Analiza i porównanie algorytmów sortowania

Anna Leszczyńska

Listopad 2025

1 Wstęp

Omówiona tu będzie implementacja i analiza działania czterech algorytmów sortowania:

- QuickSort z jednym pivotem,
- QuickSort dwupivotowy,
- RadixSort ,
- Sortowanie kubełkowe.

Mierzono:

- liczbę porównań kluczy,
- liczbę przypisań,
- czas działania w milisekundach.

.

Dane zostały zapisane w pliku wyniki2.csv, który został wykorzystany do generowania tabel i wykresów.

2 Najważniejsze fragmenty kodu i opis zaimplementowanych algorytmów

2.1 QuickSort z jednym pivotem

QuickSort jest klasycznym algorytmem sortowania dziel–i–zwyciężaj. Jako pivot wybierany jest ostatni element tablicy.

```
int podziel_jeden_pivot(vector<int>& a, int lewy, int prawy) {
    int pivot = a[prawy];
    g_liczniki.przypisania++;

    int i = lewy;
    for (int j = lewy; j < prawy; j++) {
        if (porownaj_mniejsze_rowne(a[j], pivot)) {
            zamien(a[i], a[j]);
            i++;
        }
    }
    zamien(a[i], a[prawy]);
    return i;
}
```

2.2 QuickSort z dwoma pivotami (dual pivot)

```
void szybkie_sortowanie_dwa_pivoty(vector<int>& a, int lewy, int prawy) {
    if (lewy >= prawy) return;

    if (porownaj_wieksze(a[lewy], a[prawy]))
        zamien(a[lewy], a[prawy]);

    int p = a[lewy], q = a[prawy];
    g_liczniki.przypisania += 2;

    int lt = lewy + 1, gt = prawy - 1, i = lt;

    while (i <= gt) {
        if (porownaj_mniejsze(a[i], p)) {
            zamien(a[i], a[lt]); lt++; i++;
        } else if (porownaj_wieksze(a[i], q)) {
            zamien(a[i], a[gt]); gt--;
        } else i++;
    }
    lt--; gt++;
    zamien(a[lewy], a[lt]);
    zamien(a[prawy], a[gt]);

    szybkie_sortowanie_dwa_pivoty(a, lewy, lt - 1);
    szybkie_sortowanie_dwa_pivoty(a, lt + 1, gt - 1);
    szybkie_sortowanie_dwa_pivoty(a, gt + 1, prawy);
}
```

2.3 RadixSort

Najważniejszym etapem algorytmu RadixSort jest sortowanie po pojedynczej cyfrze przy zadanej podstawie. Funkcja zlicz_cyfre wykonuje sortowanie pozycyjne.

```
void zlicz_cyfre(vector<int> &v, int exp, int base) {
    int n = (int)v.size();
    vector<int> out(n);
    vector<int> cnt(base, 0);

    //Zliczanie wyst pien ka dej cyfry
    for (int i = 0; i < n; i++) {
        int c = (v[i] / exp) % base;
        if (c < 0) c += base;   //do liczb ujemnych
        cnt[c]++;
    }

    //Tworzenie prefiksow
    for (int i = 1; i < base; i++)
        cnt[i] += cnt[i - 1];

    // Przepisanie element w w odpowiedniej kolejno ci
    for (int i = n - 1; i >= 0; i--) {
        int c = (v[i] / exp) % base;
        if (c < 0) c += base;
        przyp(out[cnt[c] - 1], v[i]);
        cnt[c]--;
    }

    //Kopiowanie posortowanej tablicy z powrotem do v
    for (int i = 0; i < n; i++)
        przyp(v[i], out[i]);
}
```

Funkcja wykonuje jeden pełny etap sortowania pozycyjnego, nie wykonuje żadnych porównań między wartościami, przez co RadixSort jest algorytmem nieporównawczym.

2.4 Sortowanie kubełkowe

```
void sortowanie_w_kubelku_przez_wstawianie(vector<double>& b) {
    for (int i = 1; i < b.size(); i++) {
        double klucz = b[i];
        g_liczniki.przypisania++;
        int j = i - 1;

        while (j >= 0 && porownaj_wieksze(b[j], klucz)) {
            przypisz(b[j+1], b[j]);
            j--;
        }
        przypisz(b[j+1], klucz);
    }
}
```

3 Wykresy

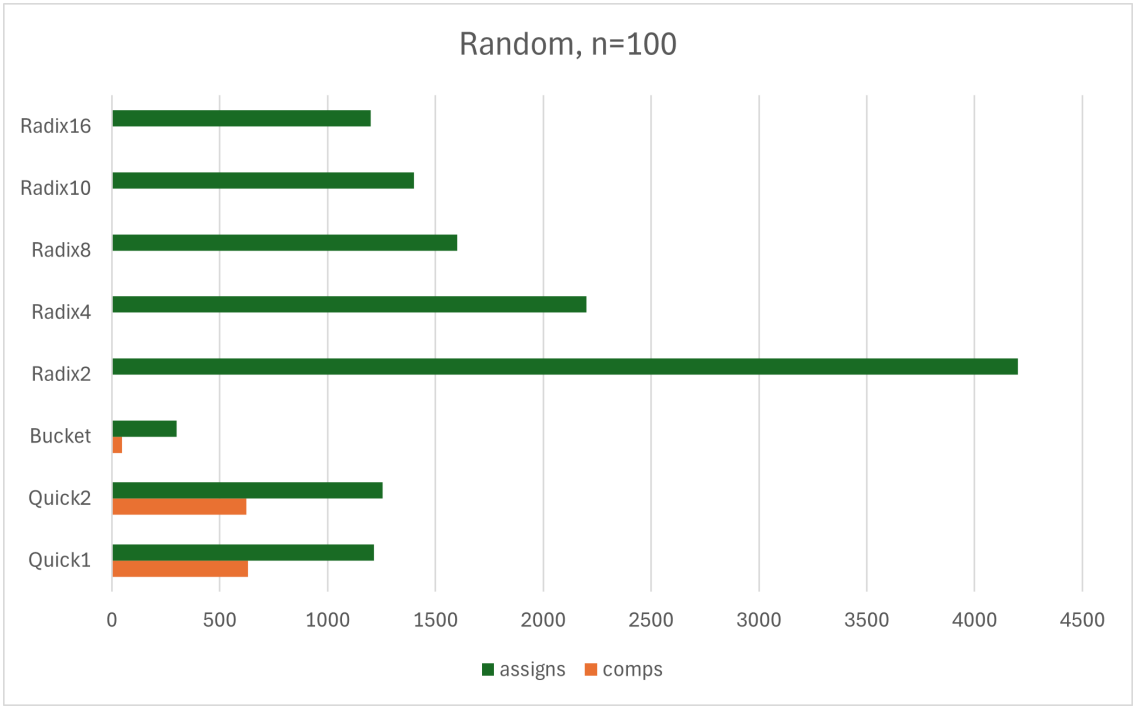

Figure 1: Porównanie liczby przypisań i porównań dla różnych algorytmów. n=100, random

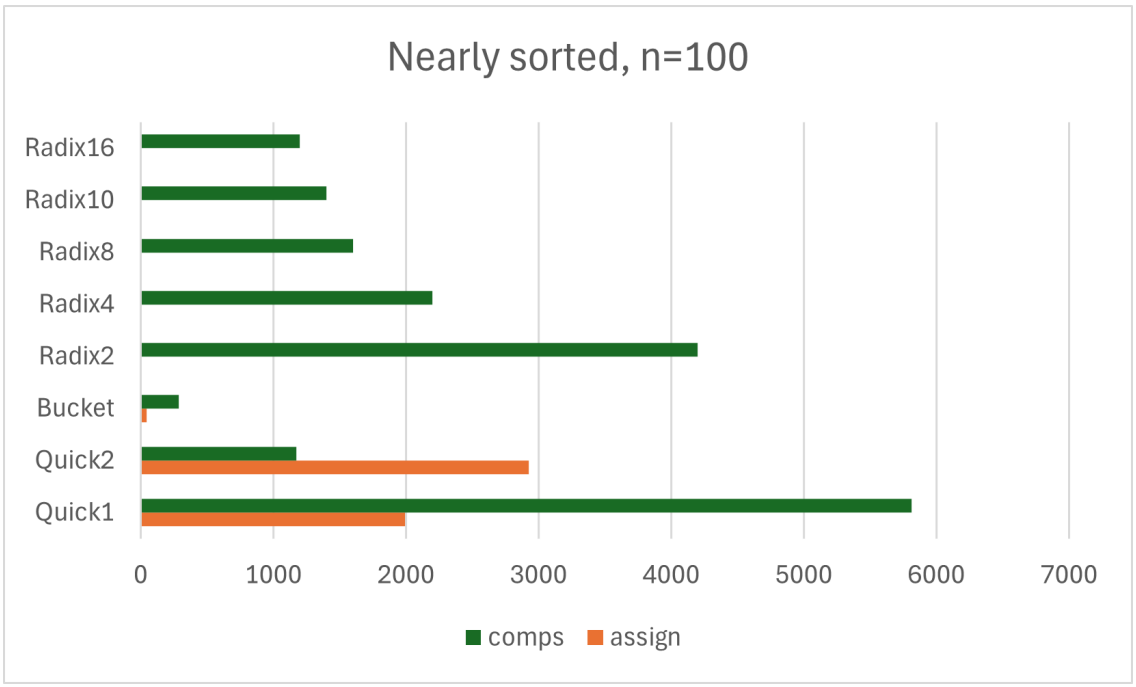

Figure 2: Porównanie liczby przypisań i porównań dla różnych algorytmów. n=100, nearly sorted

4 Podsumowanie

- QuickSort z jednym pivotem generuje największą liczbę przypisań, ponieważ wykonuje wiele zamian elementów podczas podziału tablicy.
- QuickSort z dwoma pivotami zachowuje się nieco bardziej stabilnie i, wykonuje mniej przypisań niż jednopivotowy.
- Bucket sort wykonuje stosunkowo niewiele operacji porównania. Liczba przypisań zależy od tego, ile elementów trafia do poszczególnych kubełków. Wyniki pokazują, że przy danych zbliżonych do równomiernego rozkładu działa on stabilnie i najwydajniej.
- Przy danych NearlySorted (rys. 2) widać wyraźnie, że QuickSort (1 pivot) wykonuje znacznie więcej przypisań niż dla danych losowych, ponieważ pivot wybierany na końcu tablicy jest niekorzystny przy danych prawie uporządkowanych .