K-ty element

Weronika Orzechowska

10 marca 2025

1 Zadanie

Napisz programy, które przetestują kilka algorytmów wybierania k-tego elementu z tablicy rozmiaru n. Twój program powinien działać najszybciej jak to tylko możliwe (optylanie w czasie $O(n)$ z jak najmniejszą stałą przy n). Przetestuj kilka rozwiązań i zobacz w jakim rzeczywistym czasie działają na kolejnych plikach wejściowych:

- algorytm oparty na algorytmie sortowania przez proste wstawianie,
- algorytm oparty na szybkim algorytmie sortowania,
- najlepszy deterministyczny algorytm (algorytm magicznych piątek),
- algorytm probabilistyczny oparty na próbkowaniu.

2 Implementacja

2.1 Proste wstawianie

Pierwszym zaimplementowanym algorytmem był Insertion Sort, czyli prosty algorytm przez wstawianie. Polega on na wstawianiu kolejnych elementów do posortowanego już fragmentu ciągu. W związku z tym nie była możliwa optymalizacja ze względu na szukanie k-tego elementu, ponieważ taki element w przypadku pesymistycznym znajdowałby się na samym końcu listy i trafiłby na właściwą (k-tą) pozycję dopiero na koniec sortowania.

W związku z tym, w ramach eksperymentu zaimplementowałam drugi prosty algorytm: Selection Sort, czyli sortowanie przez wybór. W tym przypadku, ponieważ w każdym kroku szukamy kolejnego najmniejszego elementu, nie musimy sortować całej listy, a wystarczy, że zatrzymamy się po k przeszukaniach.

2.2 Szybkie sortowanie

Kolejnym zaimplementowanym algorytmem był Quick Sort. W ramach tej funkcji dodałam trzy opcje wybrania pivotu: losowy, nielosowy - pierwszy element tablicy i wyznaczony algorytmem magicznych piątek. W tej części rozpatrzę jedynie opcję losową i nielosową.

Ponieważ szukamy k-tego elementu, możemy wprowadzić modyfikację algorytmu Quick Sort na Quick Select, co pozwala na zmniejszenie rozmiaru sortowanej listy. Tak jak w Quick Sort, robimy podział na część lewą (mniejszą od pivotu), środkową (równą) i prawą (większą). W zależności od wielkości k i dlugości kolejnych list możemy kontynuować szukanie w części lewej, prawej lub zakończyć wiedząc, że k-ty element znajduje sie w części środkowej. W implementacji tego algorytmu istotnym elementem było dodanie warunku, który zabezpieczał przed "wpadnięciem"w pustą listę.

2.3 Algorytm deterministyczny - magiczne piątki

Algorytm magicznych piątek wykorzystałam w szybkim sortowaniu, co dało teoretyczną pewność wybrania optymalnego pivotu do podziału listy.

Pierwszym krokiem jest stworzenie tablicy median podzbiorów długości 5. Mediany tych malutkich podzbiorów znajdowane były algorytmem InsertionSort jako, że dobrze sprawdza się dla bardzo małych list. Następnie dla tablicy median ponownie wywołujemy algorytm magicznych piątek (działamy rekurencyjnie).

2.4 Algorytm probabilistyczny oparty na próbkowaniu

Główną ideą zaimplementowanego algorytmu jest znalezienie k-tego elementu w $2n$ porównaniach. Działa on z prawdopodobieństwem $1 - \frac{1}{n}$, czyli z bardzo dużym prawdopodobieństwem.
Pierwszym krokiem jest wylosowanie 1000 próbek ze zwracaniem, a następnie posortowanie tej tablicy, do czego wykorzystałam algorytm Quick Sort. Ponadto dodałam warunek, że jeśli tablica ma mniej niż 1000 elementów, nie wykonujemy algorytmu probabilistycznego, tylko zwykłe sortowanie Quick Sort, ponieważ przy losowaniu 1000 próbek nie miałoby to sensu.

Następnie znajdujemy proporcjonalny odpowiednik k w tablicy próbek i odchylamy się od niego na lewo i na prawo o 40 elementów. Skrajny lewy i skrajny prawy element oznaczamy odpowiedni jako: l_1 oraz l_2.

Następnie wybieramy elementy listy, w której poszukujemy k-tego elementu takie, że należą do przedziału $[l_1, l_2]$. Od razu też aktualizujemy parametr. Musi być on pomniejszony o liczbę elementów głównej listy mniejszych od l_1. Jednak na tym etapie mogą pojawić się następujące problemy:

- $l_1 > l_2$ - wtedy po prostu wywołujemy ponownie nasz algorytm probabilistyczny
- element k-ty należy od przedziału $\leq l_1$ (objawia się to tym, że uaktualniony parametr k jest ujemny) - wtedy nowym parametrem l_1 staje się najemnieszy element tablicy próbek, a jeśli to nie pomoże, z głównej tablicy wybieramy te elementy, które są mniejsze l_1
- element k-ty należy do przedziału $\geq l_2$ (objawia się to tym, że długość nowej tablicy jest mniejsza od auktualnionego k. Postępujemy w zasadzie analogicznie do poprzedniego przypadku

2.5 Algorytmy zaproponowane przez AI

2.5.1 Introspective Quickselect

Sztuczna inteligencja (chat gpt4o) podaje następujące wytłumaczenie algorytmu:

"Introspective QuickSelect to hybrydowy algorytm znajdowania k-tego najmniejszego elementu, który dynamicznie przełącza się między strategiami w zależności od głębokości rekurencji. Rozpoczyna działanie jak klasyczny QuickSelect, wybierając pivot i dzieląc tablicę, ale monitoruje liczbę wywołań rekurencyjnych. Jeśli głębokość przekroczy ustalony próg, zamiast losowego pivota stosuje Median of Medians (BFPRT), gwarantując stabilne $O(n)$. W przypadku dalszych problemów przełącza się na HeapSelect, działający w $O(n + k \log k)$, co zapobiega najgorszym przypadkom $O(n^2)$. Dzięki temu algorytm dostosowuje się do struktury danych, zapewniając optymalną wydajność i stabilność."

2.5.2 Monte Carlo Quickselect

Sztuczna inteligencja (chat gpt4o) podaje następujące wytłumaczenie algorytmu:

"Monte Carlo QuickSelect to probabilistyczna wersja QuickSelect, która wykorzystuje losowe próbkowanie i dynamiczny wybór pivota do znajdowania k-tego najmniejszego elementu. Algorytm losuje próbkę z tablicy, sortuje ją i wybiera pivot na podstawie percentyla odpowiadającego pozycji k-tego elementu, a następnie dzieli tablicę na trzy części. Jeśli k leży w lewej lub prawej części, rekurencyjnie zawęża wyszukiwanie, a jeśli pivot źle podzielił tablicę, może powtórzyć próbkowanie. Dzięki temu unika deterministycznych wzorców, osiągając średnią złożoność$O(n)$ i działając szybciej niż klasyczne metody selekcji."

2.5.3 Hoare Partition

Sztuczna inteligencja (chat gpt4o) podaje następujące wytłumaczenie algorytmu:

"Hoare Partition to efektywna metoda podziału tablicy stosowana w algorytmie QuickSort i QuickSelect, pozwalająca na stabilne dzielenie zbioru względem pivota w czasie $O(n)$. Algorytm wybiera pivot i umieszcza go na końcu, a następnie utrzymuje dwa wskaźniki: jeden przesuwa się w prawo, szukając wartości większych od pivota, a drugi w lewo, szukając wartości mniejszych. Gdy wskaźniki się spotkają lub miną, pivot zostaje umieszczony na swojej docelowej pozycji, a zwrócony indeks określa jego nowe miejsce. Dzięki minimalnej liczbie operacji zamiany Hoare Partition działa szybciej niż metoda Lomuto i zapewnia lepszą wydajność na dużych zbiorach danych."

2.5.4 Ultra Fast Kth

Sztuczna inteligencja (chat gpt4o) podaje następujące wytłumaczenie algorytmu:

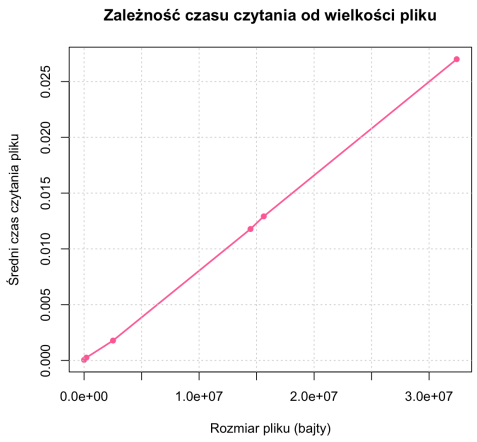
"Ultra Fast K-th to zaawansowany algorytm znajdowania k-tego najmniejszego elementu, łączący szybkie probabilistyczne metody z deterministycznymi strategiami. Wykorzystuje Median of Medians (BFPRT) do stabilnego wyboru pivota oraz Hoare Partition dla efektywnego podziału tablicy w czasie $O(n)$. Jeśli rekurencja staje się zbyt głęboka, algorytm adaptacyjnie przełącza się na HeapSelect, co zapewnia odporność na najgorsze przypadki$O(n^2)$. Dzięki dynamicznemu dostosowywaniu strategii osiąga optymalną wydajność, eliminując problemy klasycznego QuickSelect."

3 Testy

3.1 Wczytanie plików

Jak można się spodziewać, wczytanie plików to zależność prawie liniowa.

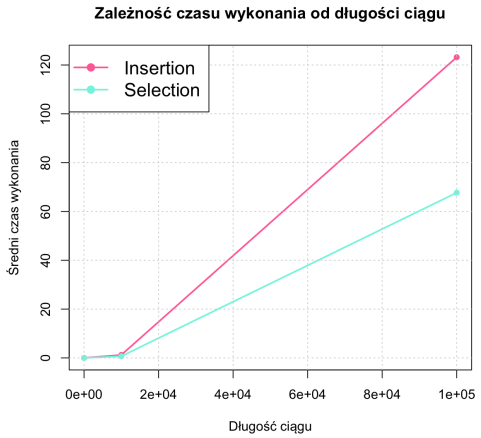
plik	średni czas wykonania (s)
plik 1	0.0000463000
plik 2	0.0002407333
plik 3	0.0023271667
plik 4	0.0125021667
plik 5	0.0169384667
plik 6	0.0286827333



3.2 Insertion i Selection Sort

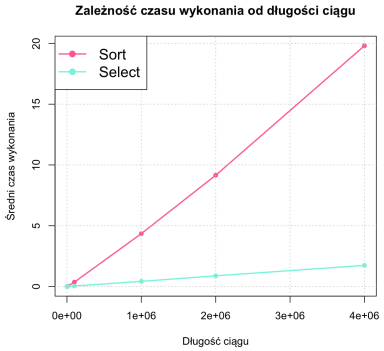
Po krótkiej analizie wykresu widzimy, że dzięki możliwości szybszego zakończenia obliczeń, lepiej od Insertion Sort wypada Selection. Uzależnienie czasu obliczeń od wielkości k względem długości listy doskonale widać było przy wywoływaniu algorytmu Selection Sort dla pliku 3 (tabelka). Jeszcze jedną ciekawą zależnością jaką zauważyłam przy Insertion Sort (przez niepoprawne napisanie pętli testującej) było to, że główny wpływ na czas obliczeń algorytmu Insertion Sort ma ilość przestawień, jakie musimy wykonać.

k	średni czas wykonania (s)
22222	36.182577
44444	62.937324
66666	80.005119
88888	90.411745

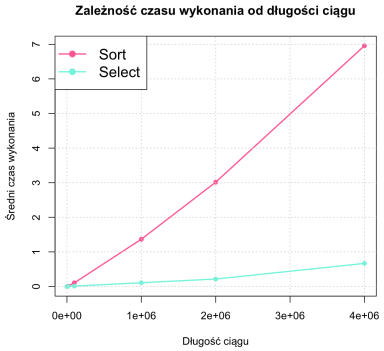


3.3 Quick Sort i Quick Select

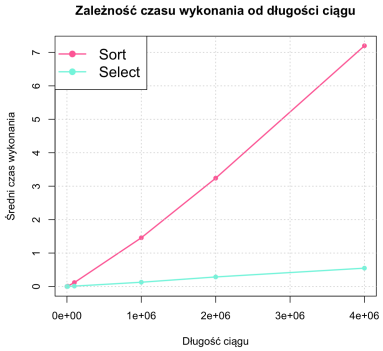
Analiza czasów wykonania algorytmów Quick Sort i Quick Select potwierdziła "roboczą hipotezę", że poszukiwanie w tylko tych fragmentach tablicy, w których wiemy, że znajduje sie k-ty element da lepsze wyniki niż sortowanie całej tablicy.



Rysunek 1: pivot: Magic5



Rysunek 2: pivot: arr[0]



Rysunek 3: pivot: random

plik	Quick Sort	Quick Select
plik 1	0.0000133	0.000007
plik 2	0.0288925	0.004081
plik 3	0.3588878	0.040255
plik 4	4.3480231	0.423198
plik 5	9.1512754	0.878189
plik 6	19.803616	1.734991

Tabela 1: pivot: Magic5

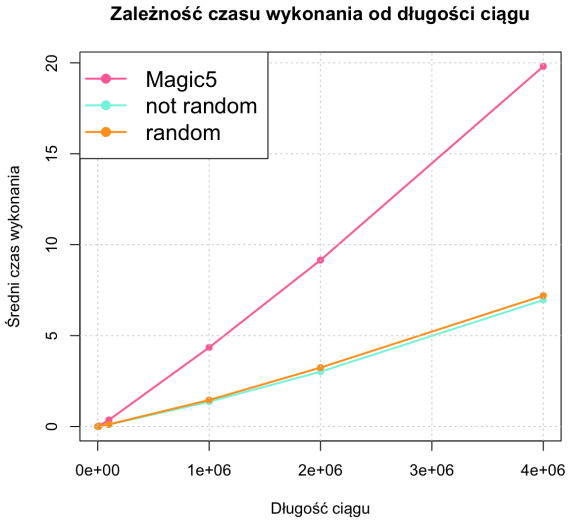
plik	Quick Sort	Quick Select
plik 1	0.00001	0.0000036
plik 2	0.00878	0.0009032
plik 3	0.10635	0.0113022
plik 4	1.36617	0.1087371
plik 5	3.01435	0.2156279
plik 6	6.95719	0.6672298

Tabela 2: pivot: arr[0]

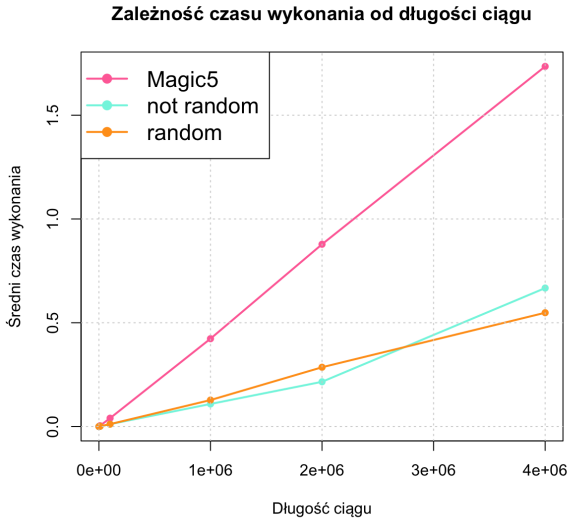
plik	Quick Sort	Quick Select
plik 1	0.0000072	0.0000056
plik 2	0.0106425	0.0010707
plik 3	0.1169365	0.0115854
plik 4	1.4572353	0.1275098
plik 5	3.2398908	0.2856519
plik 6	7.1989285	0.5483731

Tabela 3: pivot: random

Co ciekawe, najgorzej w naszym przypadku wypadł algorytm Magicznych piątek mimo, że powinien przynosić teoretyczną poprawę. Porównując linię wykresu z pivotem random i pivotem wybieranym z początku tablicy można przyjąć, że to przez "dobre posortwanie"listy, czyli takie, które sprawia, że nie wpadamy z pivotem w przypadki pesymistyczne.Kolejną obserwacją jaką można poczynić jest to, że oba szybkie algorytmu mają "najbardziej liniowy"wykres przy korzystaniu z algorytmu magicznych piątek.



Rysunek 4: Quick Sort



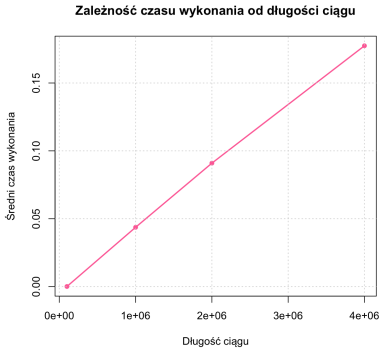
Rysunek 5: Quick Select

3.4 Algorytm probabilistyczny

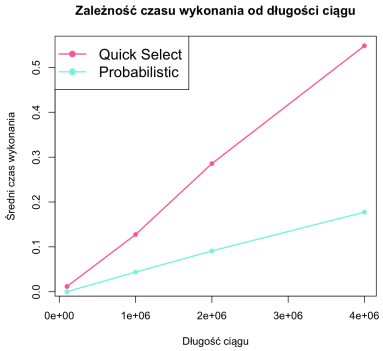
Wykres potwierdza nasze przypuszczenia teoretyczne, że algorytm będzie działał w czasie liniowym $O(2n)$. Warto też zauważyć przewagę tego algorytmu nawet nad najszybszym algorytmem sortowania szybkiego - Quick Select z losowym pivotem.

plik	Probabilistyczny
plik 3	0.0000344
plik 4	0.0436559
plik 5	0.0909212
plik 6	0.1774288

Tabela 4: Czasy algorytmu probabilistycznego



Rysunek 6: Wykres czasów algorytmu probabilistycznego



Rysunek 7: Porównanie probabilsitycznego i Quick Select

3.5 Algorytmy zaproponowane przez sztuczną inteligencję

Niestety żaden z algorytmów zaproponowanych przez AI (Chat GPT 4o) nie wypadł lepiej od zaimplementowanego algorytmu probabilistycznego.

plik	AI1	AI2	AI3	AI4
plik 1	0.0000266	0.0000151	0.0000067	0.0001231
plik 2	0.0052338	0.0008005	0.0053403	0.0017618
plik 3	0.0541578	0.0065483	0.0548647	0.0177969
plik 4	0.6037985	0.0659172	0.6013792	0.1736753
plik 5	1.3446393	0.1695191	1.3795149	0.3597991
plik 6	2.6512344	0.2586613	2.6916162	0.8278988