# 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 Tabele wyników

Table 1: Fragment danych (Random)

| Scen. | Algorytm | Run | n | Por. | Przyp. | Czas (ms) |
|---|---|---|---|---|---|---|
| Random | Radix2 | 0 | 100 | 0 | 4200 | 0 |
| Random | Radix4 | 0 | 100 | 0 | 2200 | 0 |
| Random | Radix8 | 0 | 100 | 0 | 1600 | 0 |
| Random | Radix10 | 0 | 100 | 0 | 1400 | 0 |
| Random | Radix16 | 0 | 100 | 0 | 1200 | 0 |
| Random | Radix2 | 0 | 200 | 0 | 8400 | 0 |
| Random | Radix4 | 0 | 200 | 0 | 4400 | 0 |
| Random | Radix8 | 0 | 200 | 0 | 3200 | 0 |
| Random | Radix10 | 0 | 200 | 0 | 2800 | 0 |
| Random | Radix16 | 0 | 200 | 0 | 2400 | 0 |
| Random | Radix2 | 0 | 500 | 0 | 21000 | 0 |
| Random | Radix4 | 0 | 500 | 0 | 11000 | 0 |
| Random | Radix8 | 0 | 500 | 0 | 8000 | 0 |
| Random | Radix10 | 0 | 500 | 0 | 7000 | 0 |
| Random | Radix16 | 0 | 500 | 0 | 6000 | 0 |
| Random | Radix2 | 0 | 1000 | 0 | 42000 | 0 |
| Random | Radix4 | 0 | 1000 | 0 | 22000 | 0 |
| Random | Radix8 | 0 | 1000 | 0 | 16000 | 0 |
| Random | Radix10 | 0 | 1000 | 0 | 14000 | 0 |
| Random | Radix16 | 0 | 1000 | 0 | 12000 | 0 |
| Random | Radix2 | 0 | 2000 | 0 | 84000 | 1 |
| Random | Radix4 | 0 | 2000 | 0 | 44000 | 0 |
| Random | Radix8 | 0 | 2000 | 0 | 32000 | 0 |
| Random | Radix10 | 0 | 2000 | 0 | 28000 | 0 |
| Random | Radix16 | 0 | 2000 | 0 | 24000 | 0 |
| Random | Radix2 | 0 | 5000 | 0 | 210000 | 3 |
| Random | Radix4 | 0 | 5000 | 0 | 110000 | 2 |
| Random | Radix8 | 0 | 5000 | 0 | 80000 | 1 |
| Random | Radix10 | 0 | 5000 | 0 | 70000 | 1 |
| Random | Radix16 | 0 | 5000 | 0 | 60000 | 1 |
| NearlySorted | Radix2 | 0 | 100 | 0 | 4200 | 0 |
| NearlySorted | Radix4 | 0 | 100 | 0 | 2200 | 0 |
| NearlySorted | Radix8 | 0 | 100 | 0 | 1600 | 0 |
| NearlySorted | Radix10 | 0 | 100 | 0 | 1400 | 0 |
| NearlySorted | Radix16 | 0 | 100 | 0 | 1200 | 0 |
| NearlySorted | Radix2 | 0 | 200 | 0 | 8400 | 0 |
| NearlySorted | Radix4 | 0 | 200 | 0 | 4400 | 0 |
| NearlySorted | Radix8 | 0 | 200 | 0 | 3200 | 0 |
| NearlySorted | Radix10 | 0 | 200 | 0 | 2800 | 0 |
| NearlySorted | Radix16 | 0 | 200 | 0 | 2400 | 0 |
| NearlySorted | Radix2 | 0 | 500 | 0 | 21000 | 0 |
| NearlySorted | Radix4 | 0 | 500 | 0 | 11000 | 0 |
| NearlySorted | Radix8 | 0 | 500 | 0 | 8000 | 0 |
| NearlySorted | Radix10 | 0 | 500 | 0 | 7000 | 0 |
| NearlySorted | Radix16 | 0 | 500 | 0 | 6000 | 0 |
| NearlySorted | Radix2 | 0 | 1000 | 0 | 42000 | 0 |
| NearlySorted | Radix4 | 0 | 1000 | 0 | 22000 | 0 |
| NearlySorted | Radix8 | 0 | 1000 | 0 | 16000 | 0 |
| NearlySorted | Radix10 | 0 | 1000 | 0 | 14000 | 0 |
| NearlySorted | Radix16 | 0 | 1000 | 0 | 12000 | 0 |
| NearlySorted | Radix2 | 0 | 2000 | 0 | 84000 | 1 |
| NearlySorted | Radix4 | 0 | 2000 | 0 | 44000 | 0 |
| NearlySorted | Radix8 | 0 | 2000 | 0 | 32000 | 0 |

# 4 Wykresy

# 5 Podsumowanie

Zaimplementowane algorytmy wykazują różne właściwości, co dobrze obrazuje analiza czasu, liczby porównań i przypisań.