Sprawozdanie

Yahor Sharshniou

December 2025

1 Porównanie wydajności

W tym rozdziale będę porównywał ilość przepisań i porównań quick sort'a, radix + counting oraz bucket (zmodyfikowany) + insertion. Dla generacji zbiorów użyłem tej strony: https://randomus.ru/

1.1 10 liczb {72, 24, 32, 66, 67, 99, 77, 12, 38, 87}

	Quick sort	Radix Sort	Bucket Sort
Porównania	25	9	19
Przypisania	57	41	24

1.2 10 liczb {69634, 10491, 5928, 93618, 16266, 71902, 77171, 45922, 88858, 62746}

	Quick sort	Radix Sort	Bucket Sort
Porównania	25	9	19
Przypisania	54	101	24

Wniosek: w moich obliczeniach Radix + Counting nie ma porównań, aktualna liczba wychodzi z funkcji getMax, która przechodzi przez cały szereg, wybierając maksymalną liczbę.

1.3 10 liczb {3, 5, 2, 6, 1, 10, 8, 7, 4, 9}

	Quick sort	Radix Sort	Bucket Sort
Porównania	23	9	19
Przypisania	60	43	26

Wniosek: Chociaż w samym Radix'e nie ma porównań ani przypisań, ponieważ jest stworzony dla liczb z kilkoma cyframi, częściej wywołuje Counting Sort, w którym nadal nie ma porównań, ale są przypisania.

1.4 10 liczb {98047, 520044, 762501, 155013, 331320, 498483, 5188, 109916, 693245, 754312}

	Quick sort	Radix Sort	Bucket Sort
Porównania	29	9	21
Przypisania	72	122	26

1.5 1000 liczb od 1 do 1.000.000 (7 raz)

	Quick sort	Radix Sort	Bucket Sort
Porównania	10185	999	2244
Przypisania	15108	12007	2262
Porównania	10446	999	2205
Przypisania	19446	12009	2222
Porównania	11147	999	2254
Przypisania	18783	12007	2268
Porównania	12862	999	2255
Przypisania	17571	12006	2267
Porównania	11481	999	2252
Przypisania	16167	12006	2267
Porównania	10701	999	2239
Przypisania	17034	12006	2255
Porównania	12459	999	2250
Przypisania	17841	12007	2265

Wniosek: Widać z tego, że Radix + Counting nie zależy od liczb, lecz od ilości cyfr największej liczby. Bucket Sort wygląda mniej więcej stabilnie, bo liczby w tablicy były wygenerowany randomowo. Zagęszczanie liczb w jednej części przedziału spowodowałoby znacznie mniej stabilne wyniki. Quick sort w tych testach wygląda najgorzej.

1.6 1000 liczb od 1 do 2.000 (5 raz)

	Quick sort	Radix Sort	Bucket Sort
Porównania	10241	999	2127
Przypisania	17676	8003	2141
Porównania	12194	999	2130
Przypisania	22125	8007	2143
Porównania	9882	999	2130
Przypisania	16089	8009	2148
Porównania	11245	999	2136
Przypisania	17943	8003	2150
Porównania	10560	999	2124
Przypisania	17943	8005	2137

1.7 1000 liczb od 1 do 10 (5 raz)

	Quick sort	Radix Sort	Bucket Sort
Porównania	53140	999	1998
Przypisania	157617	4003	2005
Porównania	54102	999	1998
Przypisania	159945	4003	2004
Porównania	54102	999	1998
Przypisania	157842	4004	2005
Porównania	54810	999	1998
Przypisania	159405	4003	2006
Porównania	54707	999	1998
Przypisania	159405	4003	2005

Wniosek: Quick sort źle sobie radzi z powtórzeniami, natomiast Radix + Counting ma dwa razy mniej przypisań. Bucket też ustabilizował się, prawdopodobnie dlatego, że na takim przedziale ma tylko 10 bucketów z tymi samymi liczbami.

1.8 1000 liczb od 1 do 3

	Quick sort	Radix Sort	Bucket Sort
Porównania	167520	999	1998
Przypisania	502650	2002	2002

Wniosek: Tak, Quick Sort źle sobie radzi z powtórzeniami. Partition umieszcza większość powtarzających się wartości w jednej części, co powoduje, że rekurencja nie zmniejsza istotnie rozmiaru podtablicy.

2 Najciekawszy element kodów

Spośród omawianych algorytmów najciekawszy jest Counting Sort. Sortuje on liczby bez bezpośredniego porównywania elementów, wykorzystując wartości jako indeksy w tablicy o rozmiarze odpowiadającym zakresowi danych, można powiedzieć poetycko, że „odwraca indeksy i wartości pod nimi". W przypadku bardzo dużych liczb takie podejście staje się niepraktyczne, ale w połączeniu z Radix Sort, który ogranicza zakres wartości do 10 na każdym etapie, powstaje niezwykle efektywny sposób sortowania.

```
int count[10] = { 0 };
for (i = 0; i < n; i++){
    count[(A[i] / exp) % 10] = count[(A[i] / exp) % 10] + 1;
}
for (i = 1; i < 10; i++){
    count[i] = count[i] + count[i - 1];
}
```

Na tym przykładzie kodu widać, że Counting sort wykorzystuje wartości elementów jako indeksy do zliczania i umieszczania ich w posortowanym zbiorze.