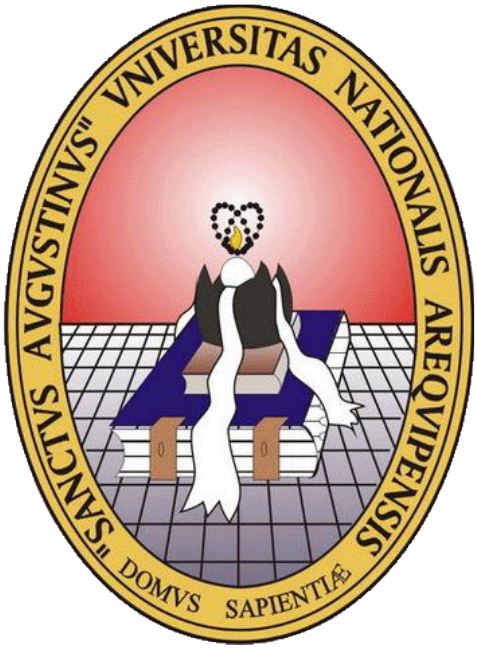
UNIVERSIDAD NACIONAL DE SAN AGUSTÍN DE AREQUIPA
FACULTAD DE PRODUCCIÓN Y SERVICIOS
ESCUELA DE CIENCIAS DE LA COMPUTACIÓN



ALGORITMOS DE ORDENAMIENTO CON PUNTEROS A FUNCIONES

CURSO DE CIENCIAS DE LA COMPUTACIÓN II

ESTUDIANTE:
RUIZ MAMANI EDUARDO GERMÁN

EMAIL: eruizm@unsa.edu.pe
CUI: 20193061

TURNO:
A

AREQUIPA- PERÚ
2021

LINK AL GITHUB: https://github.com/EGRM23/CCII_20193061.git

La carpeta de ORD CON PUNT A FUNCIONES

Use los algoritmos de INSERTSORT, QUICKSORT Y MERGESORT.

Los convertí a funciones de tipo general para que pudieran ordenar cualquier tipo de dato y además agregué que tuvieran como argumento el criterio de ordenamiento

```
template <typename R>
void quickSort(R lista[], bool(*criterio)(R,R), int ini, int fin) {
    int u;
    if (ini < fin) {
        u = ubicar(lista, criterio, ini, fin);
        quickSort(lista, criterio, ini, u - 1);
        quickSort(lista, criterio, u + 1, fin);
    }
}
```

```
template <typename R>
void mergeSort(R lista[], bool(*criterio)(R,R), int ini, int fin) {
    if (ini < fin) {
        int med = (ini+fin) / 2;
        mergeSort(lista, criterio, ini, med);
        mergeSort(lista, criterio, med + 1, fin);
        mezcla(lista, criterio, ini, med, fin);
    }
}
```

```
template <typename R>
void insertSort(R lista[], bool(*criterio)(R,R),  int tam, int aux = 0) {
    R actual;
    int j;
    for(int i = 1 ; i < tam ; i++) {
```

Para el criterio de ordenamiento use dos funciones generales que devolvían datos de tipo bool, las cuáles son “ascendente” y “descendente”

```
template <typename R>
bool ascendente(R v1, R v2) {
    if (v1 >= v2)
        return true;
    else
        return false;
}

template <typename R>
bool descendente(R v1, R v2) {
    if (v1 <= v2)
        return true;
    else
        return false;
}
```

Por último, hice pruebas con los diferentes tipos de datos

Use enteros para el algoritmo QUICKSORT

```
int listaint[] = {1,10,3,8,5,6,7,4,9,2};
tam = sizeof(listaint) / sizeof(listaint[0]);
cout << "Lista de ejemplo: \n";
printLista<int>(listaint, tam);
int auxint[10];

duplicar(listaint, auxint, tam);
quickSort<int>(auxint, ascendente, 0, tam-1);
cout << "Lista ordenada: \n";
printLista<int>(auxint, tam);

duplicar(listaint, auxint, tam);
quickSort<int>(auxint, descendente, 0, tam-1);
cout << "Lista ordenada: \n";
printLista<int>(auxint, tam);
```

```
Lista de ejemplo:
1 10 3 8 5 6 7 4 9 2

Lista ordenada:
1 2 3 4 5 6 7 8 9 10

Lista ordenada:
10 9 8 7 6 5 4 3 2 1
```

Use caracteres para el algoritmo MERGESORT

```
char listachar[] = {'a','j','c','h','e','f','g','d','i','b'};
tam = sizeof(listachar) / sizeof(listachar[0]);
cout << "Lista de ejemplo: \n";
printLista<char>(listachar, tam);
char auxchar[10];

duplicar(listachar, auxchar, tam);
mergeSort<char>(auxchar, ascendente, 0, tam-1);
cout << "Lista ordenada: \n";
printLista<char>(auxchar, tam);

duplicar(listachar, auxchar, tam);
mergeSort<char>(auxchar, descendente, 0, tam-1);
cout << "Lista ordenada: \n";
printLista<char>(auxchar, tam);
```

```
Lista de ejemplo:
a j c h e f g d i b

Lista ordenada:
a b c d e f g h i j

Lista ordenada:
j i h g f e d c b a
```

Y por último use la clase PERSONA

```
class persona {
private:
    string nombre;
    int edad;
public:
    persona() {}
    persona(string n, int e) {
        nombre = n;
        edad = e;
    }
    void operator=(persona &p2) {
        nombre = p2.nombre;
        edad = p2.edad;
    }
    bool operator>=(persona &p2) {return edad>=p2.edad;}
    bool operator<=(persona &p2) {return edad<=p2.edad;}
    void imprimir() {
        cout << nombre << " Edad: " << edad << endl;
    }
};
```

Para el algoritmo INSERTSORT, donde sobrecargué algunos operadores y use como criterio de comparación la edad.

```
persona listapersona[] = {persona("Juan", 20),persona("Carlos", 14),
    persona("Sofia", 25),persona("Sandro", 19),persona("Jeronimo", 22)};
tam = 5;
cout << "Lista de ejemplo: \n";
for (int i = 0; i < 5; i++) {
    listapersona[i].imprimir();
}
persona auxpersona[] = {persona("Juan", 20),persona("Carlos", 14),
    persona("Sofia", 25),persona("Sandro", 19),persona("Jeronimo", 22)};

cout << endl;

insertSort<persona>(listapersona, ascendente, tam);
cout << "Lista ordenada: \n";
for (int i = 0; i < 5; i++) {
    listapersona[i].imprimir();
}

cout << endl;

insertSort<persona>(auxpersona, descendente, tam);
cout << "Lista ordenada: \n";
for (int i = 0; i < 5; i++) {
    auxpersona[i].imprimir();
}
```

```
Lista de ejemplo:
Juan Edad: 20
Carlos Edad: 14
Sofia Edad: 25
Sandro Edad: 19
Jeronimo Edad: 22
```

```
Lista ordenada:
Carlos Edad: 14
Sandro Edad: 19
Juan Edad: 20
Jeronimo Edad: 22
Sofia Edad: 25
```

```
Lista ordenada:
Sofia Edad: 25
Jeronimo Edad: 22
Juan Edad: 20
Sandro Edad: 19
Carlos Edad: 14
```