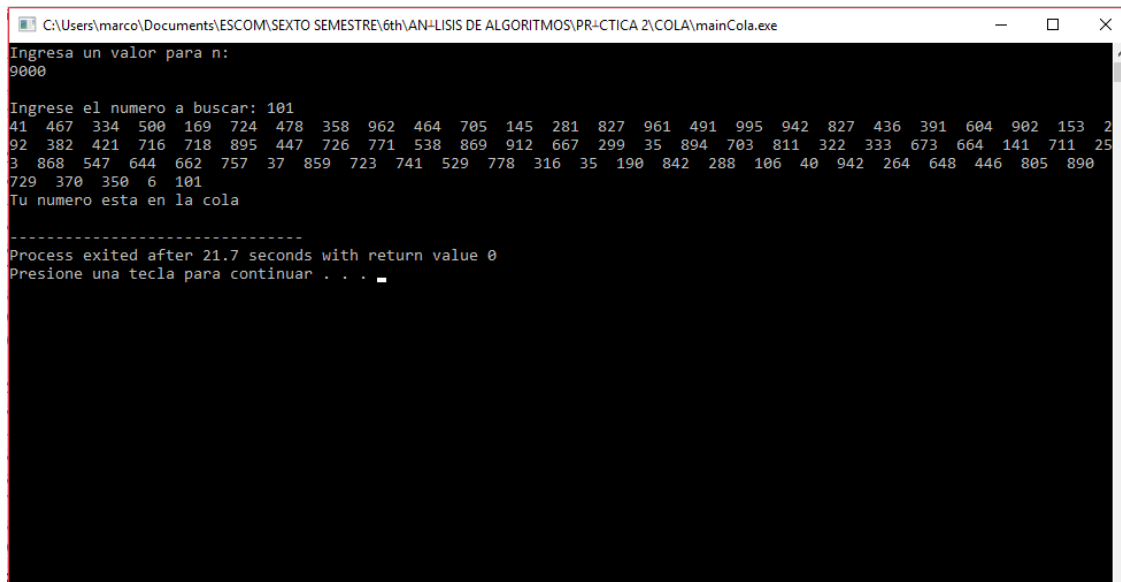
```

```

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;
139.
140.         return r;
141.     }
142.
143.     void Destroy (lista *l){
144.         while (l->tamano > 0){
145.             EliminarInicio (l);
146.         }
147.     }
148.
149.     int Size(lista *l){
150.         return l->tamano;
151.     }
152.
153.     int empty(lista *l)
154.     {
155.         return l->tamano == 0;
156.     }
157.
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.     }
171.
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.         }
180.
181.     }

```

Funcionamiento

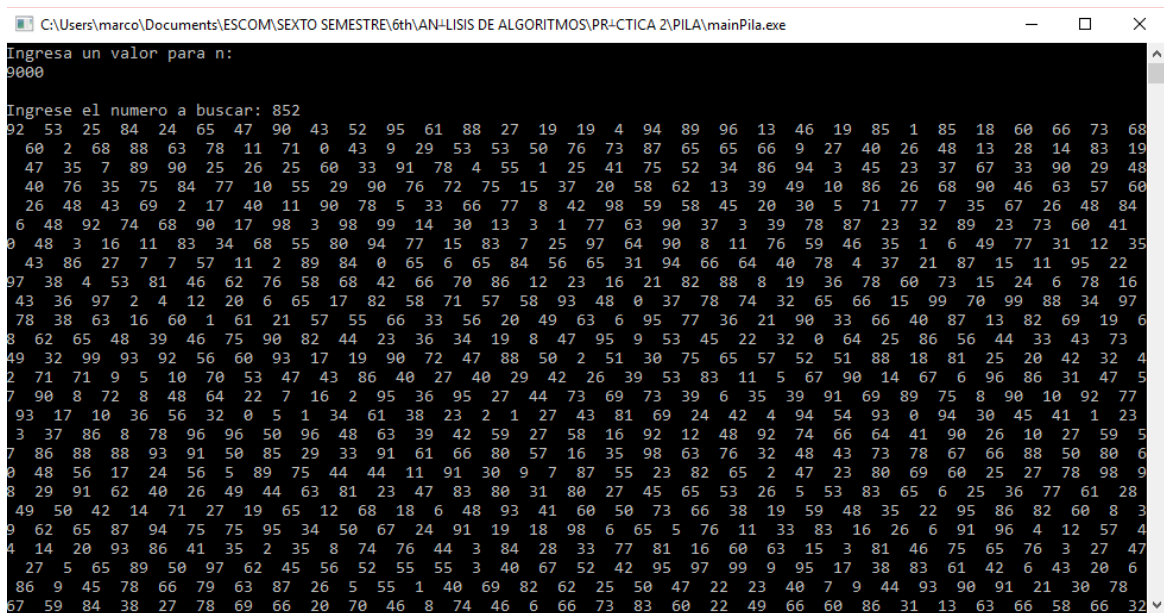


```
C:\Users\marco\Documents\ESCOM\SEXTO SEMESTRE\6th\ANÁLISIS DE ALGORITMOS\PRÁCTICA 2\COLA\mainCola.exe
Ingresar un valor para n:
9000

Ingresar el numero a buscar: 101
41 467 334 500 169 724 478 358 962 464 705 145 281 827 961 491 995 942 827 436 391 604 902 153 2
92 382 421 716 718 895 447 726 771 538 869 912 667 299 35 894 703 811 322 333 673 664 141 711 25
3 868 547 644 662 757 37 859 723 741 529 778 316 35 190 842 288 106 40 942 264 648 446 805 890
729 370 350 6 101
Tu numero esta en la cola

-----
Process exited after 21.7 seconds with return value 0
Presione una tecla para continuar . . . _
```

Imagen 2.1 Prueba Cola



```
C:\Users\marco\Documents\ESCOM\SEXTO SEMESTRE\6th\ANÁLISIS DE ALGORITMOS\PRÁCTICA 2\PILA\mainPila.exe
Ingresar un valor para n:
9000

Ingresar el numero a buscar: 852
92 53 25 84 24 65 47 90 43 52 95 61 88 27 19 19 4 94 89 96 13 46 19 85 1 85 18 60 66 73 68
60 2 68 88 63 78 11 71 0 43 9 29 53 53 50 76 73 87 65 65 66 9 27 40 26 48 13 28 14 83 19
47 35 7 89 90 25 26 25 60 33 91 78 4 55 1 25 41 75 52 34 86 94 3 45 23 37 67 33 90 29 48
40 76 35 75 84 77 10 55 29 90 76 72 75 15 37 20 58 62 13 39 49 10 86 26 68 90 46 63 57 60
26 48 43 69 2 17 40 11 90 78 5 33 66 77 8 42 98 59 58 45 20 30 5 71 77 7 35 67 26 48 84
6 48 92 74 68 90 17 98 3 98 99 14 30 13 3 1 77 63 90 37 3 39 78 87 23 32 89 23 73 60 41
0 48 3 16 11 83 34 68 55 80 94 77 15 83 7 25 97 64 90 8 11 76 59 46 35 1 6 49 77 31 12 35
43 86 27 7 7 57 11 2 89 84 0 65 6 65 84 56 65 31 94 66 64 40 78 4 37 21 87 15 11 95 22
97 38 4 53 81 46 62 76 58 68 42 66 70 86 12 23 16 21 82 88 8 19 36 78 60 73 15 24 6 78 16
43 36 97 2 4 12 20 6 65 17 82 58 71 57 58 93 48 0 37 78 74 32 65 66 15 99 70 99 88 34 97
78 38 63 16 60 1 61 21 57 55 66 33 56 20 49 63 6 95 77 36 21 90 33 66 40 87 13 82 69 19 6
8 62 65 48 39 46 75 90 82 44 23 36 34 19 8 47 95 9 53 45 22 32 0 64 25 86 56 44 33 43 73
49 32 99 93 92 56 60 93 17 19 90 72 47 88 50 2 51 30 75 65 57 52 51 88 18 81 25 20 42 32 4
2 71 71 9 5 10 70 53 47 43 86 40 27 40 29 42 26 39 53 83 11 5 67 90 14 67 6 96 86 31 47 5
7 90 8 72 8 48 64 22 7 16 2 95 36 95 27 44 73 69 73 39 6 35 39 91 69 89 75 8 90 10 92 77
93 17 10 36 56 32 0 5 1 34 61 38 23 2 1 27 43 81 69 24 42 4 94 54 93 0 94 30 45 41 1 23
3 37 86 8 78 96 96 50 96 48 63 39 42 59 27 58 16 92 12 48 92 74 66 64 41 90 26 10 27 59 5
7 86 88 88 93 91 50 85 29 33 91 61 66 80 57 16 35 98 63 76 32 48 43 73 78 67 66 88 50 80 6
9 48 56 17 24 56 5 89 75 44 44 11 91 30 9 7 87 55 23 82 65 2 47 23 80 69 60 25 27 78 98 9
8 29 91 62 40 26 49 44 63 81 23 47 83 80 31 80 27 45 65 53 26 5 53 83 65 6 25 36 77 61 28
49 50 42 14 71 27 19 65 12 68 18 6 48 93 41 60 50 73 66 38 19 59 48 35 22 95 86 82 60 8 3
9 62 65 87 94 75 75 95 34 50 67 24 91 19 18 98 6 65 5 76 11 33 83 16 26 6 91 96 4 12 57 4
4 14 20 93 86 41 35 2 35 8 74 76 44 3 84 28 33 77 81 16 60 63 15 3 81 46 75 65 76 3 27 47
27 5 65 89 50 97 62 45 56 52 55 55 3 40 67 52 42 95 97 99 9 95 17 38 83 61 42 6 43 20 6
86 9 45 78 66 79 63 87 26 5 55 1 40 69 82 62 25 50 47 22 23 40 7 9 44 93 90 91 21 30 78
67 59 84 38 27 78 69 66 20 70 46 8 74 46 6 66 73 83 60 22 49 66 60 86 31 13 63 66 58 66 32
```

Imagen 2.2 Prueba Pila


```
C:\Users\marco\Documents\ESCOM\SEXTO SEMESTRE\6th\AN-LISIS DE ALGORITMOS\PR-CTICA 2\PILA\mainPila.exe
26 70 21 25 59 75 41 11 54 42 23 38 75 48 67 35 11 83 7 63 32 69 57 72 54 34 84 62 53 55
96 32 59 65 29 29 23 41 85 96 17 12 17 22 96 61 69 71 37 57 26 57 26 16 56 85 23 50 72 56
35 4 11 40 5 84 93 28 60 50 1 87 13 8 77 24 57 60 3 39 58 0 77 7 65 78 58 13 0 29 62 51
40 63 88 24 90 20 53 47 11 71 11 26 30 16 2 60 64 25 95 61 37 96 23 62 2 20 23 28 29 27 25
74 77 87 65 86 82 9 64 58 43 27 76 15 70 57 10 26 60 87 0 52 19 64 18 84 51 93 78 66 71
2 58 94 76 25 19 24 95 94 40 50 85 67 96 60 11 33 35 70 70 74 13 60 87 0 64 86 22 42 92 31
25 93 5 32 3 81 61 72 23 24 43 25 27 99 2 27 13 80 2 85 92 68 63 70 73 77 18 97 87 70 33
59 72 74 34 24 14 14 77 25 2 91 58 5 87 18 68 22 74 13 86 49 45 10 54 24 53 39 92 59 44
66 73 50 35 61 29 22 96 44 81 82 46 58 21 56 13 45 32 8 12 53 85 55 13 63 86 71 16 1 14 34
55 82 88 5 86 16 97 95 93 49 66 37 49 29 87 96 79 23 39 53 18 60 45 12 41 49 34 55 25 92
68 29 76 89 21 54 69 32 32 57 17 26 88 85 82 85 2 23 55 89 1 88 61 69 3 10 0 12 75 89 92
76 57 99 21 51 22 98 81 16 44 42 69 33 75 86 62 46 28 55 72 34 2 11 56 88 15 91 58 57 90
79 38 92 38 22 72 57 9 98 89 81 98 96 80 18 58 0 48 3 14 87 23 43 93 85 95 2 89 9 44 8 24
3 98 56 19 49 0 51 35 16 9 14 13 17 10 21 7 83 48 93 28 67 27 88 0 38 18 99 53 34 81 84
68 99 48 21 96 20 74 36 91 50 2 41 95 83 48 37 24 59 10 55 62 91 0 68 13 30 6 46 22 88 21
58 9 9 45 83 53 87 57 37 7 91 7 30 66 24 41 50 50 52 31 74 55 67 55 36 61 90 86 12 97 73
77 29 70 22 24 45 21 86 73 6 77 30 4 58 39 15 33 41 29 82 18 38 37 23 26 39 44 8 31 76 6
6 40 56 54 84 23 29 48 93 1 6 50 70 29 90 5 46 48 64 42 40 6 88 42 90 35 16 78 29 41 23 5
9 37 57 62 44 47 68 53 11 41 64 73 33 22 11 3 94 35 99 67 12 69 38 71 26 47 95 18 16 21 82
92 53 2 4 91 36 27 42 95 91 61 27 81 45 5 64 62 58 78 24 69 0 34 67 41
Tu numero no esta en la pila

-----
Process exited after 23.48 seconds with return value 0
Presione una tecla para continuar . . .
```

Imagen 2.3 Prueba Pila

```
C:\Users\marco\Documents\Downloads\mainLista.exe
Ingresa un valor para n: 100000
Ingresa el numero a buscar: 1000
629 253 280 801 777 500 951 554 501 0 761 302 560 251 556 521 239 345 135 78 637 195 181 507 347
2 100 7 630 731 924 10 8 625 732 621 80 498 57 815 251 496 732 87 291 604 161 156 29 926 550 10
3 551 226 774 694 777 573 463 683 819 551 432 117 744 873 230 998 369 2 345 718 421 709 18 63 9
91 862 312 905 599 686 779 303 526 763 897 970 68 364 742 829 260 501 689 402 90 286 612 571 419
4 106 748 700 294 77 714 441 290 694 56 955 40 822 992 583 601 728 82 617 267 826 508 269 768 45
2 262 895 127 720 398 949 607 721 153 402 184 943 100 653 700 323 641 825 703 975 965 775 235 16
0 353 539 110 239 912 579 623 141 920 69 14 902 924 418 540 16 953 294 715 524 271 579 198 157
123 333 252 424 898 463 113 588 704 776 110 58 993 479 729 487 471 157 383 433 414 787 450 646 5
08 682 53 444 445 619 455 165 293 430 127 394 806 376 853 331 861 664 321 612 448 209 991 971 69
1 757 19 88 747 544 801 597 450 236 144 270 724 317 768 875 681 133 385 981 595 992 462 270 362
0 907 114 101 21 921 168 632 92 303 945 820 620 858 450 338 147 926 548 907 797 969 664 534 13 52
0 553 920 287 241 644 236 637 41 934 436 112 287 253 231 922 58 265 684 617 183 716 599 702 319
6 63 434 775 551 599 183 826 272 544 356 795 265 568 622 694 898 7 633 9 660 397 552 700 885 396
4 145 906 353 311 204 188 814 107 572 356 560 653 653 623 437 509 608 79 658 793 612 743 214 100
118 815 70 680 706 683 714 761 75 847 719 644 160 950 181 103 449 818 866 907 497 878 359 216 33
3 58 994 575 109 495 254 597 952 292 302 878 526 327 72 400 985 847 80 818 572 833 715 556 600
866 630 64 363 333 983 905 898 702 299 563 113 629 880 123 378 782 589 160 635 128 319 362 795 2
04 686 97 32 888 616 745 433 788 925 60 850 469 83 969 776 865 44 403 346 866 326 381 737 286 3
77 925 682 317 415 541 835 100 0 673 155 202 844 798 765 992 37 456 655 502 37 39 614 292 685 3
70 317 126 118 619 320 387 332 965 570 96 616 844 534 304 934 435 581 62 263 852 820 542 8 43 1
93 384 515 59 702 543 894 429 797 833 618 166 53 559 797 1 764 167 314 252 39 703 152 198 975 2
61 33 889 461 549 789 237 904 571 95 54 733 601 255 478 344 914 320 125 736 392 867 447 984 507
197 831 828 611 230 171 793 863 146 671 961 550 84 455 22 254 112 823 509 36 104 108 542 267 25
0 912 578 831 320 418 205 524 321 884 325 618 728 793 746 132 731 270 231 666 740 736 337 130 37
246 774 302 761 304 232 826 567 24 195 172 147 415 196 897 575 563 827 741 190 664 571 603 442
69 824 902 663 48 825 863 176 747 679 440 690 280 46 192 727 964 610 135 324 640 748 348 373 10
1 38 116 72 312 27 550 304 996 700 621 132 823 791 389 664 527 713 378 875 879 639 308 554 84 9
```

Imagen 2.4 Prueba Lista

```

C:\Users\marco\Documents\Downloads\mainLista.exe
487 565 86 882 109 164 758 43 227 576 315 170 757 10 926 60 87 800 152 119 464 18 484 851 593
678 466 371 302 658 694 576 125 19 624 695 694 140 550 285 667 896 760 711 833 235 170 270 974 4
13 360 187 900 64 286 222 142 492 31 725 593 505 432 3 181 61 972 23 924 543 625 527 99 802 627
213 480 102 985 192 668 763 270 773 177 518 297 487 70 833 159 372 874 334 824 314 414 477 625
202 391 958 905 787 18 168 22 474 313 186 649 745 510 154 24 253 439 292 659 44 466 173 450 535
161 129 222 196 144 481 982 646 558 321 756 313 945 832 808 912 53 185 355 313 263 786 671 316
701 114 734 455 282 488 105 286 416 297 195 193 949 866 437 549 529 687 996 279 423 139 753 718
60 145 512 441 549 434 555 425 692 368 329 976 189 721 154 169 932 832 757 617 426 88 285 182 5
85 2 423 255 789 401 688 861 869 3 510 600 712 75 389 892 476 557 699 21 651 322 998 881 416 84
4 142 869 433 875 886 362 646 328 55 272 634 202 511 156 888 815 191 958 657 190 179 38 292 538
622 472 157 9 798 589 281 798 796 580 618 458 200 448 503 314 587 523 343 93 485 195 702 989 60
9 844 8 224 303 798 556 519 249 600 451 935 616 309 514 813 617 310 421 807 483 648 893 728 467
127 788 900 938 418 999 53 734 281 484 668 199 348 21 596 20 374 836 291 350 602 41 595 483 548
537 624 359 410 655 762 591 900 168 413 30 506 946 422 588 221 758 209 909 945 383 753 287 457
337 7 191 107 430 966 724 941 150 350 52 31 574 655 767 355 636 161 290 986 512 97 573 777 829
270 72 924 745 21 386 673 306 977 930 704 658 639 115 833 541 929 82 118 538 537 323 626 439 94
4 308 931 376 966 840 756 954 84 623 629 548 393 101 6 350 370 729 890 805 446 648 264 942 40 1
06 288 842 190 35 316 778 529 741 723 859 37 757 662 644 547 868 253 711 141 664 673 333 322 811
703 894 35 299 667 912 869 538 771 726 447 895 718 716 421 382 292 153 902 604 391 436 827 942
995 491 961 827 281 145 705 464 962 358 478 724 169 500 334 467 41
Tu numero no esta en la lista

-----
Process exited after 39.58 seconds with return value 0
Presione una tecla para continuar . . .

```

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.

Especificaciones del dispositivo	
HP Laptop 15-bs0xx	
Nombre del dispositivo	LAPTOP-13V7QO38
Procesador	Intel(R) Core(TM) i3-6006U CPU @ 2.00GHz 2.00 GHz
RAM instalada	8.00 GB
Id. del dispositivo	ACEA8FAF-1F85-49D4-BC5D-A7059ECE254D
Id. del producto	00327-30000-00000-AAOEM
Tipo de sistema	Sistema operativo de 64 bits, procesador x64
Lápiz y entrada táctil	La entrada táctil o manuscrita no está disponible para esta pantalla

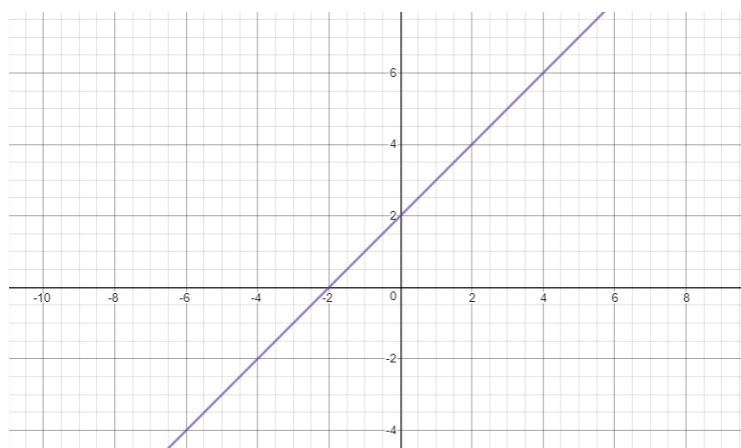
Imagen 3.1 Plataforma Experimental

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

Graficas de funciones

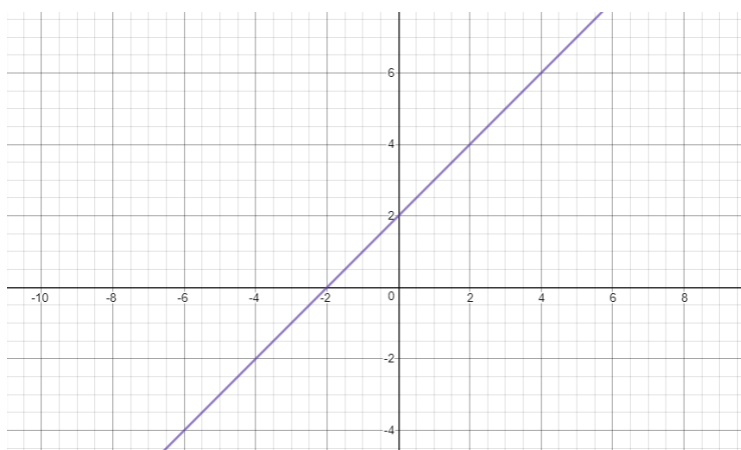
Para la cola, la pila y la lista la gráfica es la misma ya que todas son de función $x=n+2$.

Cola



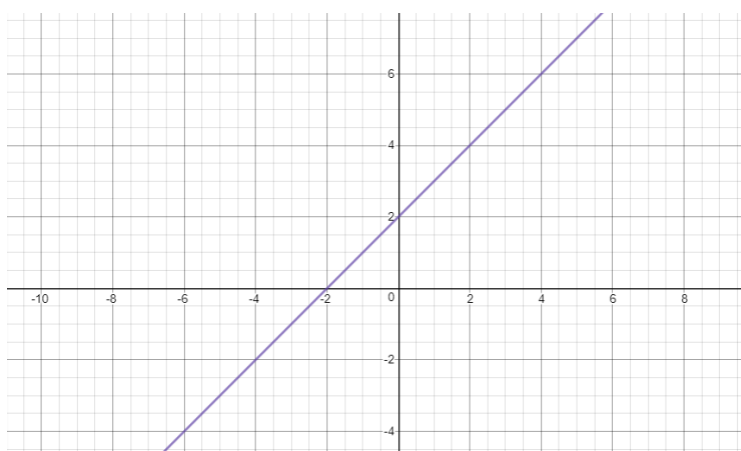
Gráfica 1.1 Cola

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 hacer 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 número 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.