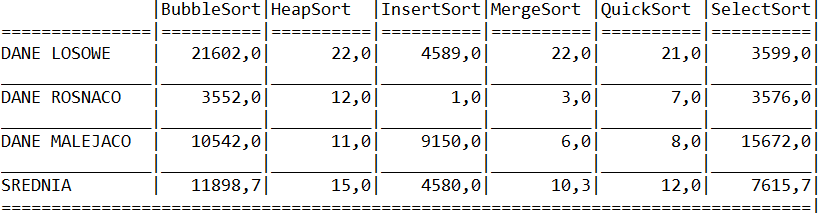
Raport

Sortowanie różnymi sposobami

*Uladzislau Lapko, 245864*

**Dane:**

* 100.000 elementów;
* 3 ciągi (losowe liczbe, posortowane rosnąco i posrtowane malejąco);

**Tabela czasu:  
Wnioski:**

Robiłem wszystkie tablicy za pomocą metody Random i Random.nextInt().

W Danych Rosnących są wyniki najlepsze dlatego że mamy już posortowane tablicy i algortmy muszą tylko sprawdzić to, czyli zrobić tylko jedno podejście. W Malejących są średnie wyniki, bo mamy posortowane, ale odwrotnie. Losowe wyniki są najgorsze.

Można zobaczyć, że najgorszym algorytmem jest *BubbleSort*, ale on jest najłatwiejszy do zaimplementowania. Potem idzie *SelectionSort*, za nim *InsertSort*. Najlepszymy są *Heap, Merge* i *Quick*, ale one są konstruwowane za pomocą rekurencji.

Sortowanie bąbelkowe (Bubble sort)

Nazwa pochodzi od obrazowego przedstawienia zasady działania — w każdym, przebiegu’ kolejna wartość w kolejności ‚wypływa’ jak bąbelek powietrza na powierzchnie wody.

W algorytmie tym w celu posortowania n elementów wykonujemy n-1 przebiegów. W każdym przebiegu zaczynamy od początku zbioru, bierzemy dwa elementy i zamieniamy je miejscami tak, aby większy był *‚wyżej’* (lub z większym indeksem itp). W pierwszym kroku robimy to dla elementów 1 i 2, w drugim kroku dla 2 i 3 itd. Każdy przebieg *‚wyciąga’* do góry kolejny element w kolejności, po wykonaniu wszystkich przebiegów nasz zbiór jest posortowany.

Sortowanie szybkie (Quicksort)

Algorytm ten jest najpopularniejszy, jeśli chodzi o ilość zastosowań, ponieważ w ogólnym i uśrednionym przypadku jest on najbardziej optymalny, a jednocześnie prosty w implementacji. Jego wadą jest negatywny przypadek, kiedy to złożoność wynosi n^2 .

Algorytm jest rekursywny, tj. odwołuje się do samego siebie i jest przykładem podejścia dziel i zwyciężaj (divide and conquer). W każdym kroku ze zbioru elementów wybieramy dowolny element (nazywany pivot; może to być po prostu pierwszy z brzegu lub losowy, dla nieuporządkowanych danych nie ma to znaczenia) a następnie dzielimy pozostałe elementy na dwie grupy — te mniejsze od wybranego elementu (‚przed nim’) oraz większe od niego (‚za nim’). Każdą z tych grup sortujemy w ten sam sposób.

Sortowanie przez scalanie (Merge sort)

Jest to kolejny przykład podejścia dziel i rządź — w tym wypadku polega on na podziale zbioru na *n* równych podzbiorów (najczęściej jednoelementowych) a następnie scalaniu ich ze sobą uzyskując większe, posortowane podzbiory. Najprostsza implementacja jest rekurencyjna, ale nie wynika to bezpośrednio z samego algorytmu — scalanie może się odbywać iteracyjnie, dołączając kolejne podzbiory do jednego ‚posortowanego’. Dużą zaletą tego algorytmu jest możliwość zrównoleglenia — operacje sortowania można wykonywać na kilku wątkach lub nawet na wielu różnych maszynach w tym samym czasie.

Sortowanie kopcowe (heap sort; funkcjonuje także nazwa sortowanie przez kopcowanie)

Sortowanie przez kopcowanie (częściej spotykaną nazwą, także w polskiej literaturze, jest heap sort) to kolejny z algorytmów mający praktyczne zastosowanie — jego wydajność jest najczęściej minimalnie mniejsza niż QuickSort, ale wydajność pesymistyczna jest zdecydowanie lepsza. Polega on na budowaniu kopca — czyli struktury, w której elementy są posortowane w określonym porządku — a następnie iteracyjnym dodawaniu kolejnych elementów zbioru. Po każdym dodaniu konieczne jest przywrócenie właściwości kopca, tzn. uporządkowanie elementów.

Sortowanie przez wstawianie (insertion sort)

Sortowanie przez wstawianie (insertion sort) jest naturalnym algorytmem sortowania. Wielu ludzi stosuje go często, nawet o tym nie wiedząc, chociażby przy układaniu kart w ręce. Sortowanie to jest stabilne, *in situ*, o złożoności średniej O(n^2). Jest uznawane za najszybsze z sortowań prostych, dodatkowo jest bardzo intuicyjne i nieskomplikowane w implementacji. Nadaj się również do sortowania list. Czasem używa się go w [sortowaniu szybkim](https://4programmers.net/Algorytmy/Sortowanie/Quick_Sort) do sortowania odpowiednio małych przedziałów, co korzystnie wpływa na czas tej operacji.

1. Utwórz zbiór elementów posortowanych i przenieś do niego dowolny element ze zbioru nieposortowanego.
2. Weź dowolny element ze zbioru nieposortowanego.
3. Wyciągnięty element porównuj z kolejnymi elementami zbioru posortowanego, póki nie napotkasz elementu równego lub elementu większego (jeśli chcemy otrzymać ciąg niemalejący) lub nie znajdziemy się na początku/końcu zbioru uporządkowanego.
4. Wyciągnięty element wstaw w miejsce, gdzie skończyłeś porównywać.
5. Jeśli zbiór elementów nieuporządkowanych jest niepusty, wróć do punktu 2.

Sortowanie przez wybieranie  (selection sort)

Sortowanie przez wybieranie - jedna z prostszych metod [sortowania](https://pl.wikipedia.org/wiki/Sortowanie) o [złożoności](https://pl.wikipedia.org/wiki/Z%C5%82o%C5%BCono%C5%9B%C4%87_obliczeniowa) O(n2). Polega na wyszukaniu elementu mającego się znaleźć na żądanej pozycji i zamianie miejscami z tym, który jest tam obecnie. Operacja jest wykonywana dla wszystkich indeksów sortowanej tablicy.

Algorytm przedstawia się następująco:

1. wyszukaj minimalną wartość z tablicy spośród elementów od i do końca tablicy.
2. zamień wartość minimalną, z elementem na pozycji *i.*

Gdy zamiast wartości minimalnej wybierana będzie maksymalna, wówczas tablica będzie posortowana od największego do najmniejszego elementu.