Algorytmy sortowania – sprawozdanie

Selection sort

Najlepszy przypadek: O()

Średni przypadek: O()

Najgorszy przypadek: O()

Jak widać na wykresie powyżej, Selection sort ma taką samą złożoność obliczeniową niezależnie od rodzaju danych. Wynika to z zasady działania algorytmu. W sortowaniu przez wybieranie wyszukujemy n razy najmniejszy element z pozostałych i przesuwamy go na odpowiednie miejsce. Nie da się znaleźć minimum inaczej niż liniowo (musimy sprawdzić każdy element tablicy), dlatego cały algorytm (uwzględniając, że n razy szukamy rzeczonego minimum) ma złożoność kwadratową. Nie istnieją lepsze czy gorsze przypadki dla tego algorytmu.

Insertion sort

Najlepszy przypadek: O()

Średni przypadek: O()

Najgorszy przypadek: O()

W przypadku Insertion sorta widać wyraźnie, że jego czas działania zależy od podanych danych wejściowych. Warto zastanowić się, dlaczego tak się dzieje. Algorytm działa najgorzej, gdy liczby w tablicy są ułożone malejąco. Nie jest to zaskoczeniem, gdy przeanalizuje się zasadę działania sortowania przez wstawianie. W algorytmie tym zakładamy najpierw, że pierwszy element jest posortowany. Następnie po kolei staramy się dołożyć kolejne liczby do ciągu posortowanego, zwiększając jego długość. W przypadku danych ułożonych malejąco właśnie owo dokładanie jest najbardziej problematyczne, bo zawsze musimy „cofnąć się” do początku tablicy. Odwrotnie jest w przypadku danych rosnących lub stałych, ponieważ, zwiększając długość ciągu liczb posortowanych, dokładamy kolejny element na sam koniec, co nie powoduje zwiększania czasu działania programu. Algorytm zachowuje się średnio dla danych losowych lub A-kształtnych, bo szukanie odpowiedniego miejsca dla dokładanej liczby jest zazwyczaj lepsze niż w przypadku malejących, a gorsze niż stałych, czy rosnących.

Shell sort

Najlepszy przypadek: O(, gdzie m to liczba przebiegów

Średni przypadek: O

Najgorszy przypadek: O(

Shell sort jest tak naprawdę udoskonaleniem algorytmu sortowania przez wstawianie. Wykorzystuje fakt, że Insertion sort zachowuje się bardzo dobrze dla danych rosnących. Cała idea polega na wstępnym posortowaniu danych w pewnych odstępach. Powoduje to wydłużenie czasu działania dla ciągów stałych lub rosnących, bo kilkukrotnie wykonujemy wstawianie tych liczb w coraz to dłuższe ciągi (zmniejszając odstępy), ale znacznie przyspiesza działanie algorytmu dla innych danych wejściowych. Złożoność obliczeniowa Shell sorta jest bardzo trudna do określenia, bo zależy nie tylko od danych, ale przede wszystkim od wybranych odstępów, które decydują, jakie zbiory liczb będziemy najpierw sortować. Odpowiednio dobrane odstępy do odpowiednich danych mogą bardzo usprawnić działanie programu. Algorytm najgorzej zachowa się dla ciągów losowych, bo najtrudniej jest przewidzieć, jakie odstępy byłby najlepsze.

Heap sort

Najlepszy przypadek: O(

Średni przypadek: O()

Najgorszy przypadek: O()

Heap sort jest algorytmem sortowania zaliczanym do klasy złożoności . By przejść do sortowania, najpierw musimy utworzyć kopiec, co można zrobić liniowo. Samo sortowanie już jest bardziej złożone obliczeniowo, ponieważ n razy (dla każdego elementu) musimy odtworzyć własność kopca w czasie . Widać, że algorytm działa najlepiej dla ciągów stałych, a dzieje się tak z bardzo prostej przyczyny. W takim przypadku zbędne jest odtwarzanie własności kopca, ponieważ niezależnie od ułożenia tych elementów i przestawiania jednego do wierzchołka, zawsze będą one zachowane.

Dane wejściowe - ciąg losowy

Wykres ilustruje czas działania poszczególnych algorytmów dla losowych danych wejściowych. Najgorzej działa Selection sort, ponieważ za każdym razem wyszukuje minimum w ciągu nieposortowanych jeszcze liczb, działając kwadratowo. Insertion sort też jest algorytmem o złożoności kwadratowej, lecz zachowuje się średnio lepiej od sortowania przez wybieranie. Wynika to z faktu, że dla danych losowych nie zawsze musi „cofnąć się” aż do początku tablicy, by znaleźć odpowiednie miejsce dla wstawianego elementu. Shell sort jest ulepszeniem sortowania przez wybieranie, co łatwo można zaobserwować na wykresie. Bardzo wydajnym algorytmem jest Heap sort, który praktycznie zawsze (oprócz ciągów stałych) ma złożoność liniowo-logarytmiczną.

Dane wejściowe – ciąg stały

Z wykresu wyraźnie widać, że prawie wszystkie algorytmy, z wyjątkiem Selection sorta, bardzo dobrze radzą sobie z ciągiem stałym jako ciągiem wejściowym. Jak już wcześniej było napisane, Selection sort zwyczajnie nie ma lepszych, czy gorszych przypadków, zawsze działa tak samo. Najlepiej radzi sobie Insertion sort, który w takim przypadku tylko sprawdza, czy elementy są na swoim miejscu, nie musząc nic zmieniać. Nie zmienia też nic Shell sort, ulepszenie Insertion sorta, ale działa troszeczkę gorzej w tym przypadku, ponieważ wykonuje więcej porównań dla różnych odstępów. Ciąg stały jest też najlepszym przypadkiem dla Heap sorta, bo nie musi być odtwarzana własność kopca. Sortowanie przez kopcowanie nie jest algorytmem stabilnym, więc dojdzie do zamiany elementów miejscami.

Dane wejściowe – ciąg rosnący

W takim przypadku z wyjaśnionych przyczyn najgorzej działa Selection sort. Ciąg rosnący, podobnie do stałego, jest również najlepszym przypadkiem dla Insertion sorta, który nie musi nic zmieniać. Shell sort zadziała minimalnie gorzej, bo wykona więcej porównań, wywołując sortowanie przez wstawianie kilkukrotnie. Heap sort, będąc bardzo wydajnym liniowo-logarytmicznym algorytmem, mimo że nie jest to jego najlepszy przypadek, zadziała nieznacznie gorzej niż Insertion sort, czy Shell sort.

Dane wejściowe – ciąg malejący

Ciąg malejący jest najgorszym przypadkiem dla Insertion sorta, co widać wyraźnie. W tym przypadku algorytm ten zadziałał nawet gorzej niż zawsze działający tak samo Selection sort. Z wykresu wyraźnie widać, że Shell sort jest prawdziwym ulepszeniem Insertion sorta. Dzieląc tablicę na bardzo małe zbiory danych i wstępnie sortując, bardzo ograniczył czas działania. Nie musiał dla każdego elementu „cofać się” aż to początku przez dużą ilość elementów. Heap sort działa w takim przypadku jak zawsze dobrze.

Dane wejściowe – ciąg A-kształtny

Dla danych A-kształtnych Selection sort działa tak samo źle, jak w pozostałych przypadkach. Ciąg A-kształtny jednak nie jest też zbyt korzystny dla Insertion sorta. Choć pierwsza połowa tablicy jest posortowana i nie wymaga dokonania zmian przez algorytm, to druga połowa jest malejąca i wstawianie elementów z tego fragmentu do posortowanego ciągu z pierwszej połówki wymaga wracania prawie że do początku, co nie jest zadowalające czasowo. Shell sort radzi sobie znacznie lepiej przez wstępne posortowanie tablicy w odstępach. Zaletą Heap sorta jest jego złożoność praktycznie niezależąca od danych wejściowych.