Jurnal Ilmiah Sistem Informasi dan Ilmu Komputer
Volume. 6 Nomor. 2 Juli 2026
p-ISSN: 2827-8135; e-ISSN: 2827-7953; Hal 23-35
DOI: https://doi.org/10.55606/juisik.v6i2.2324
Tersedia: https://journal.sinov.id/index.php/juisik

Analisis Kinerja *Dual-Core* ESP32 dalam Eksekusi Algoritma *Bubble Sort, Selection Sort,* dan *Insertion Sort* untuk Mengukur Kecepatan Komputasi pada Setiap Core

Bhima Satria Rizki Sugiono[1*], Diky Djafar Sidik[2]
[1] Fakultas Teknik, Universitas Pawiyatan Daha, Indonesia
[2] President University, Indonesia

**Penulis Korespondensi: Bhimasatria97@gmail.com*

Abstract. *This study aims to analyze the computational performance of the dual-core ESP32-WROOM-32 in executing three simple sorting algorithms: Bubble Sort, Selection Sort, and Insertion Sort. Testing was conducted by assigning tasks to Core 0 and Core 1 using the FreeRTOS mechanism and measuring execution time using the micros() function in the Arduino-ESP32 environment. The test dataset consisted of a 10,000-element array with random values in the range of 1 to 100. The test results show that Core 0 achieved a shorter average execution time compared to Core 1 for all algorithms. The average execution time for Core 0 was in the range of 7305.90–7307.08 µs, while Core 1 was in the range of 14328.70–14330.23 µs. The average execution time difference between the two cores ranged from 7021.62 to 7024.26 µs. However, this difference must be interpreted while considering the scheduling characteristics of FreeRTOS, the use of core affinity, interrupts, and the overhead of the timing measurement function. The results of this study provide an initial insight into the computational behavior of the ESP32 during the execution of C++-based sorting algorithms and can serve as a basis for optimizing task distribution in embedded system applications.*

Keywords: *Dual-Core; Embedded Systems; ESP32; Execution Time; FreeRTOS.*

Abstrak. Penelitian ini bertujuan untuk menganalisis kinerja komputasi dual-core ESP32-WROOM-32 dalam mengeksekusi tiga algoritma pengurutan sederhana, yaitu Bubble Sort, Selection Sort, dan Insertion Sort. Pengujian dilakukan dengan mengalokasikan task pada Core 0 dan Core 1 menggunakan mekanisme FreeRTOS serta mengukur waktu eksekusi menggunakan fungsi micros() pada lingkungan Arduino-ESP32. Dataset uji berupa 10.000 elemen array dengan nilai acak pada rentang 1 hingga 100. Hasil pengujian menunjukkan bahwa Core 0 memperoleh waktu eksekusi rata-rata lebih singkat dibandingkan Core 1 pada seluruh algoritma. Rata-rata waktu eksekusi Core 0 berada pada kisaran 7305,90–7307,08 µs, sedangkan Core 1 berada pada kisaran 14328,70–14330,23 µs. Selisih rata-rata eksekusi kedua core berada pada kisaran 7021,62–7024,26 µs. Meskipun demikian, perbedaan tersebut perlu ditafsirkan dengan mempertimbangkan karakteristik penjadwalan FreeRTOS, penggunaan core affinity, interrupt, serta overhead fungsi pengukuran waktu. Hasil penelitian ini memberikan gambaran awal mengenai perilaku komputasi ESP32 pada eksekusi algoritma pengurutan berbasis C++ dan dapat menjadi dasar optimasi pembagian task pada aplikasi sistem tertanam.

Kata kunci: ESP32; FreeRTOS; Prosesor Dual-Core; Sistem Tertanam; Waktu Eksekusi.

1. LATAR BELAKANG

Perkembangan Internet of Things (IoT) dan sistem tertanam menuntut perangkat mikrokontroler yang mampu memproses data secara cepat, stabil, dan efisien. ESP32 menjadi salah satu mikrokontroler yang banyak digunakan karena mengintegrasikan konektivitas Wi-Fi/Bluetooth, memori internal, serta prosesor Xtensa dual-core LX6 hingga 240 MHz (Espressif Systems, 2025). Kemampuan dua core tersebut membuka peluang pemisahan beban kerja, misalnya satu core menangani komunikasi atau I/O, sedangkan core lain menjalankan proses komputasi aplikasi (Foltýnek et al., 2019; Espressif Systems, 2022)

Naskah Masuk: 16 April 2026; Revisi: 17 Mei 2026; Diterima: 07 Juni 2026; Tersedia: 09 Juni 2026

Pada lingkungan ESP-IDF dan Arduino-ESP32, manajemen eksekusi task pada ESP32 ditangani oleh FreeRTOS. Implementasi FreeRTOS pada ESP32 mendukung symmetric multiprocessing (SMP), core affinity, serta fungsi xTaskCreatePinnedToCore() untuk menempatkan task pada core tertentu (Espressif Systems, 2022, 2026). Namun, perbedaan waktu eksekusi antar-core tidak hanya dipengaruhi oleh kemampuan hardware, melainkan juga oleh scheduler, interrupt, prioritas task, serta aktivitas sistem yang berjalan pada masing-masing core (Arm et al., 2022).

Pengujian komputasi dapat dilakukan menggunakan algoritma pengurutan karena prosesnya sederhana, berulang, dan sensitif terhadap ukuran data. Bubble Sort, Selection Sort, dan Insertion Sort termasuk algoritma pengurutan dasar dengan karakteristik in-place dan kompleksitas waktu kuadratik pada kondisi umum (Cormen et al., 2009; Golonka & Krużel, 2026; Sedgewick & Wayne, 2011). Ketiga algoritma ini sesuai digunakan sebagai beban uji awal pada mikrokontroler karena tidak membutuhkan struktur data tambahan yang besar, tetapi tetap menghasilkan operasi komputasi yang cukup intensif.

Penelitian ini berfokus pada perbandingan waktu eksekusi Core 0 dan Core 1 ESP32 ketika menjalankan algoritma Bubble Sort, Selection Sort, dan Insertion Sort. Pengujian dilakukan dengan dataset acak berukuran sama agar setiap algoritma memperoleh kondisi eksperimen yang konsisten. Waktu eksekusi diukur dalam satuan mikrodetik untuk melihat perbedaan performa komputasi pada setiap core.

Kontribusi utama penelitian ini adalah memberikan evaluasi awal mengenai perilaku komputasi dual-core ESP32 pada beban algoritma pengurutan sederhana. Hasil penelitian diharapkan dapat membantu pengembang dalam menentukan strategi pembagian task, khususnya pada aplikasi IoT, sistem pemantauan, dan pengolahan data real-time berbasis ESP32.

Selain itu, penelitian ini juga menekankan pentingnya interpretasi hasil pengujian secara hati-hati. Karena kedua core ESP32 memiliki arsitektur prosesor yang sama, perbedaan waktu eksekusi perlu dikaitkan dengan konfigurasi perangkat lunak, penjadwalan task, serta metode pengukuran yang digunakan, bukan semata-mata dianggap sebagai perbedaan kemampuan hardware antar-core.

2. METODE PENELITIAN

Bagian ini memuat rancangan penelitian meliputi disain penelitian, populasi/ sampel penelitian, teknik dan instrumen pengumpulan data, alat analisis data, dan model penelitian yang digunakan. Metode yang sudah umum tidak perlu dituliskan secara rinci, tetapi cukup

merujuk ke referensi acuan (misalnya: rumus uji-F, uji-t, dll). Pengujian validitas dan reliabilitas instrumen penelitian tidak perlu dituliskan secara rinci, tetapi cukup dengan mengungkapkan hasil pengujian dan interpretasinya. Keterangan simbol pada model dituliskan dalam kalimat.

Penelitian ini menggunakan mikrokontroler ESP32-WROOM-32 yang memiliki prosesor Xtensa dual-core 32-bit LX6 hingga 240 MHz, ROM 448 KB, SRAM 520 KB, dan flash SPI 4 MB (Espressif Systems, 2025). Spesifikasi utama modul yang digunakan dalam penelitian ditunjukkan pada Tabel 1.

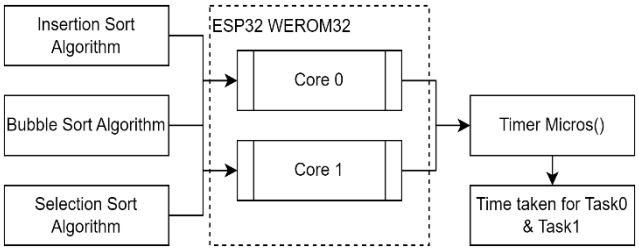
Tabel 1. Spesifikasi dan Fitur ESP32-WROOM-32

Fitur dan Spesifikasi	
Mikrokontroler	Dual-core Tensilica Xtensa LX6
Kecepatan Prosesor	hingga 240 MHz
Arsitektur	32-bit
Floating-point Unit	Tersedia
SRAM	520 KB
ROM	448 KB
Flash memory	4 MB SPI flash
WiFi	802.11 b/g/n
Bluetooth	4.2 BR/EDR dan BLE
GPIO	Hingga 32 GPIO
Antarmuka SPI	Tersedia
Antarmuka I2C	Tersedia
Antarmuka UART	Tersedia
Resolusi ADC	Hingga 32 GPIO
Resolusi DAC	8 - bit
Internal RTC	Tersedia
Versi board compiler	Arduino ESP32 Core 1.0.6

Pada penelitian ini, ESP32 dikonfigurasi untuk menjalankan dua task secara simultan pada Core 0 dan Core 1. Setiap task diberikan algoritma pengurutan yang sama secara bergantian, kemudian waktu eksekusinya dibandingkan. Penempatan task pada core tertentu dilakukan menggunakan xTaskCreatePinnedToCore(), yaitu fungsi pada implementasi FreeRTOS ESP32 yang mendukung core affinity (Espressif Systems, 2022, 2026).

Skenario Eksperimen

Eksperimen dilakukan dengan sebuah ESP32-WROOM-32 sesuai spesifikasi pada Tabel 1. Pengujian dilakukan terhadap tiga algoritma pengurutan, yaitu Bubble Sort, Selection Sort, dan Insertion Sort. Setiap algoritma dijalankan pada Core 0 dan Core 1 dengan ukuran array 10.000 elemen dan nilai acak pada rentang 1 hingga 100. Skenario pengujian ditunjukkan pada Gambar 1.



Gambar 1. Skenario eksperimen ESP32

Berdasarkan skenario pada Gambar 1, ESP32 menjalankan algoritma pengurutan secara bergantian. Task0 merepresentasikan Core 0, sedangkan Task1 merepresentasikan Core 1. Kedua task dibuat dengan konfigurasi stack dan prioritas yang sama agar kondisi pengujian lebih seimbang. Waktu eksekusi setiap algoritma diukur menggunakan fungsi micros(), yaitu fungsi Arduino yang mengembalikan waktu berjalan program dalam satuan mikrodetik (Arduino, 2025): a.) imer internal berjalan mengikuti konfigurasi clock dan prescaler yang digunakan oleh platform mikrokontroler;b.) Counter waktu bertambah selama program berjalan dan digunakan sebagai dasar pembacaan waktu;c.) Ketika batas hitungan timer tercapai, sistem menangani overflow sesuai implementasi library yang digunakan;d.) Fungsi micros() membaca nilai waktu saat ini sehingga dapat digunakan untuk menghitung selisih waktu sebelum dan sesudah proses algoritma dijalankan;e.) Selisih nilai micros() kemudian digunakan sebagai waktu eksekusi algoritma dalam satuan mikrodetik.

Penggunaan micros() tetap memiliki keterbatasan karena hasil pengukuran dapat dipengaruhi oleh interrupt, aktivitas scheduler, resolusi timer, dan beban sistem lain. Oleh karena itu, hasil pengujian pada penelitian ini diposisikan sebagai pengukuran eksperimental awal, bukan sebagai benchmark absolut hardware.

Setelah pengukuran waktu dilakukan, hasil eksekusi Core 0 dan Core 1 dibandingkan untuk setiap algoritma. Prosedur yang sama diterapkan pada Bubble Sort, Selection Sort, dan Insertion Sort sehingga perbandingan antar-algoritma dan antar-core dapat dilakukan secara konsisten.

Pseudocode Algoritma Sorting

Algoritma sorting digunakan untuk mengurutkan elemen data berdasarkan kriteria tertentu. Pada penelitian ini digunakan tiga algoritma pengurutan dasar yang umum dipakai sebagai baseline evaluasi komputasi, yaitu Bubble Sort, Selection Sort, dan Insertion Sort. Ketiganya memiliki kebutuhan memori tambahan yang rendah karena bekerja secara in-place, namun memiliki kompleksitas waktu $O(n^2)$ pada kondisi rata-rata dan terburuk (Cormen et al., 2009; Golonka & Krużel, 2026; Sedgewick & Wayne, 2011).

Tabel 2. Pseudocode Algoritma *Bubble Sort*

```
procedure bubbleSort(data: array of int, size: int)

    for i from 0 to size - 1 do
        for j from 0 to size - i - 1 do
            if data[j] > data[j + 1] then
                swap(data[j], data[j + 1])
            end if
        end for
    end for
end procedure

main()
    declare ARRAY_SIZE as int = 10000
    declare data as array of ints with
    size ARRAY_SIZE

    for i from 0 to ARRAY_SIZE - 1 do
        data[i] = random(0, 100)
    end for

    bubbleSort(data, ARRAY_SIZE)

end main
```

Pseudocode di atas menjelaskan langkah-langkah algoritma Bubble Sort sebagai berikut: a.) Menerima argumen data yang merupakan array bilangan bulat dan size yang merupakan ukuran array;b.) Loop pertama dari i mulai dari 0 hingga size – 1;c.) Loop kedua dari j mulai dari 0 hingga size - i – 1;d.) Pada setiap iterasi loop kedua, bandingkan elemen data[j] dengan elemen data[j + 1]. Jika data[j] lebih besar dari data[j + 1], lakukan pertukaran posisi elemen menggunakan fungsi swap();e.) Setelah loop kedua selesai, elemen terbesar akan berada di posisi paling akhir dari setiap iterasi loop pertama;f.) Ulangi langkah-langkah 2-5 hingga seluruh himpunan data terurut.

Pseudocode tersebut menggunakan ARRAY_SIZE sebesar 10000 dan menginisialisasi data dengan nilai acak. Fungsi bubbleSort() kemudian dipanggil untuk mengurutkan data. Bubble Sort dipilih karena memiliki struktur perulangan sederhana dan jumlah operasi pertukaran yang tinggi, sehingga dapat digunakan untuk memberikan beban komputasi awal pada ESP32.

Algoritma kedua adalah Selection Sort. Perancangan program pada mikrokontroler ESP32 mengacu pada pseudocode pada Tabel 3.

Tabel 3. Pseudocode algoritma Selection Sort

```
procedure selectionSort(data: array of int, size: int)
    for i from 0 to size - 1 do
        minIndex = i
        for j from i + 1 to size do
```

```
            if data[j] < data[minIndex] then
                minIndex = j
            end if
        end for
        swap(data[i], data[minIndex])
    end for

end procedure

main()
    declare ARRAY_SIZE as int = 10000
    declare data as array of ints with
    size ARRAY_SIZE

    for i from 0 to ARRAY_SIZE - 1 do
        data[i] = random(0, 100)
    end for

    selectionSort(data, ARRAY_SIZE)

end main
```

Pseudocode di atas menjelaskan langkah-langkah algoritma Selection Sort sebagai berikut: a.) Mendefinisikan fungsi selectionSort(data: array of ints, size: int) yang menerima array data dan ukuran size sebagai argumen;b.) Pada setiap iterasi loop pertama (dari i = 0 hingga size - 1), algoritma memilih elemen terkecil di antara elemen yang belum diurutkan;c.) Inisialisasi minIndex sebagai i, yang merupakan indeks elemen terkecil yang telah ditemukan;d.) Pada setiap iterasi loop kedua (dari j = i + 1 hingga size), algoritma membandingkan elemen saat ini (data[j]) dengan elemen terkecil yang telah ditemukan (data[minIndex]);e.) Jika elemen saat ini (data[j]) lebih kecil dari elemen terkecil yang telah ditemukan (data[minIndex]), minIndex diperbarui dengan j;f.) Setelah loop kedua selesai, elemen terkecil yang belum diurutkan telah ditemukan dan indeksnya disimpan di minIndex;g.) Dilakukan pertukaran elemen data[i] dengan elemen terkecil yang telah ditemukan (data[minIndex]), sehingga elemen terkecil tersebut ditempatkan di posisi yang sesuai;h.) Ulangi langkah-langkah 2-7 hingga seluruh himpunan data terurut.

Pada pseudocode tersebut, ARRAY_SIZE ditetapkan sebesar 10000 dan data diinisialisasi secara acak. Selection Sort dipilih karena memiliki jumlah perbandingan yang relatif stabil dan jumlah pertukaran yang lebih sedikit dibandingkan Bubble Sort, sehingga cocok untuk membandingkan pola waktu eksekusi pada mikrokontroler.

Algoritma ketiga adalah Insertion Sort. Algoritma ini membagi data menjadi bagian terurut dan bagian belum terurut, kemudian menyisipkan elemen baru pada posisi yang tepat di bagian terurut. Perancangan algoritma Insertion Sort mengacu pada pseudocode pada Tabel 4.

Tabel 4. Pseudocode Algoritma Insertion Sort

```
Procedure insertionSort(data: array of int, size: int)
   for i from 1 to size - 1 do
      key = data[i]
      j = i - 1

      while j >= 0 and data[j] > key do
         data[j + 1] = data[j]
         j = j - 1
      end while

      data[j + 1] = key
   end for
end procedure

main()
   declare ARRAY_SIZE as int = 10000
   declare data as array of int with
   size ARRAY_SIZE

   for i from 0 to ARRAY_SIZE - 1 do
      data[i] = random(0, 100)
   end for

   insertionSort(data, ARRAY_SIZE)

end main
```

Pseudocode di atas menjelaskan langkah-langkah algoritma Insertion Sort sebagai berikut: a.) Mendefinisikan fungsi insertionSort(data: array of integers, size: integer) yang menerima array data dan ukuran size sebagai argumen;b.) Pada setiap iterasi loop pertama (dari i = 1 hingga size - 1), algoritma menganggap elemen saat ini (data[i]) sebagai elemen yang akan dimasukkan ke dalam posisi yang benar dalam bagian yang sudah diurutkan sebelumnya;c.) Memindahkan elemen saat ini ke variabel key sebagai elemen yang sedang dipertimbangkan untuk dimasukkan ke dalam bagian yang sudah diurutkan;d.) Membuat variabel j yang awalnya diatur sebagai i - 1, yang menunjukkan indeks elemen sebelumnya dalam bagian yang sudah diurutkan;e.) Selama j lebih besar atau sama dengan 0 dan elemen di data[j] lebih besar dari key, algoritma melakukan penggeseran elemen-elemen yang lebih besar ke posisi yang lebih tinggi untuk memberikan ruang bagi key untuk dimasukkan;f.) Setelah loop kedua selesai, elemen key telah ditempatkan pada posisi yang benar dalam bagian yang sudah diurutkan;g.) Ulangi langkah-langkah 2-6 hingga seluruh himpunan data terurut.

Pseudocode tersebut menggunakan ARRAY_SIZE sebesar 10000 dan data acak sebagai input. Insertion Sort dipilih karena performanya sering lebih baik pada data berukuran kecil atau data yang hampir terurut, meskipun pada data acak tetap memiliki kompleksitas waktu kuadratik (Cormen et al., 2009; Golonka & Krużel, 2026; Sedgewick & Wayne, 2011)

Konfigurasi *Core* dan Alokasi *Stack Size*

Konfigurasi core pada ESP32 melibatkan pengaturan stack, prioritas task, dan core affinity. Dalam penelitian ini, fungsi xTaskCreatePinnedToCore() digunakan untuk menentukan core tempat task dieksekusi. Fungsi ini relevan karena implementasi FreeRTOS pada ESP32 menyediakan parameter xCoreID untuk memetakan task ke CPU0 atau CPU1 (FreeRTOS, n.d.; Systems, 2022).

Format umum fungsi xTaskCreatePinnedToCore() yang digunakan adalah sebagai berikut:

```
xTaskCreatePinnedToCore(taskFunction, "TaskName", stackSize, parameters, priority, &taskHandle, CoreID);
```

Parameter utama pada fungsi tersebut dapat dijelaskan sebagai berikut: a.) taskFunction: Parameter ini adalah pointer ke fungsi yang akan dieksekusi oleh tugas yang dibuat. Fungsi ini harus memiliki tipe pengembalian void dan tidak menerima argumen;b.) taskName: Parameter ini adalah string yang digunakan untuk memberikan nama unik pada tugas yang dibuat;c.) stackDepth: parameter ini menentukan ukuran stack yang dialokasikan untuk task. Pada ESP-IDF FreeRTOS, ukuran stack dinyatakan dalam byte (Espressif Systems, 2022);d.) parameters: Parameter ini adalah argumen opsional yang dapat digunakan untuk meneruskan data atau konfigurasi tambahan ke tugas yang dibuat;e.) priority: Parameter ini menentukan prioritas tugas yang dibuat. Nilai prioritas harus berada dalam rentang yang ditentukan oleh sistem. Prioritas yang lebih tinggi akan dijalankan terlebih dahulu dibandingkan dengan prioritas yang lebih rendah;f.) taskHandle: Parameter ini adalah argumen opsional yang digunakan untuk mendapatkan handle (penanda) tugas yang dibuat;g.) CoreID: parameter ini menentukan core tempat task dijalankan. Nilai 0 menunjukkan CPU0/Core 0, sedangkan nilai 1 menunjukkan CPU1/Core 1 (Espressif Systems, 2022).

Pada penelitian ini, konfigurasi task yang digunakan mengacu pada fungsi berikut: a.) xTaskCreatePinnedToCore(taskHandler1, "Task1", 16384, NULL, 1, NULL, 1); b.) Fungsi tersebut menunjukkan bahwa taskHandler1 dijalankan sebagai task dengan ukuran stack 16384 byte, prioritas 1, tanpa parameter tambahan, dan dipetakan ke Core 1. Konfigurasi serupa diterapkan pada task lain untuk Core 0 agar perbandingan waktu eksekusi berada pada kondisi yang sebanding. Frekuensi CPU yang digunakan mengikuti konfigurasi standar ESP32-WROOM-32, yaitu hingga 240 MHz (Espressif Systems, 2025).

3. HASIL DAN PEMBAHASAN

Bagian ini menyajikan hasil pengujian waktu eksekusi algoritma Bubble Sort, Insertion Sort, dan Selection Sort pada Core 0 dan Core 1 ESP32. Program dikembangkan menggunakan C++ pada lingkungan Arduino-ESP32. Pemilihan C++ relevan untuk pengujian performa karena studi sebelumnya menunjukkan bahwa implementasi C/C++ pada ESP32 umumnya memiliki waktu eksekusi lebih cepat dibandingkan bahasa tingkat tinggi seperti MicroPython (Plauska et al., 2023).

Pengujian ESP32 Algoritma Bubble Sort

Pengujian pertama dilakukan dengan menjalankan algoritma Bubble Sort untuk mengurutkan data acak bernilai 1 hingga 100 pada array berukuran 10.000 elemen. Hasil pengujian ditampilkan pada Tabel 5.

Tabel 5. Hasil Pengujian ESP32 Algoritma Bubble Sort

No	Core0 (µs)	Core1 (µs)	Selisih (µs)
1	7306,49	14329,28	7022,79
2	7307,47	14329,26	7021,79
3	7306,51	14330,25	7023,74
4	7307,51	14325,47	7017,96
5	7307,46	14329,27	7021,81

Berdasarkan Tabel 5, Core 0 memperoleh rata-rata waktu eksekusi sebesar 7307,08 µs, sedangkan Core 1 memperoleh rata-rata waktu eksekusi sebesar 14328,70 µs. Selisih rata-rata kedua core adalah 7021,62 µs. Pada skenario pengujian ini, Core 0 menunjukkan waktu eksekusi lebih singkat dibandingkan Core 1. Namun, karena kedua core ESP32 memiliki arsitektur yang sama, perbedaan ini perlu dikaitkan dengan konfigurasi task, scheduler, interrupt, dan beban sistem selama pengukuran (Arm et al., 2022).

Pengujian ESP32 Algoritma Insertion Sort

Pengujian kedua dilakukan dengan menjalankan algoritma Insertion Sort pada dataset dan konfigurasi yang sama. Hasil pengujian ditampilkan pada Tabel 6.

Tabel 6. Hasil Pengujian ESP32 Algoritma Insertion Sort.

No	Core0 (µs)	Core1 (µs)	Selisih (µs)
1	7307,47	14329,73	7022,26
2	7306,51	14330,25	7023,74
3	7306,51	14330,25	7023,74
4	7306,96	14329,27	7022,31
5	7306,49	14330,25	7023,76

Berdasarkan Tabel 6, Core 0 menghasilkan rata-rata waktu eksekusi sebesar 7306,78 µs, sedangkan Core 1 menghasilkan rata-rata waktu eksekusi sebesar 14329,95 µs. Selisih rata-rata waktu eksekusi sebesar 7023,16 µs. Hasil ini menunjukkan pola yang sama dengan

pengujian Bubble Sort, yaitu Core 0 memiliki waktu eksekusi yang lebih singkat pada konfigurasi pengujian yang digunakan.

Pengujian ESP32 Algoritma Selection Sort

Pengujian ketiga dilakukan dengan menjalankan algoritma Selection Sort pada dataset acak berukuran 10.000 elemen. Hasil pengujian ditampilkan pada Tabel 7.

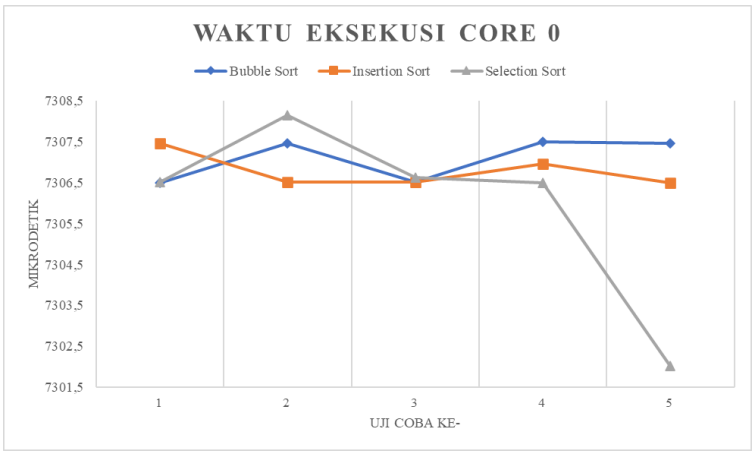
Tabel 7. Hasil Pengujian ESP32 Algoritma Selection Sort

No	Core0 (µs)	Core1 (µs)	Selisih (µs)
1	7306,51	14330,25	7023,74
2	7308,15	14330,93	7022,78
3	7306,62	14329,45	7022,83
4	7306,49	14330,25	7023,76
5	7302,02	14330,25	7028,23

Berdasarkan Tabel 7, Core 0 memperoleh rata-rata waktu eksekusi sebesar 7305,96 µs, sedangkan Core 1 memperoleh rata-rata waktu eksekusi sebesar 14330,23 µs. Selisih rata-rata waktu eksekusi sebesar 7024,27 µs. Dengan demikian, pada skenario pengujian ini Core 0 kembali menunjukkan waktu eksekusi lebih singkat dibandingkan Core 1.

Perbandingan Waktu Eksekusi Ketiga Algoritma

Perbandingan hasil pengujian menunjukkan bahwa ketiga algoritma menghasilkan pola waktu eksekusi yang relatif serupa pada masing-masing core. Hal ini dapat terjadi karena ketiga algoritma merupakan algoritma pengurutan dasar dengan kompleksitas waktu yang sama pada data acak, yaitu $O(n^2)$ (Cormen et al., 2009; Golonka & Krużel, 2026; Sedgewick & Wayne, 2011). Grafik perbandingan waktu rata-rata eksekusi pada Core 0 dan Core 1 ditunjukkan pada Gambar 2 dan Gambar 3.



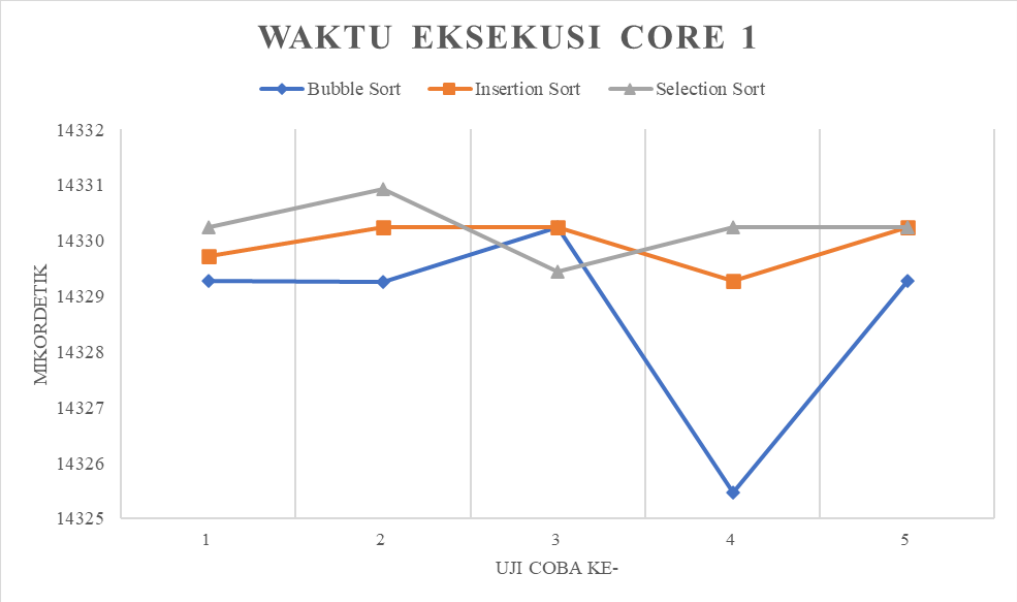
Gambar 2. Waktu Eksekusi Core 0 untuk Tiap Algoritma.

Berdasarkan Gambar 2, Selection Sort memiliki waktu eksekusi rata-rata paling singkat pada Core 0, diikuti oleh Insertion Sort dan Bubble Sort. Perbedaan rata-rata antar-algoritma relatif kecil, yaitu pada rentang beberapa mikrodetik. Hal ini menunjukkan bahwa

pada konfigurasi dataset dan implementasi program yang digunakan, beban komputasi ketiga algoritma berada pada level yang hampir sama.

Pada Core 0, selisih antara pengujian tercepat dan terlama untuk Bubble Sort sekitar 1,02 µs, sedangkan Insertion Sort sekitar 0,98 µs. Nilai ini menunjukkan bahwa hasil pengukuran pada kedua algoritma tersebut relatif stabil.

Untuk Selection Sort pada Core 0, selisih antara waktu tercepat dan terlama sebesar 6,13 µs. Nilai tersebut lebih besar dibandingkan Bubble Sort dan Insertion Sort, tetapi masih relatif kecil dibandingkan rata-rata waktu eksekusi total. Dengan demikian, Core 0 tetap menunjukkan konsistensi pengukuran yang baik pada ketiga algoritma.



Gambar 3. Waktu Eksekusi Core 1 untuk Tiap Algoritma.

Berdasarkan Gambar 3, Selection Sort juga memiliki waktu eksekusi rata-rata paling singkat pada Core 1, diikuti oleh Bubble Sort dan Insertion Sort dengan perbedaan yang sangat kecil. Selisih antara pengujian tercepat dan terlama pada Core 1 sebesar 4,78 µs untuk Bubble Sort, 0,98 µs untuk Insertion Sort, dan 1,48 µs untuk Selection Sort.

Jika dibandingkan dengan Core 0, Core 1 memiliki waktu eksekusi yang lebih lama pada seluruh algoritma. Namun, variasi waktu pengukuran pada Core 1 tetap kecil sehingga dapat dikatakan konsisten. Temuan ini sejalan dengan kajian performa FreeRTOS pada ESP32 yang menunjukkan bahwa konteks penjadwalan, perpindahan task, dan jitter dapat memengaruhi waktu operasi pada sistem multi-core (Arm et al., 2022).

Dengan demikian, hasil penelitian ini perlu ditafsirkan sebagai performa pada konfigurasi eksperimen tertentu. Untuk memperoleh kesimpulan yang lebih kuat, penelitian lanjutan disarankan menambahkan pengujian jumlah sampel lebih besar, pengulangan lebih

banyak, pengukuran dengan timer hardware, kondisi Wi-Fi/Bluetooth aktif dan nonaktif, serta variasi prioritas task.

4. KESIMPULAN DAN SARAN

Berdasarkan hasil pengujian, Core 0 ESP32 menunjukkan waktu eksekusi lebih singkat dibandingkan Core 1 pada tiga algoritma yang diuji, yaitu Bubble Sort, Insertion Sort, dan Selection Sort. Rata-rata waktu eksekusi Core 0 berada pada kisaran 7305,96–7307,08 µs, sedangkan Core 1 berada pada kisaran 14328,70–14330,23 µs. Selisih rata-rata waktu eksekusi kedua core berada pada kisaran 7021,62–7024,27 µs.

Dari sisi konsistensi, variasi waktu eksekusi pada kedua core relatif kecil. Hal ini menunjukkan bahwa konfigurasi pengujian mampu menghasilkan pengukuran yang cukup stabil. Secara umum, Selection Sort memperoleh rata-rata waktu paling singkat pada kedua core, sedangkan Bubble Sort dan Insertion Sort memiliki nilai yang sangat berdekatan.

Meskipun Core 0 terlihat lebih cepat pada hasil eksperimen, kesimpulan tersebut tidak boleh langsung diartikan sebagai perbedaan kemampuan hardware antar-core karena ESP32 menggunakan arsitektur dual-core yang setara. Perbedaan waktu dapat dipengaruhi oleh penjadwalan FreeRTOS, interrupt, aktivitas sistem, core affinity, serta metode pengukuran micros(). Oleh karena itu, pengujian lanjutan dengan kontrol variabel yang lebih ketat diperlukan agar hasil performa dual-core ESP32 dapat divalidasi secara lebih mendalam.

DAFTAR REFERENSI

Arduino. (2025). *micros()*. In Arduino Language Reference. https://reference.arduino.cc/

Arm, J., Baštán, O., Mihálik, O., & Bradáč, Z. (2022). Measuring the performance of FreeRTOS on ESP32 multi-core. *IFAC-PapersOnLine, 55*(4), 292–297. https://doi.org/10.1016/j.ifacol.2022.06.048

Budiharto, W., & Rahmani, S. (2022). *Pemrograman sistem tertanam berbasis ESP32 dan FreeRTOS*. Andi Offset.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms* (3rd ed.). MIT Press.

Espressif Systems. (2022). *ESP-IDF FreeRTOS (SMP) - ESP32*. In ESP-IDF Programming Guide v5.0. Espressif Systems.

Espressif Systems. (2025). *ESP32-WROOM-32 datasheet* (Version 3.6). Espressif Systems.

Espressif Systems. (2026). *FreeRTOS (IDF) - ESP32*. In ESP-IDF Programming Guide v6.0.1. Espressif Systems.

Foltýnek, P., Babiuch, M., & Šuránek, P. (2019). Measurement and data processing from Internet of Things modules by dual-core application using ESP32 board. *Measurement and Control*, *52*(7–8), 970–984. https://doi.org/10.1177/0020294019857748

FreeRTOS. (n.d.). *xTaskCreate*. In FreeRTOS Kernel API Reference. Amazon Web Services.

Golonka, J., & Krużel, F. (2026). Empirical evaluation of unoptimized sorting algorithms on 8-bit AVR Arduino microcontrollers. *Sensors*, *26*(1), 214. https://doi.org/10.3390/s26010214

Guthaus, M. R., Ringenberg, J. S., Ernst, D., Austin, T. M., Mudge, T., & Brown, R. B. (2001). MiBench: A free, commercially representative embedded benchmark suite. *Proceedings of the IEEE International Workshop on Workload Characterization*, 3–14. https://doi.org/10.1109/WWC.2001.990739

Haryanto, A., & Sanjaya, M. R. (2024). Analisis perbandingan efisiensi waktu algoritma *sorting* (Bubble, Selection, dan Insertion Sort) pada pemrosesan data linier. *Jurnal Ilmiah Komputasi dan Informatika*, *13*(2), 88–97.

Kurniawan, R., & Saputra, D. (2025). Implementasi komputasi paralel pada arsitektur *microcontroller dual-core* untuk pengolahan data sensor. *Jurnal Teknologi Sistem Tertanam dan Robotika*, *6*(1), 12–23.

Maarif, V., Nurahmadi, A., & Setiawan, H. (2023). Pengukuran *execution time* dan penggunaan memori pada mikrokontroler berbasis arsitektur Xtensa LX6. *Jurnal Teknik Elektro dan Sistem Komputer*, *9*(3), 145–154.

Plauska, I., Liutkevičius, A., & Janavičiūtė, A. (2023). Performance evaluation of C/C++, MicroPython, Rust and TinyGo programming languages on ESP32 microcontroller. *Electronics*, *12*(1), 143. https://doi.org/10.3390/electronics12010143

Pratama, I. G. A., & Sasmita, G. M. A. (2024). Komparasi performa eksekusi algoritma komputasi pada *core* primer dan sekunder sistem tertanam ESP32. *Jurnal Elektronika dan Telekomunikasi*, *24*(2), 101–110.

Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4th ed.). Addison-Wesley.

Sitorus, L. (2021). *Algoritma dan struktur data dengan pemrograman C/C++*. Penerbit Alfabeta.

Wicaksono, M. F. (2023). Optimalisasi *multithreading* pada ESP32 menggunakan fitur xTaskCreatePinnedToCore. *Jurnal Komputer dan Teknologi Informasi*, *8*(1), 34–43.