- Starter kit – Tema 2: Arrays y Algoritmos de Ordenación
  - src/algorithms.py
  - src/bench.py
  - report/report_template.md
  - 🧪 Cómo ejecutar (resumen)

Starter kit – Tema 2: Arrays y Algoritmos de Ordenación

Este documento incluye:

1. **`src/algorithms.py`** – Implementaciones instrumentadas.
2. **`src/bench.py`** – Benchmarks + generación de CSV y gráficas.
3. **`report/report_template.md`** – Plantilla del informe con índice propuesto.

> Requisitos sugeridos: Python 3.10+, `pip install pandas matplotlib`.

---

src/algorithms.py

```python
# src/algorithms.py
# Implementaciones instrumentadas para el Tema 2
# Autor/a: <tu nombre>
# Licencia: uso académico

from dataclasses import dataclass
from typing import List, Tuple
import random

# ------------------
# Métricas comunes
# ------------------
@dataclass
class Metrics:
    comps: int = 0  # comparaciones
    moves: int = 0  # movimientos o swaps, según el algoritmo

# ------------------
# Inserción (estable)
# ------------------

def insertion_sort_instr(a: List[int]) -> Tuple[List[int], Metrics]:
    a = list(a)
```

```
    m = Metrics()
    for i in range(1, len(a)):
        key = a[i]
        j = i - 1
        while j >= 0:
            m.comps += 1
            if a[j] > key:
                a[j+1] = a[j]; m.moves += 1
                j -= 1
            else:
                break
        a[j+1] = key; m.moves += 1
    return a, m

# ----------------
# Selección (clásica, no estable)
# ----------------

def selection_sort_instr(a: List[int]) -> Tuple[List[int], Metrics]:
    a = list(a)
    m = Metrics()
    n = len(a)
    swaps = 0
    for i in range(n-1):
        min_idx = i
        for j in range(i+1, n):
            m.comps += 1
            if a[j] < a[min_idx]:
                min_idx = j
        if min_idx != i:
            a[i], a[min_idx] = a[min_idx], a[i]; swaps += 1
    m.moves = swaps
    return a, m

# ----------------
# Burbuja (+bandera)
# ----------------

def bubble_sort_instr(a: List[int]) -> Tuple[List[int], Metrics]:
    a = list(a)
    m = Metrics()
    n = len(a)
    while True:
        swapped = False
        for j in range(0, n-1):
            m.comps += 1
            if a[j] > a[j+1]:
                a[j], a[j+1] = a[j+1], a[j]
                m.moves += 1
                swapped = True
        n -= 1
        if not swapped or n <= 1:
            break
    return a, m

# ----------------
# Shell (gaps de Knuth)
```

```python
# ----------------

def _knuth_gaps(n: int):
    gaps = []
    h = 1
    while h < n:
        gaps.append(h)
        h = 3*h + 1
    return list(reversed(gaps))


def shell_sort_instr(a: List[int]) -> Tuple[List[int], Metrics]:
    a = list(a)
    m = Metrics()
    n = len(a)
    for gap in _knuth_gaps(n):
        for i in range(gap, n):
            key = a[i]
            j = i
            while j >= gap:
                m.comps += 1
                if a[j-gap] > key:
                    a[j] = a[j-gap]; m.moves += 1
                    j -= gap
                else:
                    break
            a[j] = key; m.moves += 1
    return a, m

# ----------------
# Merge sort (estable) con buffer (in-place sobre el array con copia a buf por tramo)
# ----------------

def merge_sort_instr(a: List[int]) -> Tuple[List[int], Metrics]:
    a = list(a)
    m = Metrics()
    buf = [None] * len(a)

    def ms(lo: int, hi: int):
        if hi - lo <= 1:
            return
        mid = (lo + hi) // 2
        ms(lo, mid); ms(mid, hi)
        # merge
        buf[lo:hi] = a[lo:hi]
        i, j, k = lo, mid, lo
        while i < mid and j < hi:
            m.comps += 1
            if buf[i] <= buf[j]:
                a[k] = buf[i]; i += 1; m.moves += 1
            else:
                a[k] = buf[j]; j += 1; m.moves += 1
            k += 1
        while i < mid:
            a[k] = buf[i]; i += 1; k += 1; m.moves += 1
        # Si quedan del lado derecho, ya están colocados
```

```
    ms(0, len(a))
    return a, m

# ----------------
# Quick sort — Hoare + mediana-de-tres + corte a inserción
# ----------------

def _insertion_tail(a: List[int], lo: int, hi: int):
    for i in range(lo + 1, hi + 1):
        key = a[i]; j = i - 1
        while j >= lo and a[j] > key:
            a[j+1] = a[j]; j -= 1
        a[j+1] = key


def _median3(a: List[int], lo: int, mid: int, hi: int):
    # Ordena (lo, mid, hi) para devolver mediana
    if a[mid] < a[lo]: a[lo], a[mid] = a[mid], a[lo]
    if a[hi]  < a[lo]: a[lo], a[hi]  = a[hi],  a[lo]
    if a[hi]  < a[mid]:a[mid],a[hi]  = a[hi],  a[mid]
    return a[mid]


def quick_hoare_instr(a: List[int], cutoff: int = 16) -> Tuple[List[int],
Metrics]:
    a = list(a)
    m = Metrics()

    def part(lo: int, hi: int) -> int:
        mid = (lo + hi) // 2
        pivot = _median3(a, lo, mid, hi)
        i, j = lo - 1, hi + 1
        while True:
            i += 1
            while a[i] < pivot:
                m.comps += 1; i += 1
            m.comps += 1
            j -= 1
            while a[j] > pivot:
                m.comps += 1; j -= 1
            m.comps += 1
            if i >= j:
                return j
            a[i], a[j] = a[j], a[i]; m.moves += 1

    def qs(lo: int, hi: int):
        while lo < hi:
            if hi - lo + 1 <= cutoff:
                _insertion_tail(a, lo, hi)
                return
            p = part(lo, hi)
            if p - lo < hi - (p + 1):
                qs(lo, p)
                lo = p + 1
            else:
                qs(p + 1, hi)
```

```
                hi = p

    if a:
        qs(0, len(a) - 1)
        _insertion_tail(a, 0, len(a) - 1)  # pulido final
    return a, m

# ----------------
# Quick sort — 3-way (ideal con muchos duplicados)
# ----------------

def quick_3way_instr(a: List[int]) -> Tuple[List[int], Metrics]:
    a = list(a)
    m = Metrics()

    def qs(lo: int, hi: int):
        while lo < hi:
            p = random.randint(lo, hi)
            a[lo], a[p] = a[p], a[lo]; m.moves += 1
            pivot = a[lo]
            lt, i, gt = lo, lo + 1, hi
            while i <= gt:
                m.comps += 1
                if a[i] < pivot:
                    a[lt], a[i] = a[i], a[lt]; m.moves += 1
                    lt += 1; i += 1
                elif a[i] > pivot:
                    a[i], a[gt] = a[gt], a[i]; m.moves += 1
                    gt -= 1
                else:
                    i += 1
            if (lt - lo) < (hi - gt):
                qs(lo, lt - 1)
                lo = gt + 1
            else:
                qs(gt + 1, hi)
                hi = lt - 1

    if a:
        qs(0, len(a) - 1)
    return a, m

# ----------------
# Heap sort (max-heap, in-place)
# ----------------

def heapsort_instr(a: List[int]) -> Tuple[List[int], Metrics]:
    a = list(a)
    m = Metrics()
    n = len(a)

    def sift(i: int, end: int):
        while True:
            l = 2 * i + 1
            r = l + 1
            if l >= end:
                break
```

```
            j = l
            if r < end:
                m.comps += 1
                if a[r] > a[l]:
                    j = r
            m.comps += 1
            if a[i] >= a[j]:
                break
            a[i], a[j] = a[j], a[i]; m.moves += 1
            i = j

    # heapify (Floyd)
    for i in range((n - 2) // 2, -1, -1):
        sift(i, n)

    # sortdown
    for end in range(n - 1, 0, -1):
        a[0], a[end] = a[end], a[0]; m.moves += 1
        sift(0, end)
    return a, m

# ----------------
# Banco público para importación
# ----------------
ALGOS = {
    "insertion": insertion_sort_instr,
    "selection": selection_sort_instr,
    "bubble": bubble_sort_instr,
    "shell": shell_sort_instr,
    "merge": merge_sort_instr,
    "quick": quick_hoare_instr,
    "quick3": quick_3way_instr,
    "heap": heapsort_instr,
}
```

📄 src/bench.py

```
# src/bench.py
# Runner de experimentos para el Tema 2
# Genera CSV y gráficas (PNG) en /results
# Autor/a: <tu nombre>

import argparse
import os
import time
import random
from typing import List, Dict
import pandas as pd
import matplotlib.pyplot as plt
```

```python
from algorithms import ALGOS

# ----------------
# Generadores de datos
# ----------------

def gen_random(n: int, seed: int = 123) -> List[int]:
    rnd = random.Random(seed)
    return [rnd.randint(0, 10**6) for _ in range(n)]

def gen_sorted(n: int) -> List[int]:
    return list(range(n))

def gen_reverse(n: int) -> List[int]:
    return list(range(n, 0, -1))

def gen_nearly_sorted(n: int) -> List[int]:
    a = gen_sorted(n)
    k = max(1, n // 100)  # ~1% de pares
    for i in range(k):
        i1 = (37 * i + 13) % n
        i2 = (53 * i + 7) % n
        a[i1], a[i2] = a[i2], a[i1]
    return a

def gen_many_dups(n: int, distinct: int = 10) -> List[int]:
    rnd = random.Random(42)
    return [rnd.randint(0, distinct - 1) for _ in range(n)]

INPUTS = {
    "random": gen_random,
    "sorted": gen_sorted,
    "reverse": gen_reverse,
    "nearly": gen_nearly_sorted,
    "dups": gen_many_dups,
}

# ----------------
# Bench principal
# ----------------

def run_bench(sizes=(500, 2000, 8000, 20000), reps=3,
              algos=("insertion","shell","merge","quick","quick3","heap"),
              inputs=("random","sorted","reverse","nearly","dups"),
              seed=123) -> pd.DataFrame:
    rows = []
    for n in sizes:
        for dist in inputs:
            gen = INPUTS[dist]
            for r in range(reps):
                base = gen(n) if dist != "random" else gen(n, seed=seed + r)
                for name in algos:
                    fn = ALGOS[name]
                    arr = list(base)
                    t0 = time.perf_counter()
                    out, m = fn(arr)
                    t1 = time.perf_counter()
```

```
                    assert out == sorted(base), f"{name} falló en {dist} n=
{n}"
                    rows.append({
                        "algo": name,
                        "n": n,
                        "dist": dist,
                        "rep": r,
                        "time_ms": (t1 - t0) * 1000.0,
                        "comps": m.comps,
                        "moves": m.moves,
                    })
    return pd.DataFrame(rows)

# ----------------
# Gráficas
# ----------------

def plot_time_vs_n(df: pd.DataFrame, outdir: str):
    os.makedirs(outdir, exist_ok=True)
    # una figura por distribución
    for dist in sorted(df["dist"].unique()):
        d = (df[df["dist"] == dist]
               .groupby(["algo","n"]).agg(time_ms=
("time_ms","mean")).reset_index())
        plt.figure()
        for algo in sorted(d["algo"].unique()):
            sub = d[d["algo"] == algo]
            plt.plot(sub["n"], sub["time_ms"], marker="o", label=algo)
        plt.title(f"Tiempo vs n — {dist}")
        plt.xlabel("n"); plt.ylabel("tiempo (ms)")
        plt.legend(); plt.grid(True, alpha=0.3)
        plt.tight_layout()
        plt.savefig(os.path.join(outdir, f"time_vs_n_{dist}.png"))
        plt.close()


def plot_ops_vs_n(df: pd.DataFrame, outdir: str):
    os.makedirs(outdir, exist_ok=True)
    d = (df[df["dist"] == "random"].groupby(["algo","n"]).agg(
        comps=("comps","mean"), moves=("moves","mean")
    ).reset_index())
    for metric in ("comps","moves"):
        plt.figure()
        for algo in sorted(d["algo"].unique()):
            sub = d[d["algo"] == algo]
            plt.plot(sub["n"], sub[metric], marker="o", label=algo)
        plt.title(f"{metric} vs n — random")
        plt.xlabel("n"); plt.ylabel(metric)
        plt.legend(); plt.grid(True, alpha=0.3)
        plt.tight_layout()
        plt.savefig(os.path.join(outdir, f"{metric}_vs_n_random.png"))
        plt.close()


def plot_quick_vs_quick3_dups(df: pd.DataFrame, outdir: str):
    os.makedirs(outdir, exist_ok=True)
    d = (df[(df["dist"] == "dups") & (df["algo"].isin(["quick","quick3"]))]
```

```
            .groupby(["algo","n"]).agg(time_ms=
("time_ms","mean")).reset_index())
    plt.figure()
    for algo in ("quick","quick3"):
        sub = d[d["algo"] == algo]
        plt.plot(sub["n"], sub["time_ms"], marker="o", label=algo)
    plt.title("Quick vs Quick 3-way — dups")
    plt.xlabel("n"); plt.ylabel("tiempo (ms)")
    plt.legend(); plt.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.savefig(os.path.join(outdir, "quick_vs_quick3_dups.png"))
    plt.close()

# ----------------
# CLI
# ----------------

def main():
    parser = argparse.ArgumentParser(description="Bench Tema 2 —
Ordenación")
    parser.add_argument("action", choices=["run","plot","runplot"],
help="Qué hacer")
    parser.add_argument("--outdir", default="results", help="Directorio de
salida")
    parser.add_argument("--csv", default="sorting_results.csv", help="Nombre
de CSV")
    parser.add_argument("--sizes", default="500,2000,8000,20000",
help="Tamaños separados por coma")
    parser.add_argument("--reps", type=int, default=3, help="Repeticiones")
    args = parser.parse_args()

    sizes = tuple(int(x) for x in args.sizes.split(","))
    csv_path = os.path.join(args.outdir, args.csv)

    if args.action in ("run","runplot"):
        os.makedirs(args.outdir, exist_ok=True)
        df = run_bench(sizes=sizes, reps=args.reps)
        df.to_csv(csv_path, index=False)
        print(f"CSV guardado en {csv_path}")

    if args.action in ("plot","runplot"):
        df = pd.read_csv(csv_path)
        figs_dir = os.path.join(args.outdir, "figs")
        plot_time_vs_n(df, figs_dir)
        plot_ops_vs_n(df, figs_dir)
        plot_quick_vs_quick3_dups(df, figs_dir)
        print(f"Gráficas guardadas en {figs_dir}")

if __name__ == "__main__":
    main()
```

---

report/report_template.md

```
# Título del trabajo — Tema 2: Arrays y Algoritmos de Ordenación
Autor/a: <tu nombre>
Asignatura: <asignatura>
Fecha: <fecha>

## Enlace al vídeo (obligatorio)
- URL YouTube (no listado o público), duración ≥ 10 minutos.
- Debe verse tu cara en miniatura y la ejecución **en vivo** desde VS Code.

## 1. Introducción
Contexto, objetivos y por qué interesan **estabilidad**, **in-place**,
**constantes** y **patrones de entrada**.

## 2. Metodología
- Hardware/Software (CPU, RAM, SO, Python, VS Code).
- Algoritmos implementados e **instrumentación** (qué se cuenta y cómo).
- Diseño experimental: tamaños, distribuciones, repeticiones, control de
ruido.
- Comandos usados:
  - `python src/bench.py runplot --sizes 500,2000,8000,20000 --reps 3`

## 3. Trazas manuales
Usa `A = [5, 2, 4, 6, 1, 3, 4]` para:
- Inserción (mostrar estabilidad con duplicados).
- Selección (mostrar inestabilidad potencial).
- Burbuja (+bandera y mejor caso).

## 4. Resultados experimentales
- Tabla resumen (tiempo, comps, moves) por (algo, dist, n).
- **Gráficas** (incluye PNG en `/results/figs`):
  - Tiempo vs n por distribución.
  - Comparaciones y movimientos vs n (random).
  - Quick vs Quick 3-way (dups).
- Comenta anomalías u observaciones destacadas.

## 5. Discusión
- Constantes y **localidad de caché**: quick vs heap.
- **Estabilidad**: cuándo impone usar merge.
- **Memoria**: buffer de merge vs in-place (quick/heap).
- **Duplicados**: utilidad de 3-way.
- **Híbridos**: cortes a inserción; detección de runs.

## 6. Variantes (elige dos)
Describe, justifica y compara con evidencia:
1. Selection **estable** (sin swaps, con desplazamientos) y su impacto en
movimientos.
2. Shell con otra secuencia de gaps (Hibbard/Tokuda/Pratt).
3. Quick **mediana-de-tres** vs **aleatorio**, y profundidad de recursión.
4. Merge **bottom-up** con pequeñas optimizaciones (mini-Timsort).
5. Ordenar **índices** para objetos pesados.

## 7. Conclusiones
Reglas prácticas de elección según requisitos (estabilidad, memoria,
duplicados, tamaño).

## 8. Reproducibilidad
```

```
- Versiones, semilla, comandos, estructura del repo.
- Checklist: asserts, CSV, gráficos, vídeo.

## 9. Referencias
Cita recursos consultados (libros, apuntes, blogs técnicos, papers, doc.
oficial).
```

🧪 Cómo ejecutar (resumen)

```
# 1) Crear estructura de carpetas
mkdir -p src results/figs report

# 2) Guardar los archivos anteriores en sus rutas
#    (algorithms.py y bench.py en src/, report_template.md en report/)

# 3) Instalar dependencias
pip install pandas matplotlib

# 4) Correr benchmarks + generar CSV y gráficas
python src/bench.py runplot --sizes 500,2000,8000,20000 --reps 3

# Salidas:
# - results/sorting_results.csv
# - results/figs/*.png
```

> En el **vídeo**, muestra: (a) ejecución del benchmark con VS Code (terminal visible), (b) apertura del CSV, (c) visualización de las gráficas, y (d) una traza manual breve explicada en pantalla.