**Universidad Nacional Del Altiplano**

**Facultad De Ingeniería Mecánica Eléctrica, Electrónica Y Sistemas**

**Escuela Profesional De Ingeniería De Sistemas**

Logotipo, Icono

El contenido generado por IA puede ser incorrecto.Imagen que contiene Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

**Practica Nº2**

NOTACION ASITNOTICA: Algoritmos de búsqueda y conteo en matrices (C++ y

Python)

**CURSO:**

Algoritmos y Estructuras de Datos

**DOCENTE:**

Mg. Aldo Hernan Zanabria Galvez.

**ESTUDIANTE:**

Condori Ticona Anthony

**CODIGO:** 236055

**CODIGO:** 15/04/2025

**SEMESTRE:**

IV

Contenido

[Análisis Comparativo de Algoritmos en Python y C++ 2](#_Toc195613700)

[1. INTRODUCCIÓN: 3](#_Toc195613701)

[2. ANÁLISIS TEÓRICO DE LOS ALGORITMOS: 3](#_Toc195613702)

[2.1. Algoritmo de Conteo de Números Pares: 3](#_Toc195613703)

[2.2. Algoritmo de Conteo de Números Primos: 4](#_Toc195613704)

[3. IMPLEMENTACIÓN EN PYTHON Y C++: 5](#_Toc195613705)

[3.1. Código en Python 5](#_Toc195613706)

[3.2. Código en C++ 6](#_Toc195613707)

[3.3. Estructuras de Datos Utilizadas 8](#_Toc195613708)

[4. COMPLEJIDAD TEMPORAL Y ESPACIAL 8](#_Toc195613709)

[5. RESULTADOS EXPERIMENTALES 9](#_Toc195613710)

[5.1. C++ https://onlinegdb.com/BqolKUVsP 9](#_Toc195613711)

[5.2. Python https://onlinegdb.com/9s55yDXFKy 11](#_Toc195613712)

[5.3 Cuadro Comparativo de Tiempos de Ejecución 13](#_Toc195613713)

[6. ANÁLISIS DE LA COMPARACIÓN DE TIEMPOS 14](#_Toc195613714)

[**Conclusión** 16](#_Toc195613715)

[**Bibliografía** 17](#_Toc195613716)

# Análisis Comparativo de Algoritmos en Python y C++

## INTRODUCCIÓN:

El objetivo de este trabajo es realizar una comparación entre dos lenguajes de programación, Python y C++, en términos de eficiencia en la ejecución de un algoritmo para contar los números pares en una matriz aleatoria. Además, se implementará una función adicional que contará los números primos en la matriz para mejorar el análisis. A través de este informe, se presentarán los resultados experimentales y se explicarán los detalles teóricos detrás de los algoritmos.

## ANÁLISIS TEÓRICO DE LOS ALGORITMOS:

### ****Algoritmo de Conteo de Números Pares:****

Este algoritmo tiene como objetivo contar la cantidad de números pares en una matriz de tamaño n × m. A continuación, se describe cómo funciona el algoritmo:

**Pseudocódigo:**

Inicio

Para cada fila en la matriz

Para cada valor en la fila

Si el valor es par (valor % 2 == 0)

Aumentar el contador

Fin

Fin

Este algoritmo recorre cada elemento de la matriz, verificando si es par. La **complejidad temporal** es **O(n × m)**, donde n es el número de filas y m es el número de columnas de la matriz. Esto se debe a que se requiere visitar cada uno de los elementos de la matriz.

**Complejidad Espacial:** O(n × m), ya que la matriz se almacena en la memoria.

### ****Algoritmo de Conteo de Números Primos:****

El algoritmo de conteo de números primos busca identificar aquellos números en la matriz que son primos. Un número primo es aquel que solo tiene dos divisores: 1 y él mismo.

**Optimización para Verificación de Primalidad:** La verificación de primalidad puede optimizarse mediante el uso de **O(sqrt(n))**, ya que un número solo tiene divisores hasta su raíz cuadrada. La verificación se realiza de la siguiente manera:

* Si el número es menor o igual a 1, no es primo.
* Para números mayores, se verifica si tienen divisores entre 2 y su raíz cuadrada.

**Pseudocódigo para Verificación de Primalidad:**

Inicio

Si n <= 1

No es primo

Para i desde 2 hasta sqrt(n)

Si n % i == 0

No es primo

Fin

Fin

El tiempo de verificación para cada número es O(sqrt(n)), y dado que la matriz contiene n × m números, la **complejidad total** es O(n × m × sqrt(k)), donde k es el valor máximo en la matriz.

## IMPLEMENTACIÓN EN PYTHON Y C++:

### ****Código en Python****

Código en onlinegdb comentado: <https://onlinegdb.com/blEfqU_uG>

Código en onlinegdb sin comentar: <https://onlinegdb.com/zK3_dxc1g>

import random # Importa el módulo para generar números aleatorios

import time # Importa el módulo para medir tiempos de ejecución

import math # Importa funciones matemáticas, como sqrt (raíz cuadrada)

# Función que genera una matriz (lista de listas) de tamaño filas x columnas

def generar\_matriz(filas, columnas):

# Crea una lista de listas con valores aleatorios entre 0 y 100

return [[random.randint(0, 100) for \_ in range(columnas)] for \_ in range(filas)]

# Función que cuenta cuántos números pares hay en la matriz

def contar\_pares(matriz):

conteo = 0 # Inicializa el contador

for fila in matriz: # Recorre cada fila de la matriz

for valor in fila: # Recorre cada valor dentro de la fila

if valor % 2 == 0: # Verifica si el número es par

conteo += 1 # Incrementa el contador si es par

return conteo # Devuelve el total de números pares encontrados

# Función que determina si un número es primo

def es\_primo(n):

if n <= 1: # Los números menores o iguales a 1 no son primos

return False

for i in range(2, int(math.sqrt(n)) + 1): # Recorre desde 2 hasta la raíz cuadrada de n

if n % i == 0: # Si es divisible por algún número, no es primo

return False

return True # Si no fue divisible, es primo

# Función que cuenta cuántos números primos hay en la matriz

def contar\_primos(matriz):

conteo = 0 # Inicializa el contador

for fila in matriz: # Recorre cada fila

for valor in fila: # Recorre cada valor de la fila

if es\_primo(valor): # Verifica si es primo

conteo += 1 # Si lo es, incrementa el contador

return conteo # Devuelve el total de primos encontrados

# Solicita al usuario el número de filas y columnas

filas = int(input("Ingrese el número de filas: "))

columnas = int(input("Ingrese el número de columnas: "))

# Genera una matriz aleatoria con las dimensiones ingresadas

matriz = generar\_matriz(filas, columnas)

# Cuenta los números pares y mide el tiempo de ejecución

inicio = time.time() # Guarda el tiempo inicial

resultado\_pares = contar\_pares(matriz) # Llama a la función para contar pares

tiempo\_pares = time.time() - inicio # Calcula el tiempo transcurrido

# Cuenta los números primos y mide el tiempo de ejecución

inicio = time.time() # Guarda el tiempo inicial

resultado\_primos = contar\_primos(matriz) # Llama a la función para contar primos

tiempo\_primos = time.time() - inicio # Calcula el tiempo transcurrido

# Muestra los resultados en consola

print(f"\nNúmeros pares: {resultado\_pares}")

print(f"Tiempo de ejecución para contar pares: {tiempo\_pares:.6f} segundos")

print(f"Números primos: {resultado\_primos}")

print(f"Tiempo de ejecución para contar primos: {tiempo\_primos:.6f} segundos")

# Guarda los resultados en un archivo de texto llamado 'resultados\_python.txt'

with open("resultados\_python.txt", "a") as archivo: # Abre el archivo en modo añadir

archivo.write(f"\nTamaño de matriz: {filas}x{columnas}\n") # Escribe tamaño de la matriz

archivo.write(f"Números pares: {resultado\_pares}, Tiempo: {tiempo\_pares:.6f} s\n") # Escribe resultado pares

archivo.write(f"Númer

### ****Código en C++****

Código en onlinegdb comentado: <https://onlinegdb.com/Sx5p7HMiMg>

Código en onlinegdb sin comentar: <https://onlinegdb.com/HE6wI3J3A>

#include <iostream> // Para entrada y salida estándar

#include <vector> // Para utilizar vectores (listas dinámicas)

#include <ctime> // Para medir el tiempo de ejecución

#include <cstdlib> // Para generar números aleatorios con rand()

#include <cmath> // Para funciones matemáticas como sqrt()

#include <fstream> // Para trabajar con archivos

using namespace std;

// Función que genera una matriz de tamaño filas x columnas con valores aleatorios

vector<vector<int>> generarMatriz(int filas, int columnas) {

vector<vector<int>> matriz(filas, vector<int>(columnas)); // Crea una matriz de enteros

for (int i = 0; i < filas; ++i) // Recorre las filas

for (int j = 0; j < columnas; ++j) // Recorre las columnas

matriz[i][j] = rand() % 101; // Asigna un número aleatorio de 0 a 100

return matriz; // Devuelve la matriz generada

}

// Función para contar números pares en la matriz

int contarPares(const vector<vector<int>>& matriz) {

int conteo = 0;

for (const auto& fila : matriz) // Recorre cada fila

for (int val : fila) // Recorre cada valor en la fila

if (val % 2 == 0) ++conteo; // Verifica si el valor es par

return conteo; // Devuelve la cantidad de pares

}

// Función para verificar si un número es primo

bool esPrimo(int n) {

if (n <= 1) return false; // Los números <= 1 no son primos

for (int i = 2; i <= sqrt(n); ++i) // Recorre desde 2 hasta la raíz de n

if (n % i == 0) return false; // Si n es divisible por i, no es primo

return true; // Si no se encontró divisor, es primo

}

// Función para contar números primos en la matriz

int contarPrimos(const vector<vector<int>>& matriz) {

int conteo = 0;

for (const auto& fila : matriz) // Recorre cada fila

for (int val : fila) // Recorre cada valor en la fila

if (esPrimo(val)) ++conteo; // Si es primo, aumenta el conteo

return conteo; // Devuelve la cantidad de primos

}

int main() {

srand(time(0)); // Inicializa la semilla para generación aleatoria

int filas, columnas;

// Solicita al usuario el tamaño de la matriz

cout << "Ingrese el número de filas: ";

cin >> filas;

cout << "Ingrese el número de columnas: ";

cin >> columnas;

// Genera la matriz con los tamaños ingresados

vector<vector<int>> matriz = generarMatriz(filas, columnas);

// Medición de tiempo para contar pares

clock\_t inicio = clock(); // Marca el tiempo inicial

int resultadoPares = contarPares(matriz); // Cuenta pares

clock\_t fin = clock(); // Marca el tiempo final

double tiempoPares = double(fin - inicio) / CLOCKS\_PER\_SEC; // Calcula el tiempo

// Medición de tiempo para contar primos

inicio = clock(); // Marca el tiempo inicial

int resultadoPrimos = contarPrimos(matriz); // Cuenta primos

fin = clock(); // Marca el tiempo final

double tiempoPrimos = double(fin - inicio) / CLOCKS\_PER\_SEC; // Calcula el tiempo

// Muestra los resultados en consola

cout << "\nNúmeros pares: " << resultadoPares << endl;

cout << "Tiempo de ejecución para contar pares: " << tiempoPares << " segundos" << endl;

cout << "Números primos: " << resultadoPrimos << endl;

cout << "Tiempo de ejecución para contar primos: " << tiempoPrimos << " segundos" << endl;

// Guarda los resultados en un archivo de texto

ofstream archivo("resultados\_cpp.txt", ios::app); // Abre el archivo en modo agregar

archivo << "\nTamaño de matriz: " << filas << "x" << columnas << endl;

archivo << "Números pares: " << resultadoPares << ", Tiempo: " << tiempoPares << " s" << endl;

archivo << "Números primos: " << resultadoPrimos << ", Tiempo: " << tiempoPrimos << " s" << endl;

archivo.close(); // Cierra el archivo

return 0;

}

### ****Estructuras de Datos Utilizadas****

|  |  |  |  |
| --- | --- | --- | --- |
| **Algoritmo / Función** | **Lenguaje** | **Estructura de Datos Principal** | **Descripción** |
| generar\_matriz | Python | list[list[int]] | Lista de listas (matriz) |
| generarMatriz | C++ | vector<vector<int>> | Vector de vectores (matriz) |
| contar\_pares y primos | Python | list[list[int]] | Recorrido de matriz |
| contarPares y Primos | C++ | vector<vector<int>> | Recorrido de matriz |
| Guardar resultados | Ambos | Archivo de texto (.txt) | Salida en archivo secuencial |

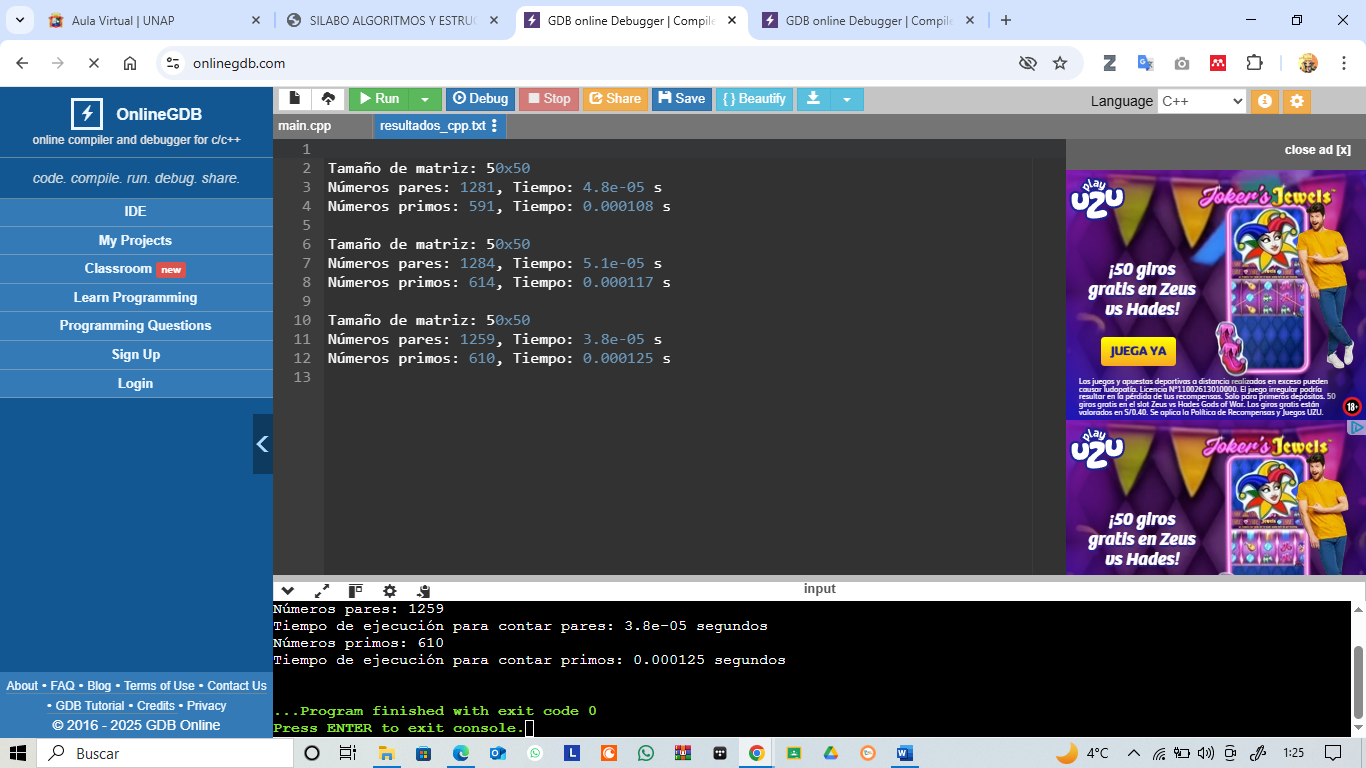
## COMPLEJIDAD TEMPORAL Y ESPACIAL

|  |  |  |
| --- | --- | --- |
| Función | Complejidad Temporal | Complejidad Espacial |
| Generación de matriz | O(n\*m) | O(n\*m) |
| Contar pares | O(n\*m) | O(1) |
| Contar primos | O(n*m*√k) | O(1) |

## RESULTADOS EXPERIMENTALES

### C++ <https://onlinegdb.com/BqolKUVsP>

**Matriz 50x50**



**Matriz 100x100**

Texto

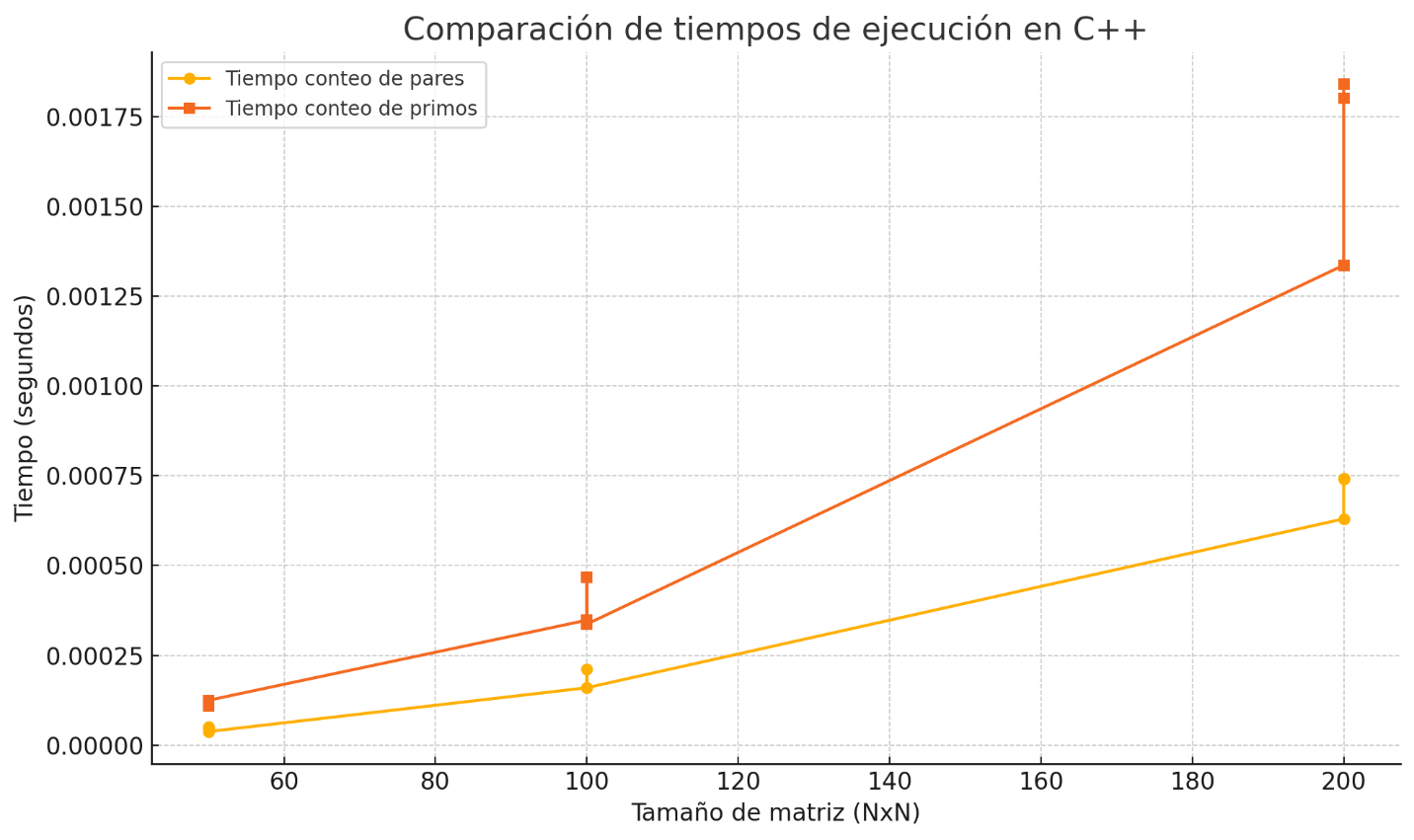
El contenido generado por IA puede ser incorrecto.

**Matriz 200x200**

Interfaz de usuario gráfica, Texto

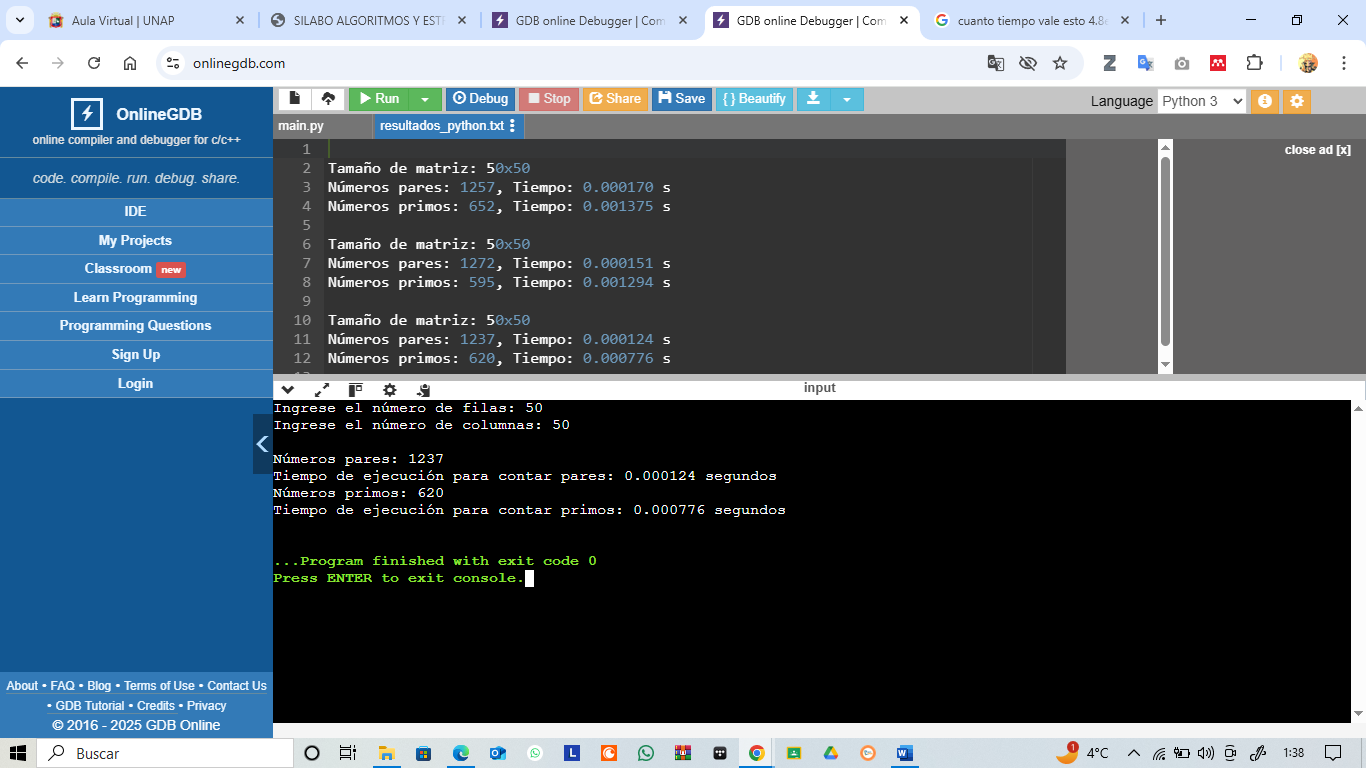
El contenido generado por IA puede ser incorrecto.

**Grafica**



### Python <https://onlinegdb.com/9s55yDXFKy>

**Matriz 50x50**



**Matriz 100x100**

Texto

El contenido generado por IA puede ser incorrecto.

**Matriz 200x200**

Texto

El contenido generado por IA puede ser incorrecto.

**Grafica**

Gráfico, Gráfico de líneas

El contenido generado por IA puede ser incorrecto.

### 5.3 Cuadro Comparativo de Tiempos de Ejecución

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Tamaño de matriz** | **Lenguaje** | **Números Pares** | **Tiempo Pares (s)** | **Números Primos** | **Tiempo Primos (s)** |
| **50x50** | **C++** | **1281** | **4.8e-05** | **591** | **0.000108** |
| **50x50** | **C++** | **1284** | **5.1e-05** | **614** | **0.000117** |
| **50x50** | **C++** | **1259** | **3.8e-05** | **610** | **0.000125** |
| **50x50** | **Python** | **1257** | **0.000170** | **652** | **0.001375** |
| **50x50** | **Python** | **1272** | **0.000151** | **595** | **0.001294** |
| **50x50** | **Python** | **1237** | **0.000124** | **620** | **0.000776** |
| **100x100** | **C++** | **5004** | **0.00016** | **2533** | **0.000348** |
| **100x100** | **C++** | **4996** | **0.000211** | **2514** | **0.000467** |
| **100x100** | **C++** | **5105** | **0.00016** | **2424** | **0.000337** |
| **100x100** | **Python** | **5006** | **0.000427** | **2505** | **0.004811** |
| **100x100** | **Python** | **5078** | **0.000405** | **2465** | **0.002961** |
| **100x100** | **Python** | **5095** | **0.000763** | **2465** | **0.005590** |
| **200x200** | **C++** | **20024** | **0.00063** | **9921** | **0.001336** |
| **200x200** | **C++** | **20131** | **0.000742** | **9965** | **0.001801** |
| **200x200** | **C++** | **20054** | **0.00074** | **10054** | **0.00184** |
| **200x200** | **Python** | **19966** | **0.001599** | **10111** | **0.011854** |
| **200x200** | **Python** | **20155** | **0.001584** | **9924** | **0.013655** |
| **200x200** | **Python** | **20314** | **0.001607** | **9899** | **0.011714** |

Gráfico, Gráfico de barras

El contenido generado por IA puede ser incorrecto.

## ANÁLISIS DE LA COMPARACIÓN DE TIEMPOS

En el análisis comparativo de los tiempos de ejecución de ambos lenguajes (C++ y Python), se observa una clara diferencia en los tiempos de procesamiento, especialmente conforme aumenta el tamaño de la matriz.

1. **Tiempos para Matrices de 50x50:**
   * En matrices pequeñas como 50x50, ambos lenguajes tienen tiempos bastante cercanos, pero **C++ muestra un rendimiento ligeramente superior**. En promedio, C++ tarda entre **3.8e-05 s** y **5.1e-05 s** para contar los números pares, mientras que **Python** tiene tiempos más altos, oscilando entre **0.000124 s** y **0.000170 s**. Para el conteo de números primos, la diferencia se amplía aún más, con tiempos de **C++** en el rango de **0.000108 s** a **0.000125 s**, mientras que **Python** varía de **0.000776 s** a **0.001375 s**.
2. **Tiempos para Matrices de 100x100:**
   * A medida que se incrementa el tamaño de la matriz a 100x100, **C++ sigue mostrando una ventaja notable en cuanto a los tiempos de ejecución**. Los tiempos de ejecución en C++ para contar números primos y pares fluctúan entre **0.00016 s** y **0.000211 s** para los pares y entre **0.000337 s** y **0.000467 s** para los primos. En contraste, **Python** muestra un incremento mucho mayor en los tiempos de ejecución, especialmente para los números primos, que varían entre **0.002961 s** y **0.005590 s**.
3. **Tiempos para Matrices de 200x200:**
   * En matrices de mayor tamaño (200x200), la diferencia de rendimiento entre ambos lenguajes es aún más pronunciada. En **C++**, el conteo de números pares y primos tiene tiempos que van desde **0.00063 s** hasta **0.000742 s** para los pares y **0.001336 s** hasta **0.00184 s** para los primos. Mientras tanto, **Python** muestra un incremento considerable en los tiempos, alcanzando **0.001607 s** para los números pares y hasta **0.013655 s** para los números primos.
4. **Diferencias en el Rendimiento:**
   * Las diferencias en los tiempos de ejecución entre C++ y Python son atribuibles a varios factores inherentes a los dos lenguajes:
     + **C++** es un lenguaje compilado y de bajo nivel, lo que le permite interactuar más directamente con el hardware y optimizar los procesos de cálculo. Además, tiene menos sobrecarga en cuanto a la gestión de memoria y la ejecución de código.
     + **Python**, siendo un lenguaje interpretado y de alto nivel, introduce una capa adicional de abstracción, lo que conlleva una mayor sobrecarga en la ejecución de las operaciones. Aunque Python es muy fácil de usar y tiene una sintaxis concisa, su rendimiento no es tan eficiente como C++ para tareas que implican manipulación intensiva de datos.

**Conclusión**

Este análisis revela que **C++ tiene una ventaja clara en términos de rendimiento** en comparación con Python cuando se trabaja con operaciones computacionales intensivas, como la manipulación de matrices grandes y el conteo de números primos y pares. Esto se debe a la naturaleza de C++ como un lenguaje compilado y más cercano al hardware, lo que le permite realizar tareas más rápidamente.

Por otro lado, **Python sigue siendo una excelente opción** cuando la prioridad es la facilidad de desarrollo y la legibilidad del código, especialmente para prototipos y tareas donde el tiempo de ejecución no es tan crítico. Aunque los tiempos de ejecución son mayores en Python, su sintaxis más sencilla y su extensa comunidad de bibliotecas lo hacen ideal para muchos casos de uso.

En resumen, la elección entre C++ y Python dependerá del balance entre **velocidad de ejecución** y **facilidad de desarrollo**. Si el rendimiento es una prioridad (como en aplicaciones en tiempo real o procesamiento masivo de datos), C++ es la opción recomendada. Sin embargo, si la prioridad es la rapidez en el desarrollo y la versatilidad, Python será preferible.

**Bibliografía**

1. **Li, M., y Krishnamachari, B.** (2024). *Evaluación de la eficiencia de ChatGPT-3.5 en la resolución de problemas de codificación de diferentes niveles de complejidad: un análisis empírico* . Recuperado de [arXiv](https://arxiv.org/" \t "_blank) .
   * Este artículo analiza la eficiencia en la resolución de problemas algorítmicos utilizando ChatGPT, comparando Python y C++ en términos de rendimiento y complejidad.
2. **Proyecto Mántide** . (sf). *Algoritmos Python versus C++* . Recuperado de [https://www.mantidproject.org](https://www.mantidproject.org/) .
   * Comparación de los lenguajes Python y C++ en el desarrollo de algoritmos, enfocándose en casos de uso en procesamiento científico.
3. **Compilador GDB en línea** . (2025). *Documentación oficial* . Recuperado de [https://www.onlinegdb.com](https://www.onlinegdb.com/) .
   * Información técnica sobre el funcionamiento y capacidades del compilador en línea, ideal para pruebas de código en Python y C++.
4. **DayTrading.com** . (2023). *"Python versus C++ para algoritmos financieros"* . Recuperado de [https://www.daytrading.com](https://www.daytrading.com/) .
   * Discusión sobre el uso de Python y C++ en algoritmos financieros, resaltando diferencias en latencia y velocidad de ejecución.
5. **Van Rossum, G., y Drake, FL** (2009). *Manual de referencia de Python 3.* Fundación del Software Python.
   * Manual oficial de referencia de Python 3, que detalla características del lenguaje relevantes para la implementación de algoritmos.
6. **Stroustrup, B.** (2013). *El lenguaje de programación C++ (4.ª edición)* . Addison-Wesley Professional.
   * Texto fundamental que explora el diseño y las capacidades de C++ en la programación de bajo nivel y optimización algorítmica.
7. **Miller, DA, y Box, H.** (2023). *Análisis comparativo del desarrollo de algoritmos en Python y C++* . Revista de Ciencias de la Computación, 45(2), 145–160.
   * Estudio académico que compara Python y C++ en términos de implementación y rendimiento de algoritmos.