|  |  |  |  |
| --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  | **AKADEMIA GÓRNICZO-HUTNICZA**  **im. Stanisława Staszica w Krakowie**  **WYDZIAŁ ZARZĄDZANIA** | wz_znak | |
| Sortowanie tablic jednowymiarowych bez porównywania elementów |
|  |
|  |
| **Autorzy**: Anna Golak, Katarzyna Wróbel  **Przedmiot**: Programowanie komputerów  **Informatyka i ekonometria, rok 1.** |
|  |

|  |
| --- |
| Kraków, 4 czerwca 2023 r. |

# Cel projektu

Przygotowanie 4 algorytmów, o różnej złożoności czasowej, sortujących elementy tablicy jednowymiarowej bez porównywania elementów.

# Opis programu

Program zawiera funkcje wykorzystujące następujące sortowania: sortowanie pozycyjne (radix sort), sortowanie kubełkowe (bucket sort), sortowanie szybkie (quick sort) oraz sortowanie przez zliczanie (counting sort) oraz funkcje, które są później wykorzystywane w funkcjach sortujących. Ponadto znajdziemy tutaj funkcje, które umożliwiają nam: wypisanie tablicy na ekran, wypełnienie tablicy losowymi liczbami, odczytanie tablicy z pliku oraz zapisanie tablicy do pliku. Funkcja choice() pozwala użytkownikowi na wybór wczytania danych i dobiera optymalny algorytm sortowania. Kryteria doboru sortowania uwzględniają rodzaj danych (dziesiętne vs. zmiennoprzecinkowe), stabilność sortowania oraz liczbę elementów w tablicy. Algorytmy radix sort i counting sort są wybrane dla danych dziesiętnych ze względu na ich specyfikę, a quick sort jest wybrany dla danych losowych i nie gwarantuje stabilności sortowania. Program daje użytkownikowi możliwość zapisania posortowanej tablicy do pliku.

# Opis poszczególnych algorytmów

## Sortowanie pozycyjne

Algorytm radix sort to stabilny algorytm sortowania, który sortuje liczby po cyfrach, zaczynając od najmniej znaczących cyfr i przechodząc stopniowo do najbardziej znaczących cyfr. Działa na zasadzie sortowania przez zliczanie (counting sort), ale zamiast sortować na podstawie całych liczb, sortuje po pojedynczych cyfrach.

Opis działania algorytmu radix sort:

1. Znajdź maksymalną liczbę w tablicy (maksimum musi być znane, aby określić liczbę cyfr).
2. Dla każdej pozycji cyfry od najmniej znaczącej do najbardziej znaczącej: a. Użyj counting sort (sortowanie przez zliczanie) do posortowania liczb na podstawie aktualnej pozycji cyfry. b. Aktualizuj tablicę wejściową z posortowanymi liczbami na podstawie wyniku counting sort.
3. Po posortowaniu wszystkich cyfr, tablica wejściowa będzie uporządkowana.

Złożoność czasowa radix sort wynosi O(d \* (n + k)), gdzie:

* n to liczba elementów w tablicy,
* d to liczba cyfr w maksymalnej liczbie,
* k to liczba możliwych wartości dla każdej cyfry (dla systemu dziesiętnego k=10).

## Sortowanie kubełkowe

Algorytm sortowania kubełkowego (bucket sort) jest algorytmem sortowania, który rozdziela elementy na kubełki na podstawie ich wartości, a następnie sortuje każdy kubełek osobno. Jest to algorytm efektywny, gdy elementy są równomiernie rozłożone w przedziale wartości.

Opis działania algorytmu sortowania kubełkowego:

1. Określ przedział wartości, na przykład od minimum do maksimum z elementów tablicy.
2. Utwórz puste kubełki w ilości odpowiadającej liczbie elementów.
3. Przejdź przez tablicę elementów i umieść każdy element w odpowiednim kubełku na podstawie jego wartości.
4. Posortuj każdy kubełek osobno.
5. Po posortowaniu każdego kubełka, połącz zawartość kubełków w jedną posortowaną tablicę.

Złożoność sortowania kubełkowego wynosi O(n^2), gdzie:

* n to liczba elementów w tablicy,

## Sortowanie szybkie

Quick sort to jedna z najbardziej efektywnych i popularnych algorytmów sortowania stosowanych w informatyce. Działa na zasadzie "dziel i zwyciężaj" i wykorzystuje rekurencję do sortowania elementów w tablicy.

Działanie Quick sort można opisać w następujących krokach:

1. Wybierz jeden element z tablicy, który nazywamy "elementem podziału" (pivotem). Może to być dowolny element, ale najczęściej wybiera się pierwszy lub ostatni element tablicy.
2. Podziel tablicę na dwa podzbiory: elementy mniejsze niż pivot i elementy większe niż pivot. W tym celu porównuje się każdy element tablicy z pivotem i umieszcza się je odpowiednio po lewej lub prawej stronie pivotu.
3. Rekurencyjnie sortuj oba podzbiory. Wykonuje się tę samą operację podziału i sortowania na podzbiorach, aż do osiągnięcia podzbiorów jednoelementowych.
4. Po posortowaniu podzbiorów scal je w jedną posortowaną tablicę. Po lewej stronie umieszcza się elementy mniejsze niż pivot, następnie pivot, a na końcu elementy większe niż pivot.

Złożoność sortowania szybkiego wynosi O(nlogn), gdzie

* n to liczba elementów w tablicy.

## Sortowanie przez zliczanie

Sortowanie przez zliczanie (Counting Sort) to algorytm sortowania, który nie opiera się na porównywaniu elementów, ale polega na zliczaniu wystąpień poszczególnych elementów i tworzeniu posortowanej tablicy na podstawie tych informacji.

Działanie sortowania przez zliczanie można opisać w następujących krokach:

1. Znajdź zakres wartości elementów w tablicy i stwórz pomocniczą tablicę zliczającą o rozmiarze odpowiadającym temu zakresowi.
2. Przejdź przez tablicę wejściową i zlicz wystąpienia każdej wartości, inkrementując odpowiednie elementy w tablicy zliczającej.
3. Na podstawie tablicy zliczającej utwórz tablicę wynikową, umieszczając wartości w odpowiedniej kolejności. Może to być osiągnięte poprzez iterację po tablicy zliczającej i wielokrotne dodanie odpowiednich wartości do tablicy wynikowej.

Złożoność sortowania przez zliczanie wynosi O(n + k), gdzie:

* n to liczba elementów w tablicy,
* k to zakres wartości elementów.

# Zbadanie efektywności poszczególnych algorytmów

Badamy średnie czasy wykonania dla różnych długości ciągów 10≤n≤1000000. Mierzymy czas w sekundach. Górny wiersz zawiera nazwę rodzaju sortowania i liczbę elementów w tablicy. Testy były wykonywane w ten sposób, że dla tablice o różnych długościach były sortowane określonym algorytmem. Pomiar powtórzyłyśmy 10 razy i wyliczyłyśmy średnią.

* 1. Tablice wypełnione liczbami losowymi

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Quick sort** | **1000** | **10000** | **100000** | **250000** |
| 1 | 0,003 | 0,002 | 0,021 | 0,055 |
| 2 | 0,001 | 0,006 | 0,021 | 0,072 |
| 3 | 0,001 | 0,003 | 0,02 | 0,054 |
| 4 | 0,002 | 0,003 | 0,02 | 0,054 |
| 5 | 0,001 | 0,003 | 0,024 | 0,056 |
| 6 | 0,001 | 0,005 | 0,022 | 0,053 |
| 7 | 0,002 | 0,002 | 0,027 | 0,067 |
| 8 | 0,001 | 0,006 | 0,024 | 0,056 |
| 9 | 0,001 | 0,003 | 0,021 | 0,058 |
| 10 | 0,001 | 0,002 | 0,024 | 0,059 |
| **ŚREDNIA:** | **0,00140** | **0,00350** | **0,02240** | **0,05840** |

Dla 100 elementów nie jesteśmy w stanie zmierzyć czasów- są one poniżej 1/1000 sekundy.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Counting sort** | **1000** | **10000** | **100000** | **250000** |
| 1 | 0,001 | 0,002 | 0,014 | 0,013 |
| 2 | 0,001 | 0,002 | 0,013 | 0,039 |
| 3 | 0,001 | 0,001 | 0,007 | 0,033 |
| 4 | 0,002 | 0,003 | 0,013 | 0,036 |
| 5 | 0,002 | 0,004 | 0,014 | 0,03 |
| 6 | 0,002 | 0,003 | 0,013 | 0,024 |
| 7 | 0,001 | 0,002 | 0,014 | 0,028 |
| 8 | 0,001 | 0,002 | 0,016 | 0,043 |
| 9 | 0,001 | 0,002 | 0,014 | 0,039 |
| 10 | 0,002 | 0,002 | 0,019 | 0,034 |
| **ŚREDNIA:** | **0,0014** | **0,00230** | **0,0137** | **0,03190** |

Dla 10 i 100 nie jesteśmy w stanie zmierzyć czasów- są one poniżej 1/1000 sekundy.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Radix sort** | **1000** | **10000** | **100000** | **250000** |
| 1 | - | 0,001 | 0,016 | 0,034 |
| 2 | - | 0,001 | 0,018 | 0,028 |
| 3 | - | 0,001 | 0,013 | 0,03 |
| 4 | - | 0,001 | 0,017 | 0,039 |
| 5 | - | 0,001 | 0,016 | 0,033 |
| 6 | - | 0,001 | 0,012 | 0,029 |
| 7 | - | 0,002 | 0,009 | 0,036 |
| 8 | - | 0,002 | 0,013 | 0,035 |
| 9 | - | 0,001 | 0,018 | 0,039 |
| 10 | - | 0,001 | 0,006 | 0,029 |
| **ŚREDNIA:** |  | **0,00120** | **0,01380** | **0,03320** |

Tutaj nie tylko dla 10 i 100, ale też dla 1000 nie jesteśmy w stanie zmierzyć czasów- są one poniżej 1/1000 sekundy.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Bucket sort** | **1000** | **10000** | **100000** | **250000** |
| 1 | 0,001 | 0,004 | 0,05 | 0,137 |
| 2 | 0,001 | 0,005 | 0,05 | 0,121 |
| 3 | 0,001 | 0,005 | 0,04 | 0,091 |
| 4 | 0,001 | 0,006 | 0,057 | 0,142 |
| 5 | 0,001 | 0,004 | 0,056 | 0,133 |
| 6 | 0,001 | 0,008 | 0,056 | 0,141 |
| 7 | 0,001 | 0,006 | 0,056 | 0,112 |
| 8 | 0,001 | 0,006 | 0,059 | 0,127 |
| 9 | 0,001 | 0,005 | 0,044 | 0,122 |
| 10 | 0,001 | 0,007 | 0,059 | 0,15 |
| **ŚREDNIA:** | **0,00100** | **0,00560** | **0,05270** | **0,12760** |

Dla 10 i 100 nie jesteśmy w stanie zmierzyć czasów- są one poniżej 1/1000 sekundy. Sortuję przy liczbie kubeł równej 100.

Na podstawie danych możemy wyciągnąć następujące wnioski:

Quick Sort:

* Średni czas wykonania jest stosunkowo niski dla wszystkich rozmiarów tablic.
* Quick Sort wydaje się być ogólnie efektywnym algorytmem sortowania, który utrzymuje swoją wydajność, nawet dla większych rozmiarów tablic.

Counting Sort:

* Czas wykonania jest bardzo niski dla wszystkich rozmiarów tablic.
* Jednakże, nie jest uniwersalnym rozwiązaniem i może mieć ograniczenia związane z zakresem wartości elementów.

Radix Sort:

* Czas wykonania jest niższy niż w przypadku Quick Sort dla mniejszych rozmiarów tablic, ale nieco wyższy dla większych rozmiarów.
* Złożoność czasowa Radix Sort zależy od liczby cyfr w największym elemencie, co może wpływać na czas wykonania.

Bucket Sort:

* Czas wykonania jest zazwyczaj wyższy niż w przypadku pozostałych algorytmów dla wszystkich rozmiarów tablic.
* Wyniki wskazują na to, że Bucket Sort ma największą złożoność czasową spośród analizowanych algorytmów.
* Pomimo tego, dla pewnych przypadków i specyficznych rodzajów danych, Bucket Sort może być efektywny.

Ogólnie rzecz biorąc, każdy algorytm sortowania ma swoje własne cechy i efektywność w zależności od rodzaju danych i rozmiaru tablicy. Quick Sort jest często preferowanym algorytmem, ponieważ jest ogólnie efektywny, ale dla określonych przypadków można rozważyć zastosowanie innych algorytmów sortowania, takich jak Counting Sort, Radix Sort lub Bucket Sort, które mogą być bardziej optymalne w niektórych sytuacjach.

* 1. Tablice posortowane

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Radix sort- tablica posortowana** | **1000** | **10000** | **100000** | **250000** |
| 1 |  | 0,002 | 0,011 | 0,028 |
| 2 |  | 0,001 | 0,009 | 0,027 |
| 3 |  | 0,001 | 0,008 | 0,028 |
| 4 |  | 0,001 | 0,008 | 0,028 |
| 5 |  | 0,001 | 0,009 | 0,035 |
| 6 |  | 0,001 | 0,01 | 0,027 |
| 7 |  | 0,001 | 0,009 | 0,03 |
| 8 |  | 0,001 | 0,012 | 0,029 |
| 9 |  | 0,001 | 0,011 | 0,032 |
| 10 |  | 0,002 | 0,008 | 0,031 |
| **ŚREDNIA:** |  | **0,00120** | **0,00950** | **0,02950** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Counting sort- tablica posortowana** | **1000** | **10000** | **100000** | **250000** |
| 1 |  |  | 0,001 | 0,005 |
| 2 |  |  | 0,001 | 0,005 |
| 3 |  |  | 0,002 | 0,004 |
| 4 |  |  | 0,001 | 0,005 |
| 5 |  |  | 0,002 | 0,003 |
| 6 |  |  | 0,001 | 0,01 |
| 7 |  |  | 0,001 | 0,005 |
| 8 |  |  | 0,002 | 0,005 |
| 9 |  |  | 0,001 | 0,004 |
| 10 |  |  | 0,001 | 0,005 |
| **ŚREDNIA:** |  |  | **0,00130** | **0,00510** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Bucket sort- tablica posortowana** | **1000** | **10000** | **100000** | **250000** |
| 1 |  | 0,001 | 0,016 | 0,044 |
| 2 |  | 0,002 | 0,018 | 0,046 |
| 3 |  | 0,002 | 0,016 | 0,051 |
| 4 |  | 0,001 | 0,017 | 0,049 |
| 5 |  | 0,001 | 0,019 | 0,053 |
| 6 |  | 0,001 | 0,017 | 0,047 |
| 7 |  | 0,001 | 0,016 | 0,045 |
| 8 |  | 0,002 | 0,015 | 0,05 |
| 9 |  | 0,001 | 0,018 | 0,052 |
| 10 |  | 0,001 | 0,017 | 0,053 |
| **ŚREDNIA:** |  | **0,00130** | **0,01690** | **0,04900** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Quick sort- tablica odwrócona** | **1000** | **10000** | **100000** | **250000** |
| 1 |  | 0,001 | 0,013 | 0,07 |
| 2 |  | 0,001 | 0,014 | 0,066 |
| 3 |  | 0,001 | 0,012 | 0,066 |
| 4 |  | 0,001 | 0,013 | 0,07 |
| 5 |  | 0,002 | 0,016 | 0,068 |
| 6 |  | 0,001 | 0,016 | 0,065 |
| 7 |  | 0,002 | 0,017 | 0,065 |
| 8 |  | 0,001 | 0,017 | 0,063 |
| 9 |  | 0,001 | 0,02 | 0,061 |
| 10 |  | 0,001 | 0,016 | 0,068 |
| **ŚREDNIA:** |  | **0,00120** | **0,01540** | **0,06620** |

Na podstawie tych danych możemy wyciągnąć następujące wnioski:

Radix Sort:

* Czas wykonania jest stosunkowo niski dla wszystkich rozmiarów tablic.
* Jest stabilnym algorytmem sortowania, który działa w czasie liniowym.
* W przypadku większych rozmiarów tablic (100000), Radix Sort utrzymuje swoją efektywność.

Counting Sort:

* Dane dla mniejszych rozmiarów tablic czas wykonywania jest na tyle mały, że nie zdołałyśmy go zmierzyć.
* Dla większych rozmiarów tablic (100000), czas wykonania Counting Sort jest niski.
* Widzimy, że dla tablic posortowanych ten algorytm jest wydajny.

Bucket Sort:

* Czas wykonania jest zazwyczaj wyższy niż w przypadku innych algorytmów sortowania dla wszystkich rozmiarów tablic.
* Mimo że Bucket Sort jest mniej efektywny w porównaniu do innych algorytmów, dla większych rozmiarów tablic (100000) nadal utrzymuje się na akceptowalnym poziomie.

Quick Sort:

* Średni czas wykonania jest stosunkowo niski dla wszystkich rozmiarów tablic.
* Quick Sort jest ogólnie efektywnym algorytmem sortowania, który utrzymuje swoją wydajność dla różnych rozmiarów tablic.

Podsumowując, Radix Sort i Quick Sort prezentują się jako skuteczne i wydajne algorytmy sortowania dla różnych rozmiarów tablic. Counting Sort jest również efektywny dla większych rozmiarów tablic, pod warunkiem, że zakres wartości elementów jest odpowiednio obsłużony. Bucket Sort może być mniej efektywny w porównaniu do innych algorytmów, ale nadal może być rozważany w pewnych przypadkach, szczególnie dla mniejszych rozmiarów tablic.

* 1. Tablice odwrócone

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Bucket sort- tablica odwrócona** | **1000** | **10000** | **100000** | **250000** |
| 1 |  | 0,002 | 0,016 | 0,039 |
| 2 |  | 0,001 | 0,018 | 0,039 |
| 3 |  | 0,002 | 0,02 | 0,047 |
| 4 |  | 0,001 | 0,019 | 0,04 |
| 5 |  | 0,002 | 0,018 | 0,047 |
| 6 |  | 0,002 | 0,016 | 0,04 |
| 7 |  | 0,001 | 0,019 | 0,048 |
| 8 |  | 0,002 | 0,016 | 0,054 |
| 9 |  | 0,001 | 0,017 | 0,042 |
| 10 |  | 0,001 | 0,018 | 0,041 |
| **ŚREDNIA:** |  | **0,00150** | **0,01770** | **0,04370** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Counting sort- tablica odwrócona** | **1000** | **10000** | **100000** | **250000** |
| 1 |  |  | 0,002 | 0,005 |
| 2 |  |  | 0,001 | 0,002 |
| 3 |  |  | 0,002 | 0,005 |
| 4 |  |  | 0,002 | 0,006 |
| 5 |  |  | 0,002 | 0,005 |
| 6 |  |  | 0,002 | 0,004 |
| 7 |  |  | 0,002 | 0,005 |
| 8 |  |  | 0,001 | 0,003 |
| 9 |  |  | 0,001 | 0,004 |
| 10 |  |  | 0,002 | 0,006 |
| **ŚREDNIA:** |  |  | **0,00170** | **0,00450** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Radix sort- tablica odwrócona** | **1000** | **10000** | **100000** | **250000** |
| 1 |  | 0,002 | 0,013 | 0,036 |
| 2 |  | 0,001 | 0,012 | 0,04 |
| 3 |  | 0,001 | 0,014 | 0,033 |
| 4 |  | 0,001 | 0,012 | 0,036 |
| 5 |  | 0,002 | 0,013 | 0,035 |
| 6 |  | 0,001 | 0,02 | 0,034 |
| 7 |  | 0,001 | 0,012 | 0,036 |
| 8 |  | 0,001 | 0,013 | 0,033 |
| 9 |  | 0,001 | 0,012 | 0,041 |
| 10 |  | 0,001 | 0,011 | 0,031 |
| **ŚREDNIA:** |  | **0,00120** | **0,01320** | **0,03550** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Quick sort- tablica posortowana** | **1000** | **10000** | **100000** | **250000** |
| 1 |  | 0,001 | 0,008 | 0,025 |
| 2 |  | 0,001 | 0,013 | 0,047 |
| 3 |  | 0,001 | 0,01 | 0,056 |
| 4 |  | 0,001 | 0,013 | 0,041 |
| 5 |  | 0,002 | 0,013 | 0,036 |
| 6 |  | 0,001 | 0,011 | 0,025 |
| 7 |  | 0,001 | 0,02 | 0,037 |
| 8 |  | 0,001 | 0,009 | 0,055 |
| 9 |  | 0,002 | 0,011 | 0,045 |
| 10 |  | 0,001 | 0,014 | 0,042 |
| **ŚREDNIA:** |  | **0,00120** | **0,01220** | **0,04090** |

Na podstawie danych możemy wyciągnąć następujące wnioski:

Bucket Sort:

* Średni czas wykonania jest stosunkowo niski dla wszystkich rozmiarów tablic.
* Pomimo że czas wykonania Bucket Sort jest nieco wyższy niż dla innych algorytmów, wciąż utrzymuje się na akceptowalnym poziomie.

Counting Sort:

* Średni czas wykonania jest stosunkowo niski dla większych rozmiarów tablic (10000, 100000).
* Dla mniejszych rozmiarów tablic czas wykonania jest tak niski, że nie byliśmy w stanie go zmierzyć.

Radix Sort:

* Średni czas wykonania jest stosunkowo niski dla wszystkich rozmiarów tablic.
* Radix Sort nadal prezentuje się jako skuteczny algorytm sortowania, zwłaszcza dla większych rozmiarów tablic.

Quick Sort:

* Średni czas wykonania jest stosunkowo niski dla wszystkich rozmiarów tablic.
* Quick Sort nadal prezentuje się jako efektywny algorytm sortowania, który utrzymuje swoją wydajność, nawet dla większych rozmiarów tablic.

Podsumowując, na podstawie średniego czasu wykonania dla każdego algorytmu sortowania, Quick Sort, Bucket Sort, Counting Sort i Radix Sort wydają się być skutecznymi algorytmami sortowania dla różnych rozmiarów tablic. Quick Sort jest ogólnie efektywny, Bucket Sort   
i Counting Sort mają niski czas wykonania, a Radix Sort jest również efektywny, zwłaszcza dla większych rozmiarów tablic.

Specyfikacja urządzenia, na którym były wykonywane testy:

Nazwa Urządzenia: LAPTOP – ME7TTOIL  
Procesor: 11th Gen Intel ® Core™ i5-11300H @ 3.10GHz 3.11 GHz  
Zainstalowana pamięć RAM 16.0 GB (dostępne 15,7 GB)  
Identyfikator urządzenia: E9BEA7D9-9F55-4649-B95E-EBF110E1617E  
Identyfikator produktu: 00325-97273-74272-AAOEM  
Typ systemu: 64-bitowy system operacyjny, procesor x64

Podział pracy:

* 1. Anna Golak – wyszukanie i wybranie algorytmów, przygotowanie algorytmu quick sort, radix sort, bucket sort, testowanie czasu działania, przygotowanie kodu dobierającego algorytm do tablicy, funkcja wczytująca z pliku i zapisująca do pliku (wspólnie), obróbka kodu i usunięcie błędów, ostrzeżeń
  2. Katarzyna Wróbel – sporządzenie sprawozdania, przygotowanie algorytmu sortowania przez zliczanie, funkcja wczytująca z pliku i zapisująca do pliku (wspólnie), przygotowanie funkcji dającej wybór czytelnikowi na rodzaj wpisywania danych oraz zapisania danych do plików, obróbka kodu i usunięcie błędów, ostrzeżeń, przygotowanie estetyczne kodu.