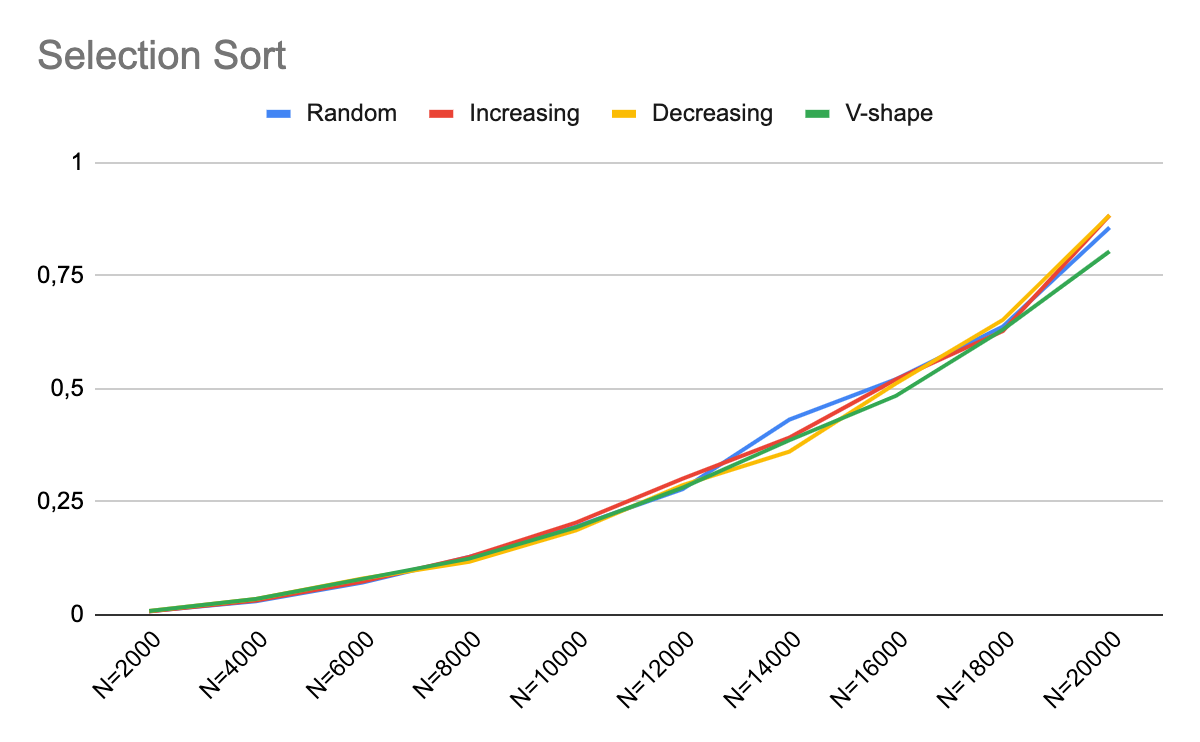
Katarzyna Zawrotniak

Kamil Albrycht

**Selection Sort**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **N=2000** | **N=4000** | **N=6000** | **N=8000** | **N=10000** | **N=12000** | **N=14000** | **N=16000** | **N=18000** | **N=20000** |
| **Random** | 0,00823 | 0,03022 | 0,07174 | 0,12827 | 0,19543 | 0,27830 | 0,43197 | 0,52122 | 0,63759 | 0,85661 |
| **Increasing** | 0,00760 | 0,03223 | 0,07432 | 0,12858 | 0,20435 | 0,30174 | 0,39220 | 0,52080 | 0,62771 | 0,88322 |
| **Decreasing** | 0,00895 | 0,03551 | 0,08084 | 0,11782 | 0,18730 | 0,28736 | 0,36093 | 0,51195 | 0,65212 | 0,88389 |
| **V-shape** | 0,00890 | 0,03560 | 0,07991 | 0,12469 | 0,19376 | 0,28137 | 0,38638 | 0,48473 | 0,63065 | 0,80362 |

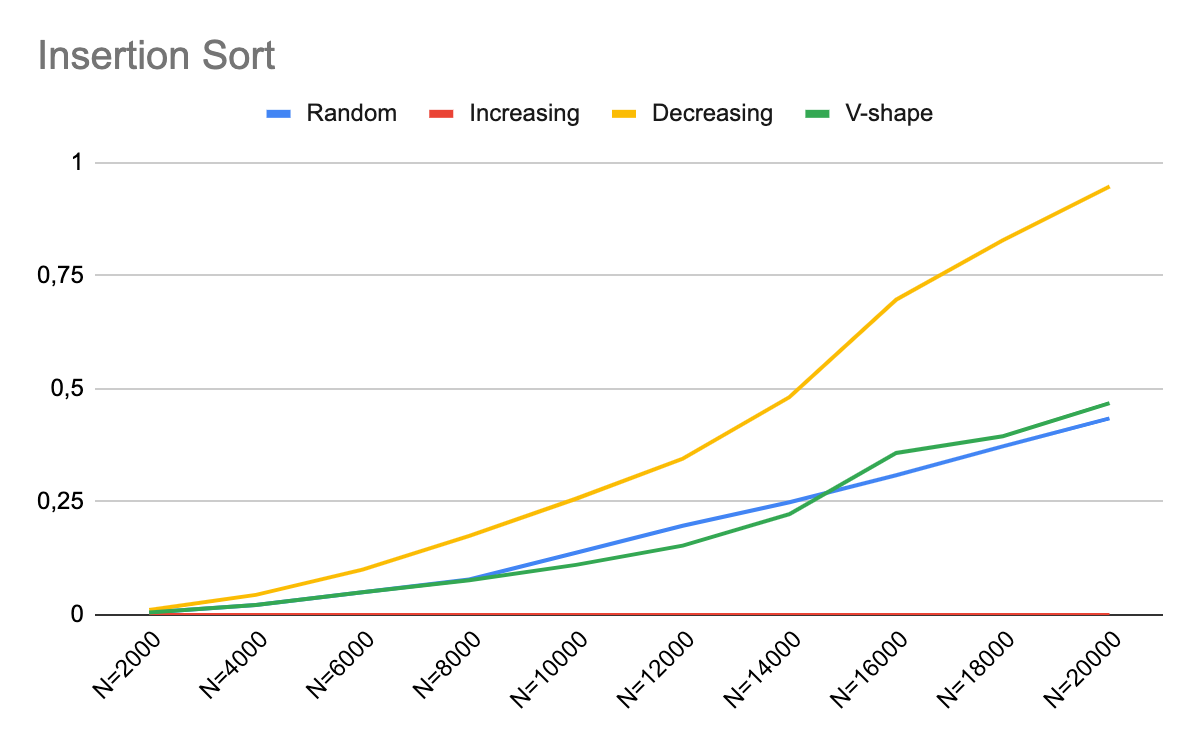


Niezależnie od typu wprowadzonych danych algorytm Selection Sort zachowuje się bardzo podobnie. Wraz ze wzrostem ilości liczb wzrasta czas wykonania tej operacji. Przy ilości liczb w wysokości 20 tys. najszybciej wynik uzyskano w przypadku rozkładu V-kształtnego, natomiast najwolniej przypadku wartości malejących.

Cechuje się stałą złożonością obliczeniową wynoszącą O(*n*2), niezależnie od wielkości danych wejściowych. Sprawia to, że algorytm ten nie jest wydajny nawet dla małej ilości danych.

**Insertion Sort**

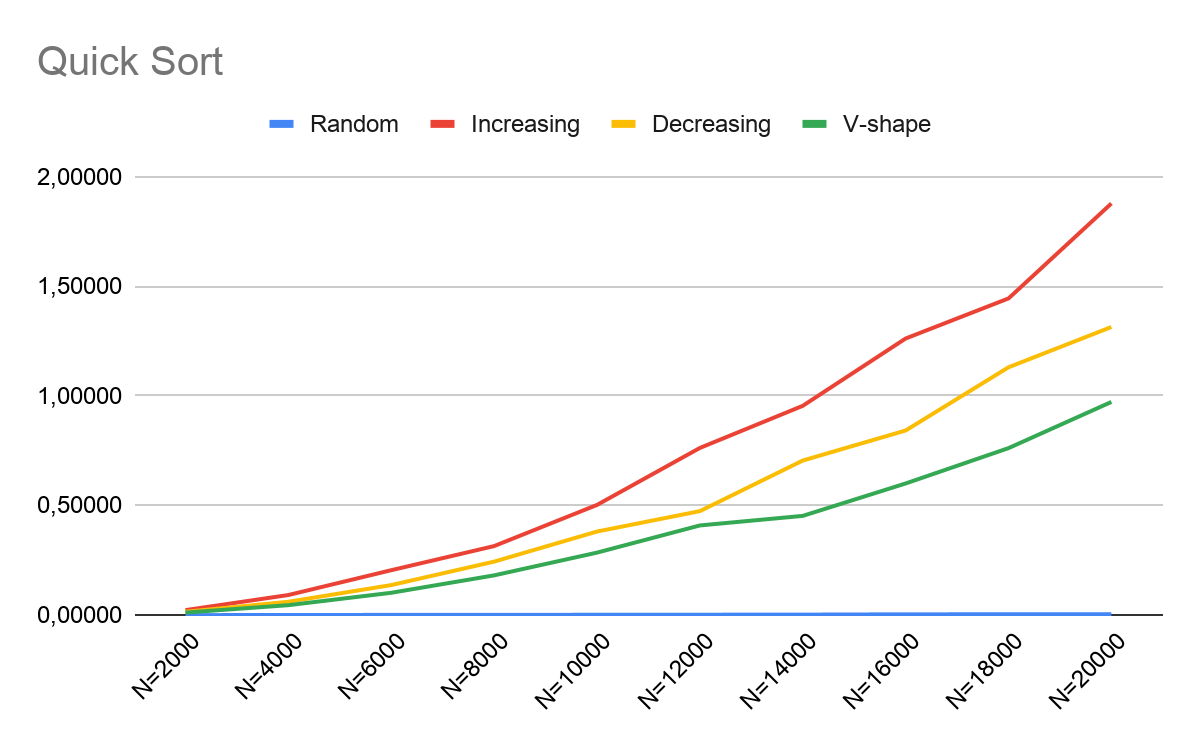
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **N=2000** | **N=4000** | **N=6000** | **N=8000** | **N=10000** | **N=12000** | **N=14000** | **N=16000** | **N=18000** | **N=20000** |
| **Random** | 0,00576 | 0,02229 | 0,05058 | 0,07860 | 0,13771 | 0,19732 | 0,24942 | 0,30893 | 0,37270 | 0,43472 |
| **Increasing** | 0,00002 | 0,00004 | 0,00005 | 0,00007 | 0,00009 | 0,00010 | 0,00008 | 0,00013 | 0,00014 | 0,00015 |
| **Decreasing** | 0,01119 | 0,04455 | 0,10026 | 0,17507 | 0,25705 | 0,34543 | 0,48158 | 0,69690 | 0,82852 | 0,94716 |
| **V-shape** | 0,00555 | 0,02210 | 0,05017 | 0,07669 | 0,11093 | 0,15340 | 0,22273 | 0,35828 | 0,39501 | 0,46841 |



W przypadku algorytmu Insertion Sort zauważyć można znaczne różnice w czasie jego wykonania pomiędzy typami wprowadzonych danych, lecz niezależnie od tego czynnika jest on najbardziej wydajny dla małej ilości danych. Zdecydowanie najszybciej zostały posortowane wartości rosnące ( nie przekroczono nawet 0,25 sek gdyż gdy dane były wstępnie posortowane przez co algorytm przyjął złożoność obliczeniową O(*n*) przez co zmniejszyła się liczba porównań). Najwolniej natomiast sortowaniu uległy wartości malejące ( algorytm przyjął złożoność obliczeniową O(*n*2), gdyż musiał wykonać dużą liczbę porównań wydłużających jego czas pracy co spowodowałoby, że w przypadku większej ilości liczb niż 20 tys. prawdopodobnie czas przekroczyłby 1 sek).Uogólniając, czas pracy Insertion Sort nie zależy głównie od wielkości wprowadzonych danych, natomiast od ich wstępnego posortowania.

**Quick Sort**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **N=2000** | **N=4000** | **N=6000** | **N=8000** | **N=10000** | **N=12000** | **N=14000** | **N=16000** | **N=18000** | **N=20000** |
| **Random** | 0,00040 | 0,00080 | 0,00123 | 0,00140 | 0,00177 | 0,00216 | 0,00259 | 0,00383 | 0,00415 | 0,00427 |
| **Increasing** | 0,02310 | 0,09191 | 0,20510 | 0,31514 | 0,50277 | 0,76266 | 0,95474 | 1,26175 | 1,44457 | 1,87762 |
| **Decreasing** | 0,01534 | 0,06119 | 0,13727 | 0,24422 | 0,38140 | 0,47449 | 0,70527 | 0,84221 | 1,13049 | 1,31424 |
| **V-shape** | 0,01186 | 0,04564 | 0,10214 | 0,18156 | 0,28528 | 0,40927 | 0,45284 | 0,60068 | 0,76183 | 0,97246 |

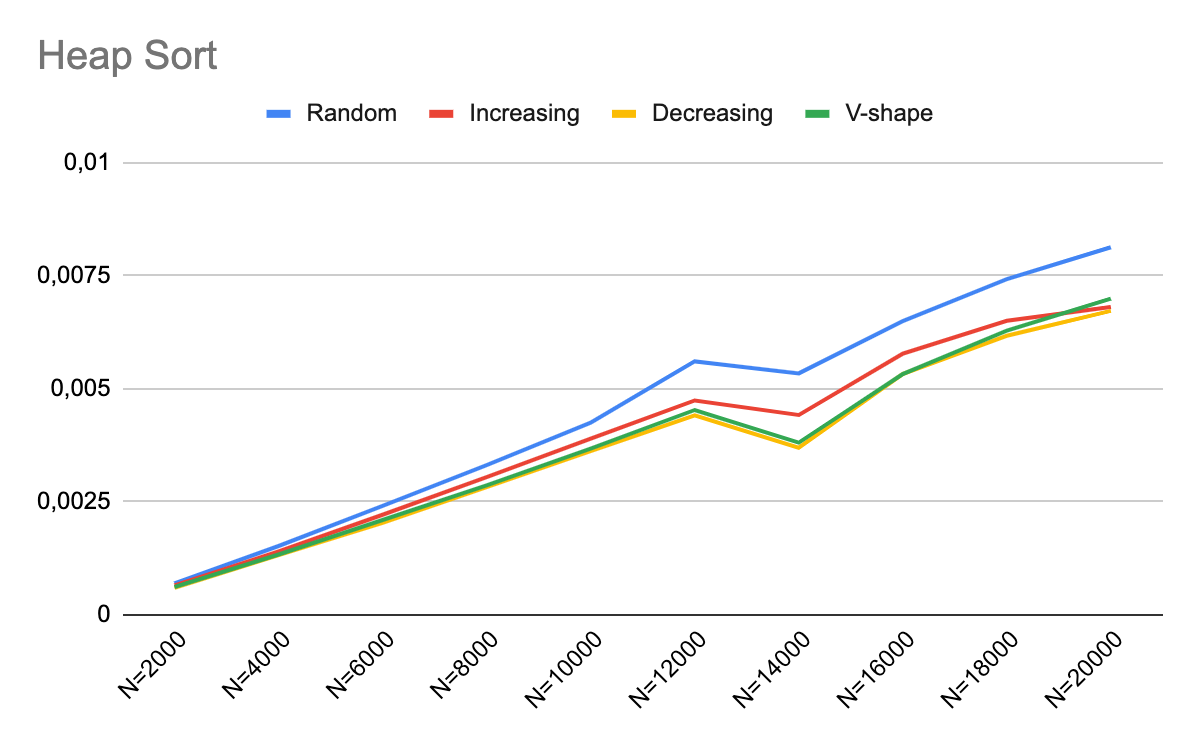


W przypadku algorytmu Quick Sort zauważyć można znaczne różnice w czasie jego wykonania pomiędzy typami wprowadzonych danych. Zdecydowanie najszybciej zostały posortowane wartości losowe ( nie przekroczono nawet 0,005 sek). Najwolniej natomiast sortowaniu uległy wartości rosnące (w przypadku większej ilości liczb niż 20 tys. prawdopodobnie czas przekroczyłby 2 sek).

Quick Sort cechuje się złożonością obliczeniową O(n log n), natomiast w najgorszym przypadku, wynosi O(*n*2). Algorytm ten jest wrażliwy na dużą ilość danych ze względu na to że do działania potrzebuje dużej ilości pamięci, gdyż wywołuje się rekurencyjnie.

**Heap Sort**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **N=2000** | **N=4000** | **N=6000** | **N=8000** | **N=10000** | **N=12000** | **N=14000** | **N=16000** | **N=18000** | **N=20000** |
| **Random** | 0,00070 | 0,00153 | 0,00241 | 0,00331 | 0,00425 | 0,00561 | 0,00534 | 0,00650 | 0,00743 | 0,00813 |
| **Increasing** | 0,00065 | 0,00141 | 0,00222 | 0,00305 | 0,00390 | 0,00474 | 0,00442 | 0,00578 | 0,00650 | 0,00681 |
| **Decreasing** | 0,00060 | 0,00132 | 0,00204 | 0,00283 | 0,00362 | 0,00441 | 0,00370 | 0,00533 | 0,00617 | 0,00672 |
| **V-shape** | 0,00062 | 0,00134 | 0,00210 | 0,00286 | 0,00368 | 0,00453 | 0,00381 | 0,00533 | 0,00629 | 0,00699 |



W algorytmie Heap Sort można zauważyć ciekawą zależność. Niezależnie od wprowadzonego typu danych czas wykonania operacji wzrasta aż do momentu gdy ilość liczb wynosi 1400 gdzie czas ten skraca się by dalej od 1600 liczb rosnąć jednostajnie. Najszybciej algorytm wykonuje się w przypadku wartości malejących, natomiast najwolniej w przypadku wartości losowych.

Heap Sort cechuje się złożonością obliczeniową O(n log n), nie jest też wrażliwy na to czy dane wejściowe są wstępnie posortowane czy też nie.