

Universidad Nacional del Altiplano
Facultad de Ingeniería Estadística e Informática
Docente: Fred Torres Cruzz
Alumno: Roberto Angel Ticona Miramira

Trabajo Encargado - N°008

Enlace a repositorio de Github: Repositorio A.D.A-2024-II-UNA en GitHub

1. Ordenamiento por selección.- Tienes un conjunto de valores que representan la cantidad de usuarios activos de una plataforma de redes sociales por día en la última semana: 580, 320, 760, 453, 520. Utiliza el algoritmo de selección para ordenarlos en orden ascendente, de forma que puedas analizar los días de mayor a menor actividad. Al finalizar, indica el número total de comparaciones realizadas.

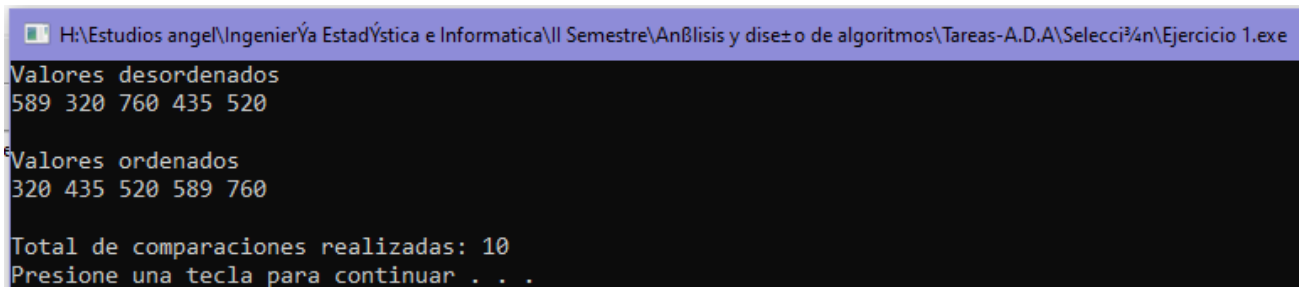
Ejercicio 1.cpp

```
1  // 1. Ordenamiento por selección
2
3  #include <iostream>
4  #include <stdlib.h>
5
6  using namespace std;
7
8  int main () {
9      int lista[] = {589, 320, 760, 435, 520};
10
11      int i,min,j,aux;
12      int comparaciones = 0;
13
14      //Imprimir lista DESORDENADA
15
16      cout << "Valores desordenados\n";
17
18      for (i = 0; i < 5; i++) {
19          cout << lista[i] << " ";
20      }
21
22      //Algoritmo de ordenamiento por selección
23
24      for (i = 0; i < 5; i++) {
25          min = i;
26          for (j = i+1; j < 5; j++) {
27              comparaciones++;
28              //condicion para encontrar el menor elemento
29              if (lista[j] < lista[min]){
30                  min = j;
31              }
32          }
33      }
```

Figura 1: Código O.Selección.1

```
32     }
33     aux = lista[i];
34     lista[i] = lista[min];
35     lista[min] = aux;
36 }
37
38 //Imprimir lista ORDENADA
39
40 cout << "\n\nValores ordenados\n";
41
42 for (i = 0; i < 5; i++) {
43     cout << lista[i] << " ";
44 }
45
46 cout << "\n\nTotal de comparaciones realizadas: " << comparaciones << endl;
47
48 system("pause");
49
50 return 0;
51 }
```

Figura 2: Código O.Selección.2



```
H:\Estudios angel\Ingeniería Estadística e Informática\II Semestre\Análisis y diseño de algoritmos\Tareas-A.D.A\Selección\4n\Ejercicio 1.exe
Valores desordenados
589 320 760 435 520

Valores ordenados
320 435 520 589 760

Total de comparaciones realizadas: 10
Presione una tecla para continuar . . .
```

Figura 3: Resultado O.Selección

2. Ordenamiento por burbuja.- Una empresa tecnológica tiene datos de tiempos de respuesta en milisegundos de su servidor para cinco solicitudes recientes: 125, 90, 150, 105, 80. Usa el algoritmo de burbuja para ordenar estos tiempos de respuesta en orden ascendente, lo que permitirá identificar y optimizar las solicitudes más lentas. Indica el número de intercambios realizados durante el proceso.

Ejercicio 2.cpp

```
1  // 2. Ordenamiento por burbuja
2
3  #include <iostream>
4  #include <stdlib.h>
5
6  using namespace std;
7
8  void bubbleSort(int arr[], int n, int &intercambios) {
9      for (int i = 0; i < n - 1; i++) {
10         for (int j = 0; j < n - i - 1; j++) {
11             if (arr[j] > arr[j + 1]) {
12                 swap(arr[j], arr[j + 1]);
13                 intercambios++;
14             }
15         }
16     }
17 }
18
19 int main() {
20     int arr[] = {125, 90, 150, 105, 80};
21     int n = sizeof(arr) / sizeof(arr[0]);
22     int intercambios = 0;
23
24     cout << "Lista desordenada: " << endl;
25     for (int i = 0; i < n; i++) {
26         cout << arr[i] << " ";
27     }
28
29 }
```

Figura 4: Código O.Burbuja.1

```
30     bubbleSort(arr, n, intercambios);
31
32     cout << "\n\nLista ordenada: " << endl;
33     for (int i = 0; i < n; i++) {
34         cout << arr[i] << " ";
35     }
36
37
38     cout << "\n\nTotal de intercambios: " << intercambios << endl;
39
40     system("pause");
41     return 0;
42 }
43
```

Figura 5: Código O.Burbuja.2

```
H:\Estudios angel\Ingeniería Estadística e Informática\II Semestre\Análisis y diseño de algoritmos\Tareas-A.D.A\Burbuja\Ejercicio 2.exe
Lista desordenada:
125 90 150 105 80

Lista ordenada:
80 90 105 125 150

Total de intercambios: 7
Presione una tecla para continuar . . .
```

Figura 6: Resultado O.Burbuja

3. Inserción directa.- En un análisis de marketing digital, tienes la cantidad de clics que recibieron diferentes anuncios en redes sociales en un día: 250, 120, 300, 95, 210. Utiliza el algoritmo de inserción directa para ordenar los valores en orden ascendente, permitiéndote visualizar cuáles anuncios generaron mayor interacción.

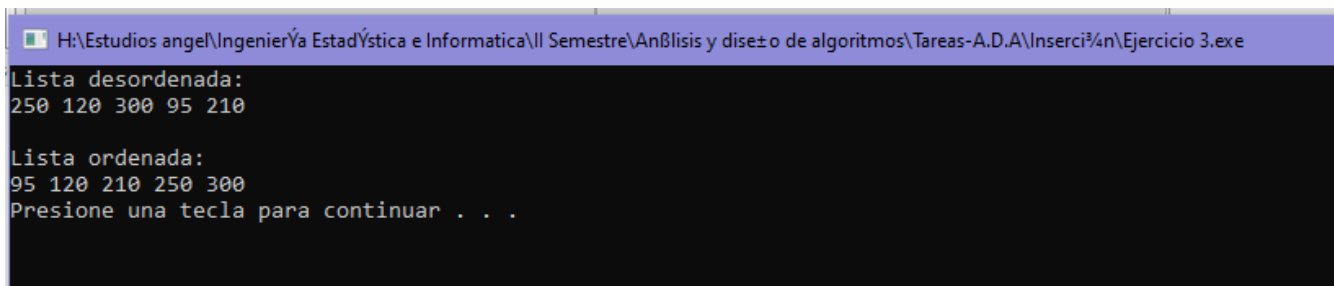
Ejercicio 3.cpp

```
1 // 3. Ordenamiento por inserción directa
2
3 #include <iostream>
4 #include <stdlib.h>
5
6 using namespace std;
7
8 void insertionSort(int arr[], int n) {
9     for (int i = 1; i < n; i++) {
10         int key = arr[i];
11         int j = i - 1;
12         while (j >= 0 && arr[j] > key) {
13             arr[j + 1] = arr[j];
14             j--;
15         }
16         arr[j + 1] = key;
17     }
18 }
19
```

Figura 7: Código Inserción directa.1

```
20 int main() {
21     int arr[] = {250, 120, 300, 95, 210};
22     int n = sizeof(arr)/sizeof(arr[0]);
23
24     cout << "Lista desordenada: " << endl;
25
26     for (int i = 0; i < n; i++) {
27         cout << arr[i] << " ";
28     }
29     insertionSort(arr,n);
30
31     cout << "\n\nLista ordenada: " << endl;
32
33     for (int i = 0; i < n; i++) {
34         cout << arr[i] << " ";
35     }
36     cout << endl;
37
38     system("pause");
39     return 0;
40 }
```

Figura 8: Código Inserción directa.2



```
H:\Estudios angel\Ingeniería Estadística e Informática\II Semestre\Análisis y diseño de algoritmos\Tareas-A.D.A\Inserción\Ejercicio 3.exe
Lista desordenada:
250 120 300 95 210

Lista ordenada:
95 120 210 250 300
Presione una tecla para continuar . . .
```

Figura 9: Resultado Inserción directa

4. Quicksort.- Un investigador está evaluando el tamaño de diferentes archivos de datos en MB almacenados en un servidor de análisis de big data: 850, 230, 690, 540, 310. Utiliza el algoritmo de quicksort para ordenar los tamaños de los archivos en un orden ascendente, lo que ayudará a gestionar el espacio de almacenamiento y optimizar la recuperación de archivos grandes.

Ejercicio 4.cpp

```
1  // 4. Ordenamiento rápido (Quicksort)
2
3  #include <iostream>
4  #include <stdlib.h>
5
6  using namespace std;
7
8  int partition(int arr[], int low, int high) {
9      int pivot = arr[high];
10     int i = low - 1;
11     for (int j = low; j < high; j++) {
12         if (arr[j] < pivot) {
13             i++;
14             swap(arr[i], arr[j]);
15         }
16     }
17     swap(arr[i + 1], arr[high]);
18     return i + 1;
19 }
20
21 void quickSort(int arr[], int low, int high) {
22     if (low < high) {
23         int pi = partition(arr, low, high);
24         quickSort(arr, low, pi - 1);
25         quickSort(arr, pi + 1, high);
26     }
27 }
28
```

Figura 10: Código Quicksort.1

```
29 int main() {
30     int arr[] = {850, 230, 690, 540, 310};
31     int n = sizeof(arr)/sizeof(arr[0]);
32
33     cout << "Lista desordenada: " << endl;
34     for (int i = 0; i < n; i++) {
35         cout << arr[i] << " ";
36     }
37
38     cout << "\n\nLista ordenada: " << endl;
39
40     quickSort(arr, 0, n - 1);
41     for (int i = 0; i < n; i++) cout << arr[i] << " ";
42     return 0;
43 }
```

Figura 11: Código Quicksort.2

```
H:\Estudios angel\Ingeniería Estadística e Informática\II Semestre\Análisis y diseño de algoritmos\Tareas-A.D.A\Quicksort\Ejercicio 4.exe
Lista desordenada:
850 230 690 540 310

Lista ordenada:
230 310 540 690 850
-----
Process exited after 0.01914 seconds with return value 0
Presione una tecla para continuar . . .
```

Figura 12: Resultado Quicksort

5. Mergesort.- En un proyecto de análisis de datos, se han recopilado tiempos en segundos de ejecución de diferentes modelos de machine learning: 30.5, 22.3, 45.6, 15.2, 28.4. Aplica el algoritmo de mergesort para ordenar estos tiempos en orden ascendente, lo cual permitirá identificar los modelos más rápidos y eficientes.

```
Ejercicio 5.cpp
1 // 5. Mergesort
2
3 #include <iostream>
4 #include <vector>
5 #include <stdlib.h>
6
7 using namespace std;
8
9
10 void merge(vector<double>& arreglo, int inicio, int mitad, int final) {
11     int elementosIzq = mitad - inicio + 1;
12     int elementosDer = final - mitad;
13
14     vector<double> izquierda(elementosIzq);
15     vector<double> derecha(elementosDer);
16
17     for (int i = 0; i < elementosIzq; i++)
18         izquierda[i] = arreglo[inicio + i];
19     for (int j = 0; j < elementosDer; j++)
20         derecha[j] = arreglo[mitad + 1 + j];
21
22     int i = 0, j = 0, k = inicio;
23     while (i < elementosIzq && j < elementosDer) {
24         if (izquierda[i] <= derecha[j]) {
25             arreglo[k] = izquierda[i];
26             i++;
27         } else {
28             arreglo[k] = derecha[j];
29             j++;
30         }
31         k++;
32     }
33     while (i < elementosIzq)
34         arreglo[k++] = izquierda[i++];
35     while (j < elementosDer)
36         arreglo[k++] = derecha[j++];
37 }
```

Figura 13: Código Mergesort.1

```
35
36
37 while (i < elementosIzq) {
38     arreglo[k] = izquierda[i];
39     i++;
40     k++;
41 }
42
43
44 while (j < elementosDer) {
45     arreglo[k] = derecha[j];
46     j++;
47     k++;
48 }
49 }
50
51
52 void mergeSort(vector<double>& arreglo, int inicio, int final) {
53     if (inicio < final) {
54         int mitad = inicio + (final - inicio) / 2;
55
56         mergeSort(arreglo, inicio, mitad);
57         mergeSort(arreglo, mitad + 1, final);
58
59         merge(arreglo, inicio, mitad, final);
60     }
61 }
62
63
64
```

Figura 14: Código Mergesort.2


```
65
66 void imprimirArreglo(const vector<double>& arreglo) {
67     for (size_t i = 0; i < arreglo.size(); i++) {
68         cout << arreglo[i] << " ";
69     }
70     cout << endl;
71 }
72
73 int main() {
74     vector<double> tiempos;
75     tiempos.push_back(30.5);
76     tiempos.push_back(22.3);
77     tiempos.push_back(45.6);
78     tiempos.push_back(15.2);
79     tiempos.push_back(28.4);
80
81     cout << "Tiempos de ejecución desordenados: ";
82     imprimirArreglo(tiempos);
83
84     mergeSort(tiempos, 0, tiempos.size() - 1);
85
86     cout << "Tiempos de ejecución ordenados (ascendente): ";
87     imprimirArreglo(tiempos);
88
89     system("pause");
90
91     return 0;
92 }
93
94
95
96
```

Figura 15: Código Mergesort.3

```
H:\Estudios angel\Ingeniería Estadística e Informática\II Semestre\Análisis y diseño de algoritmos\Tareas-A.D.A\Mergesort\Ejercicio 5.exe
Tiempos de ejecución desordenados: 30.5 22.3 45.6 15.2 28.4
Tiempos de ejecución ordenados (ascendente): 15.2 22.3 28.4 30.5 45.6
Presione una tecla para continuar . . .
```

Figura 16: Resultado Mergesort