**Złożoność Obliczeniowa  
Algorytmów Sortowania**

Marcin Duda  
Maciej Frel

**Porównanie różnych algorytmów sortowania**

**Złożoność obliczeniowa:**

|  |  |  |  |
| --- | --- | --- | --- |
| Algorytm\Przypadek | Najgorszy | Sredni | Najlepszy |
| Insertion Sort | O(n2) | O(n2) | O(n) |
| Selection Sort | O(n2) | O(n2) | O(n2) |
| Heap Sort | O(nlog2n) | O(nlog2n) | O(nlog2n) |
| Quick Sort | O(n2) | O(nlog2n) | O(nlog2n) |

**Wnioski:**

Na efektywność algorytmów sortowania ma wpływ między innymi liczba sortowanych elementów, rozkład i zakres elementów czy złożoność obliczeniowa. Przedstawione algorytmy można podzielić na dwie grupy ze względu na złożoność obliczeniową: wolne - O(n2) i szybkie - O(n log2 n). Wszystkie algorytmy sortują w miejscu, to znaczy, że nie potrzebują dodatkowych struktur do sortowania, jednakże Heap Sort i Quick Sort są algorytmami rekurencyjnymi co oznacza, ze zostanie użyta dodatkowa pamięć ze stosu.

**Wpływ danych wejściowych:**

- Heap Sort i Selection Sort dane wejściowe nie zmieniają wielkości klasy złożoności obliczeniowej

- Quick Sort z pivotem w środku najgorzej sobie radzi z ciągami A i V kształtnymi jego złożoność rośnie do O(n2). Jeżeli możemy poświecić dodatkowy czas na wybranie pivotu to najlepszym punktem jest mediana, dzięki czemu procedura partition podzieli tablice na równe części co dam nam czas wykonania O(n log2 n).

- Insertion Sort jako jedyny z badanych algorytmów osiąga złożoność liniowa dla najlepszego przypadku – ciągu posortowanego rosnąco. Pesymistycznym przypadkiem jest ciąg posortowany malejąco. W średnim przypadku algorytm sprawdza około polowe tablicy aby znaleźć miejsce do wstawienia elementu.

**Porównanie różnych typów wyboru sposobu podziału w algorytmie Quick Sort w wersji iteracyjnej**

**Wnioski:**

Wybór pivotu ma znaczący wpływ na czas wykonywania się programu. Zły dobór klucza dla ciągu wejściowego może zwiększyć czas wykonywania z O(n log2 n) do O(n2). Klucz powinien być wybierany ze względu na dane wejściowe aby zmniejszyć szanse na przypadek pesymistyczny. Zużycie pamięci wynosi O(n) ponieważ został użyty stos o rozmiarze n.

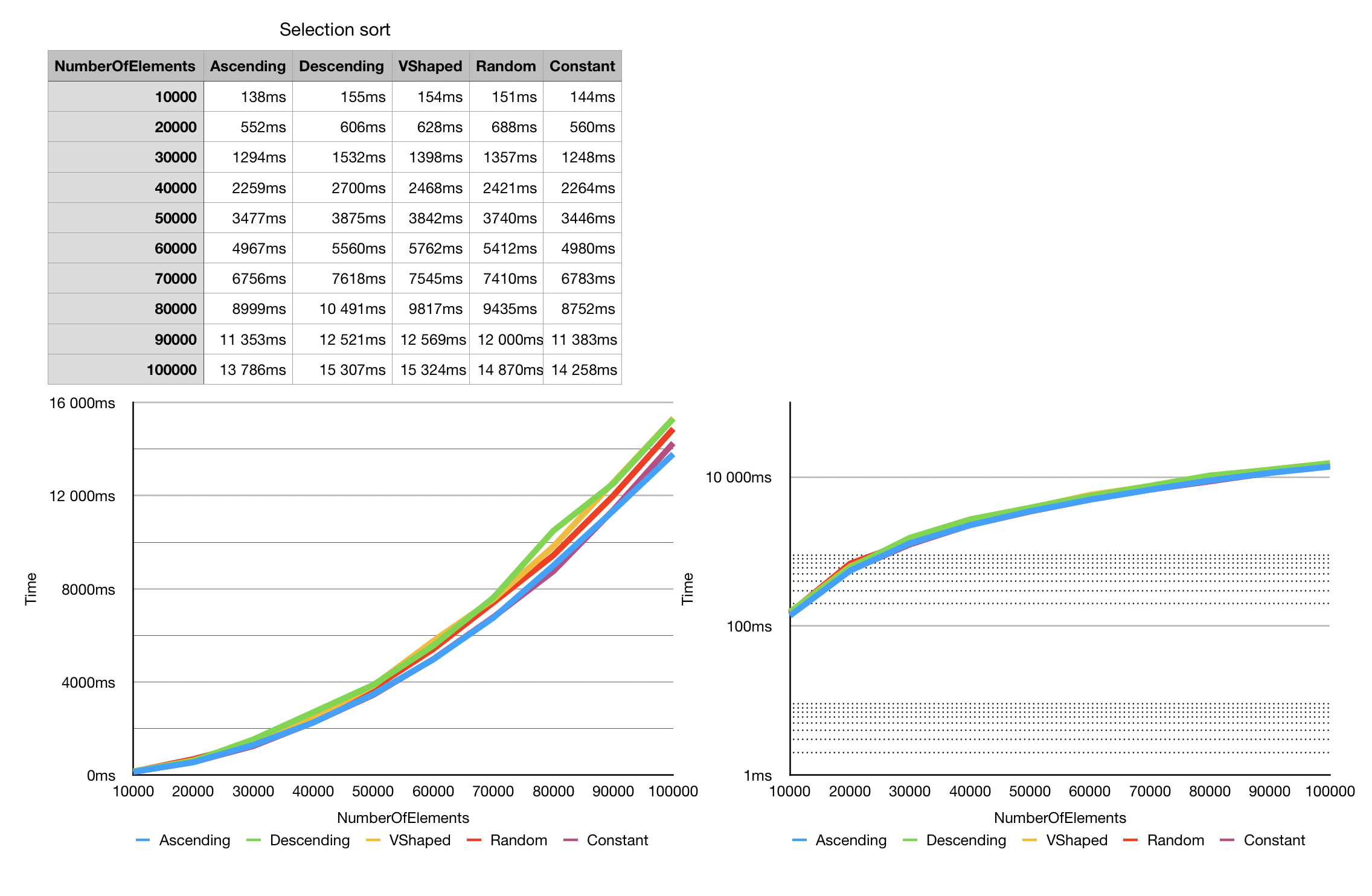
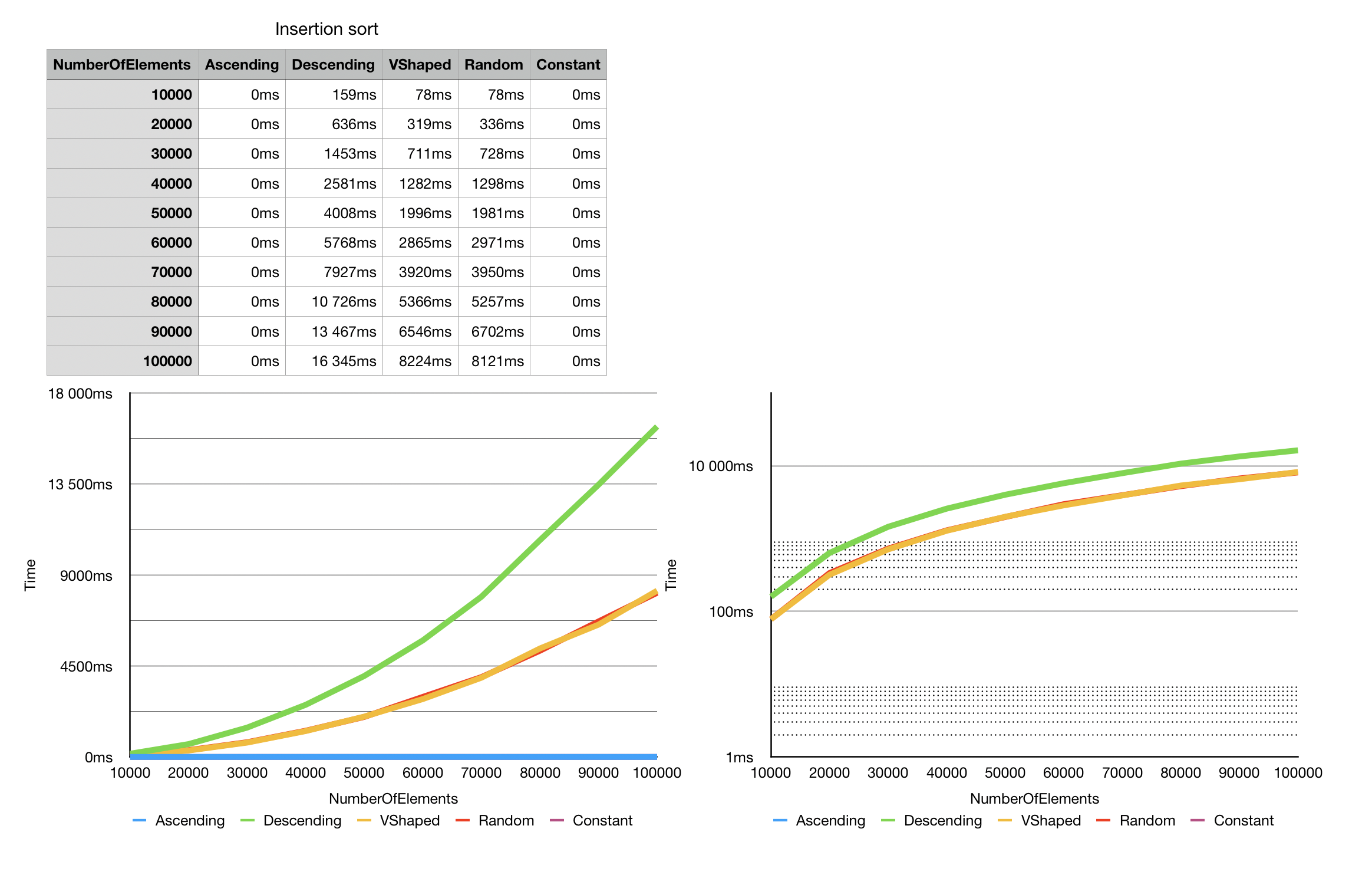
- losowy klucz - w tym przypadku nie można jednoznacznie powiązać czasu wykonania z wyborem pivotu i z danymi wejściowymi jako, ze klucz jest wybierany losowo. Najgorszym kluczem będzie najmniejszy lub największy element ciągu, a najlepszym mediana

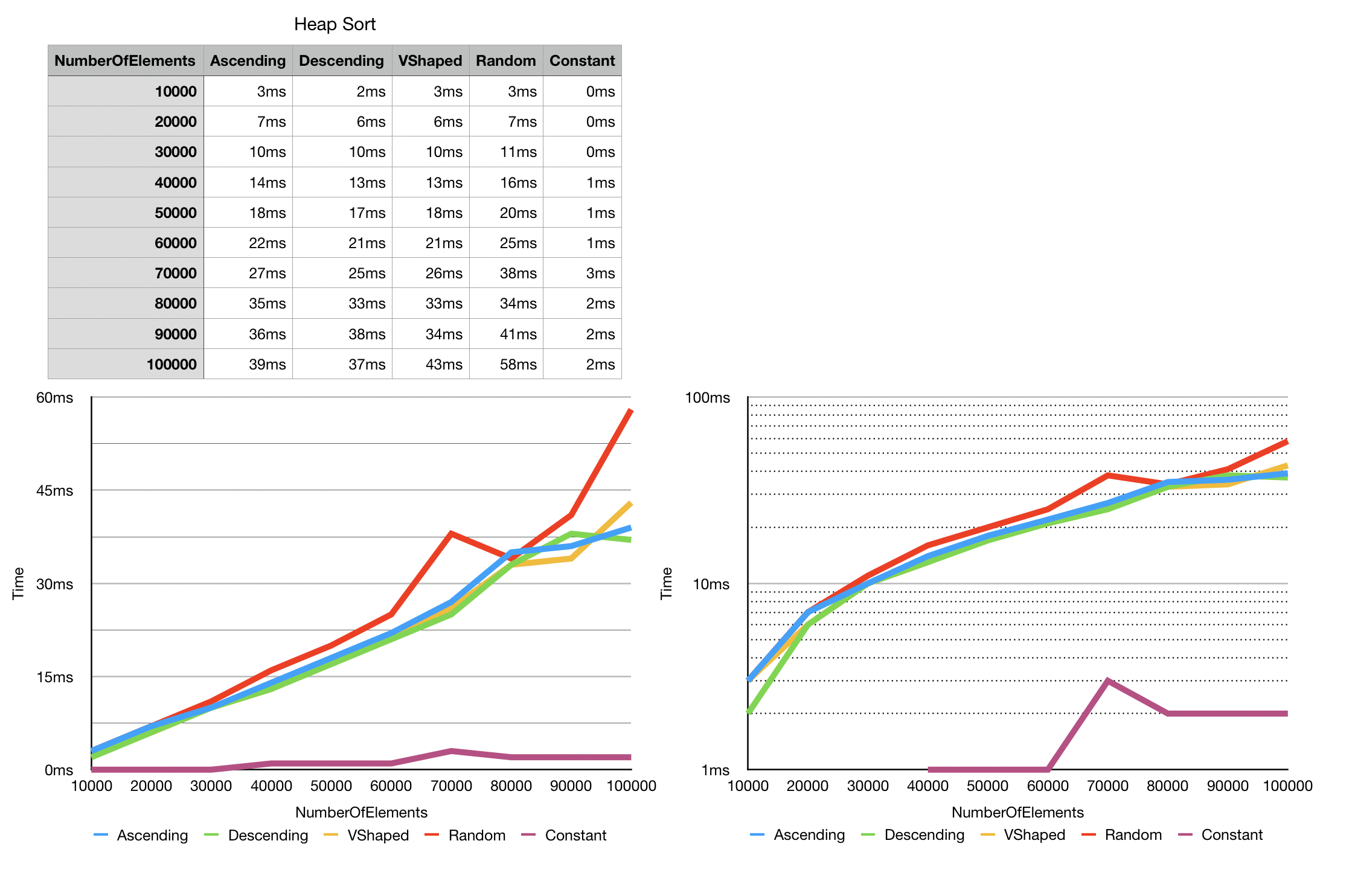
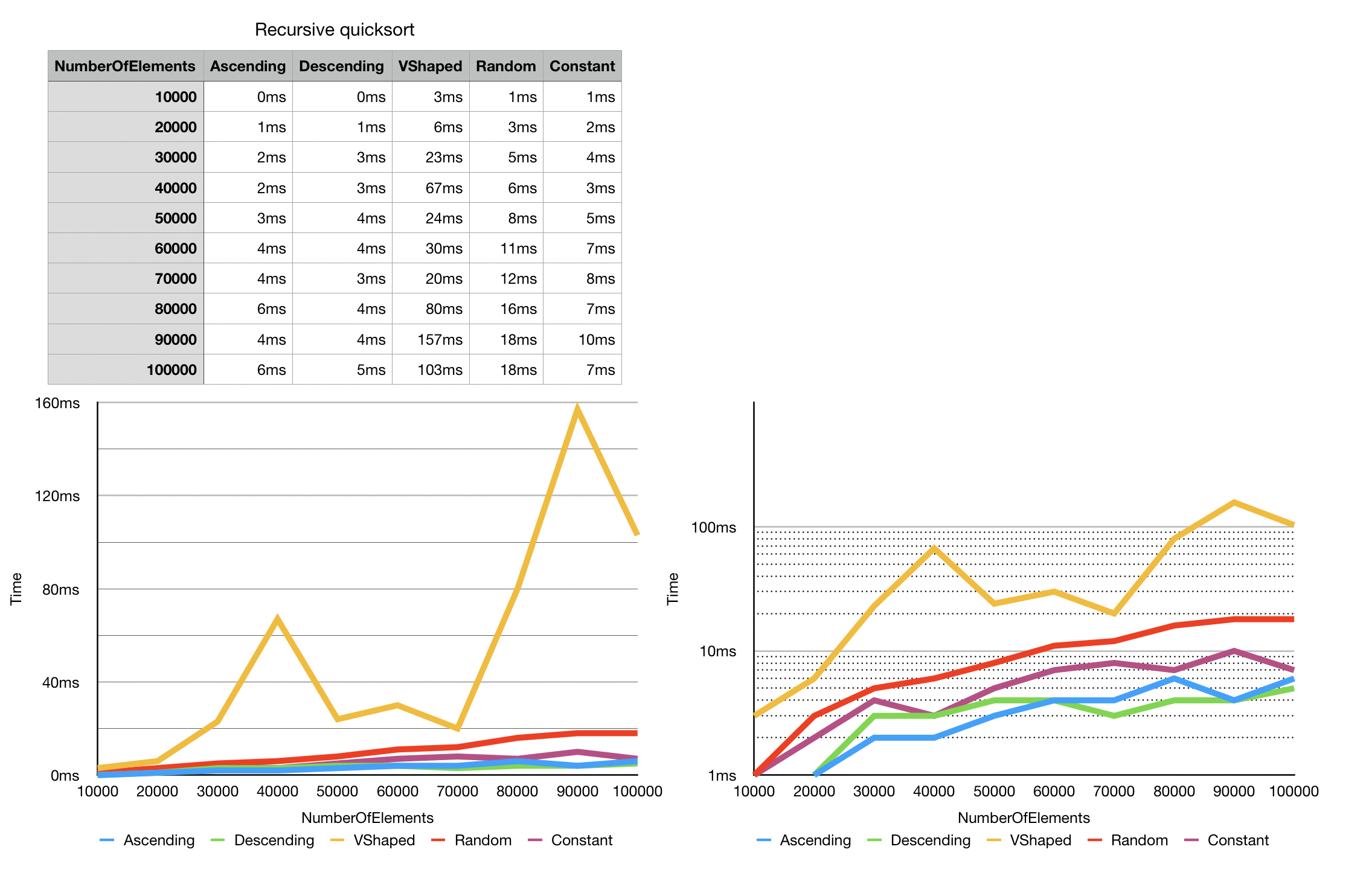
- skrajnie prawy klucz - nie radzi sobie zbyt dobrze z tablicami posortowanymi czas działania wynosi O(n2) , dzieje się tak ponieważ procedura partition podzieli jeden obszar na n-1 elementów, a drugi na 0 elementów. Najlepszy przypadek występuje gdy jeden z środkowych elementów znajduje się na końcu ciągu, dzięki czemu dane są dzielone na miarę równe części

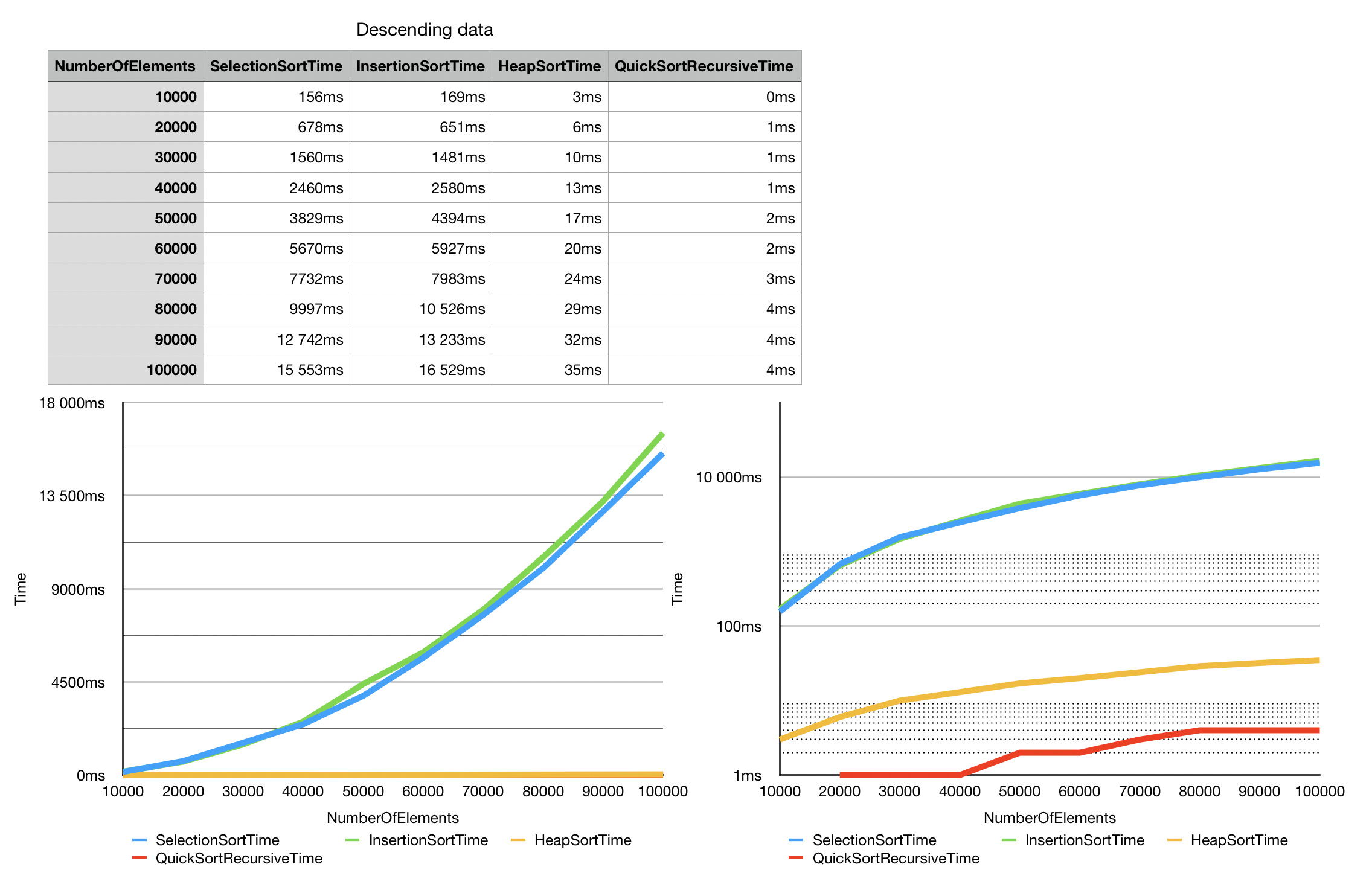
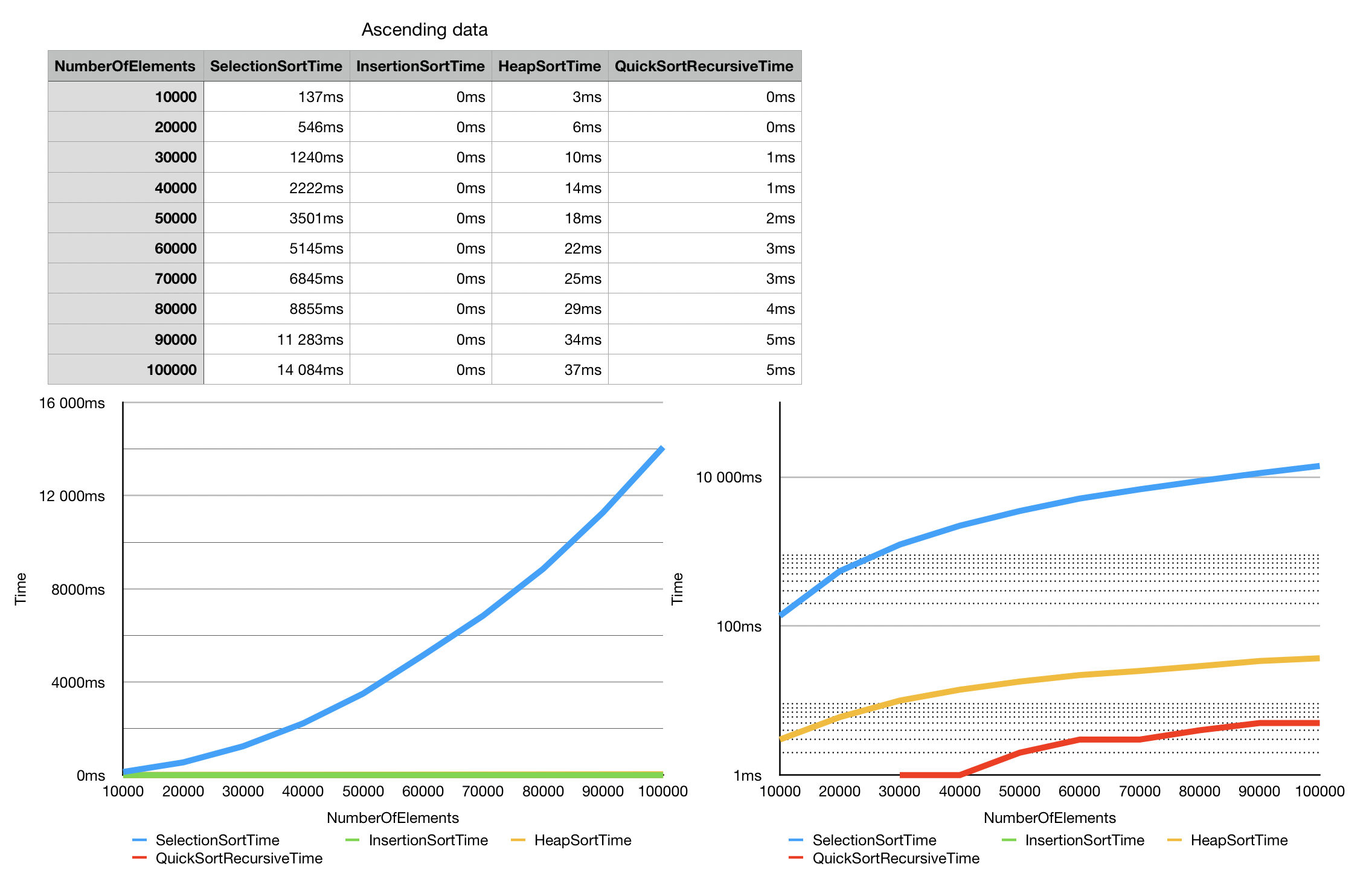
- środkowy klucz – tablice A i V kształtne są najgorszym przypadkiem dla tego klucza, jako ze na końcu ciągu występuje najmniejszy lub największy element co powoduje podział na n-1 elementów i 0 elementów. Klucz ten jest optymalny dla danych posortowanych

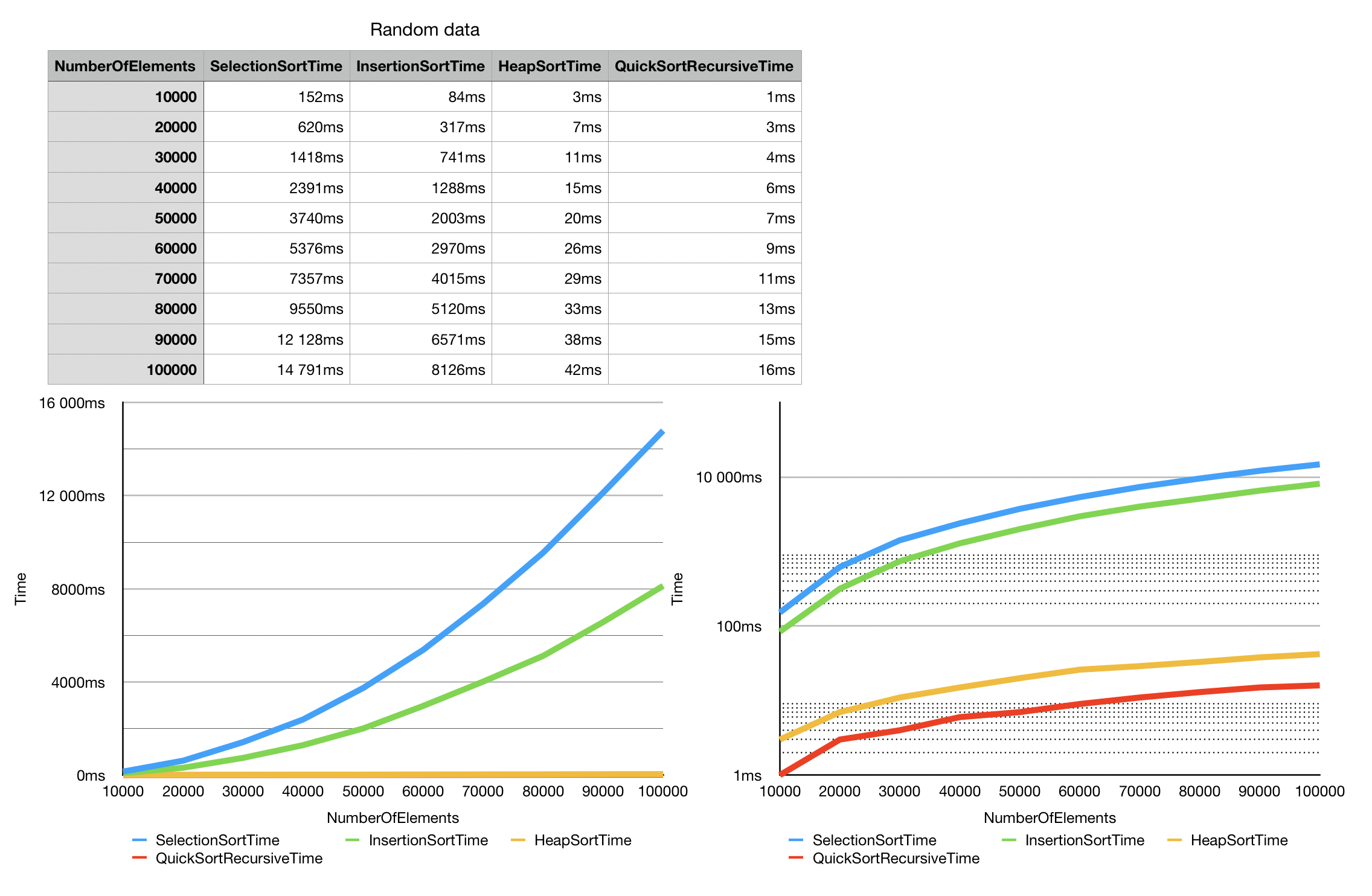
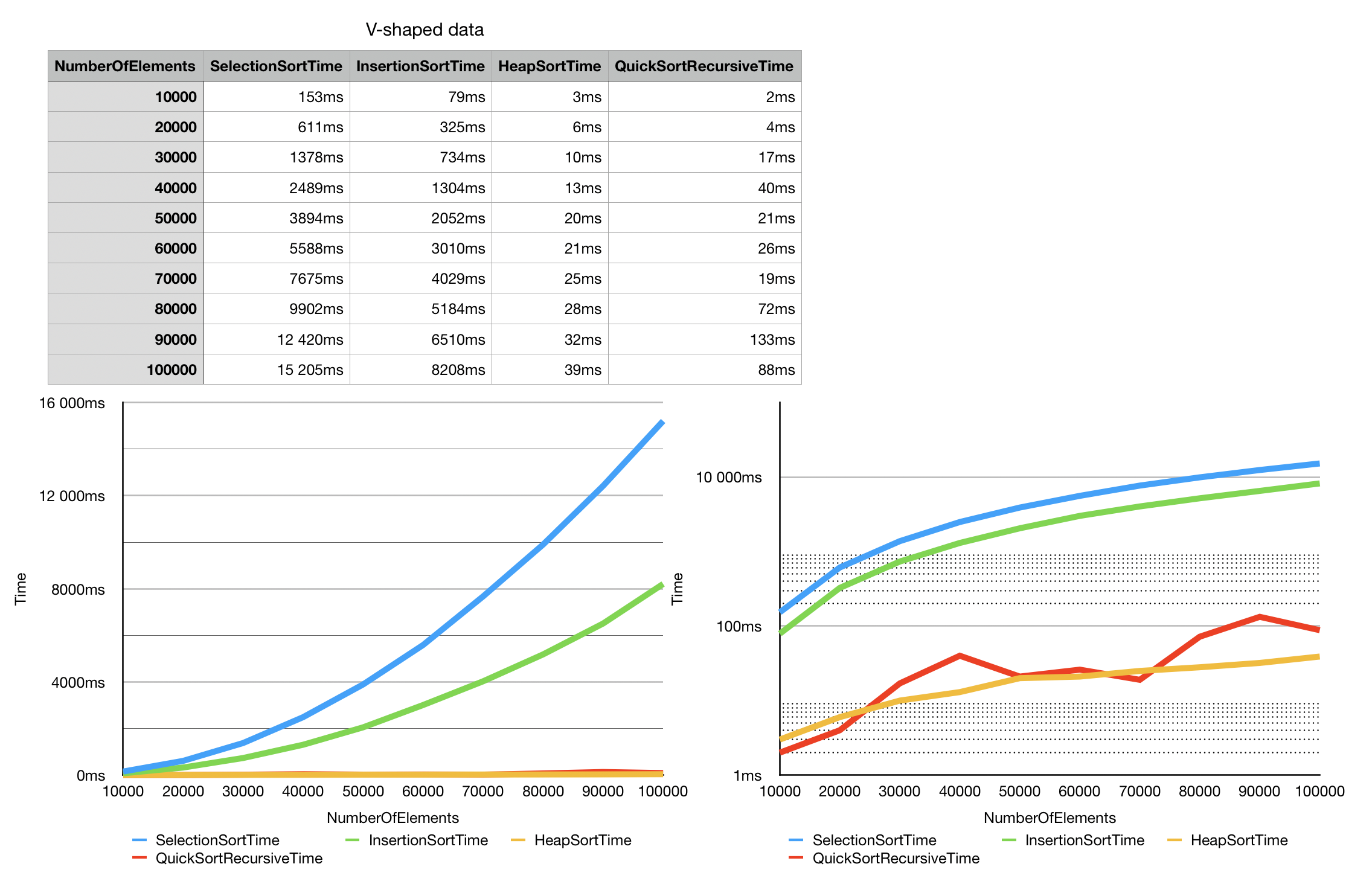
**Złożoność obliczeniowa:**

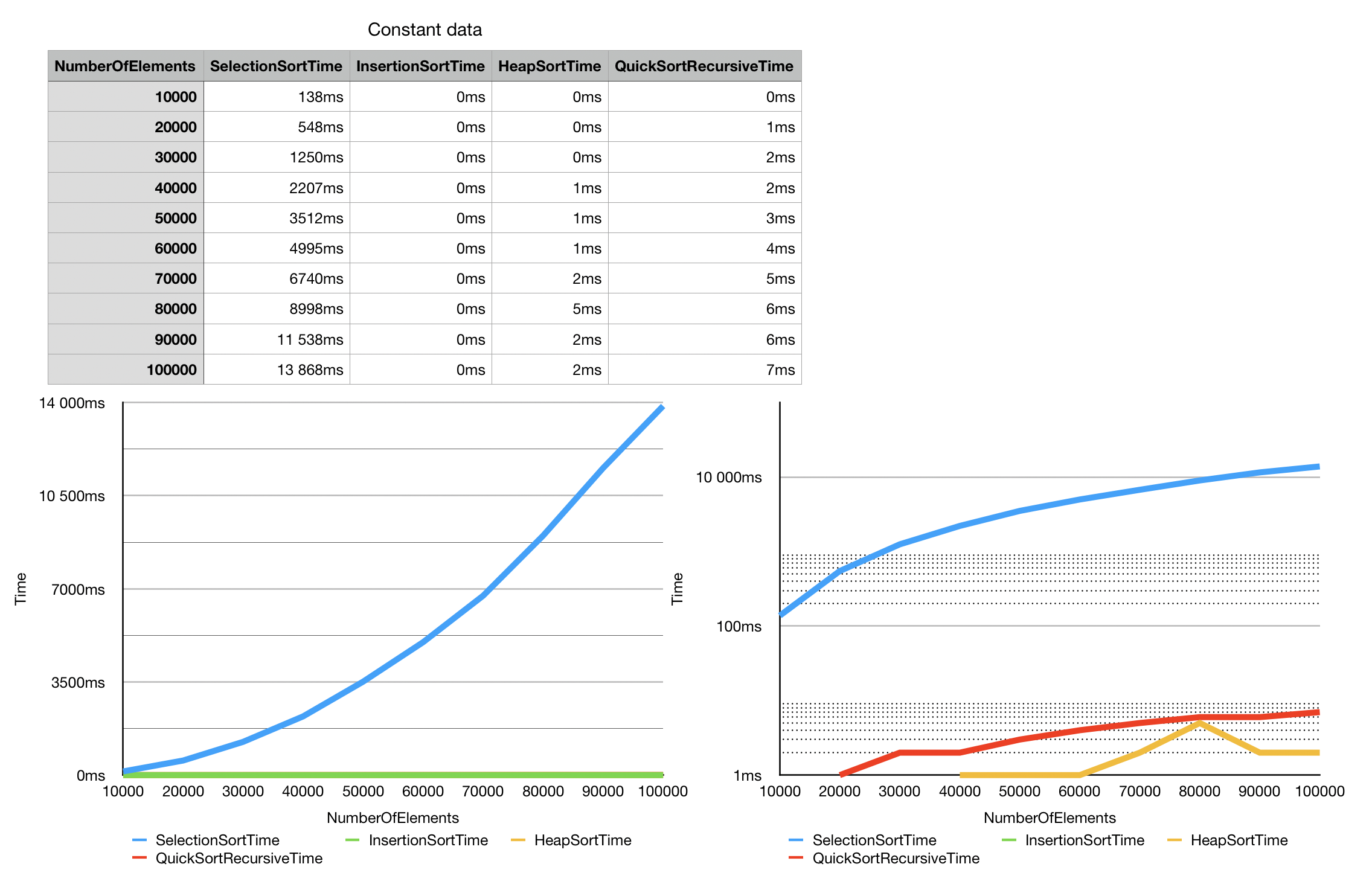
|  |  |  |  |
| --- | --- | --- | --- |
| Klucz\Przypadek | Najgorszy | Średni | Najlepszy |
| Losowy | O(n2) | O(nlog2n) | O(nlog2n) |
| Prawy | O(n2) | O(nlog2n) | O(nlog2n) |
| Środkowy | O(n2) | O(nlog2n) | O(nlog2n) |

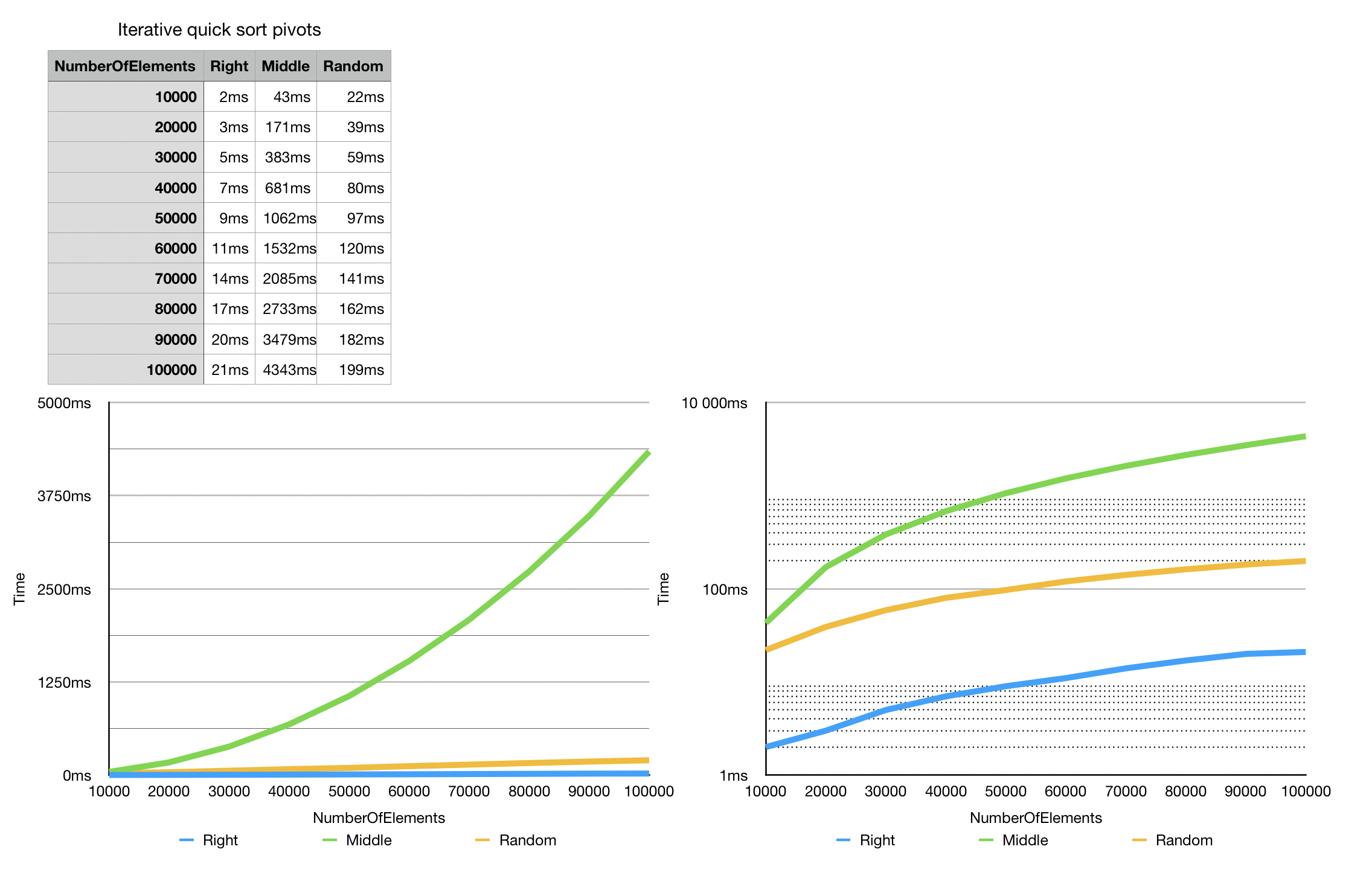
**Wykresy przedstawiające różnie algorytmy  
sortowania dla różnych typów danych**

****

****

****

****

**Wykres przedstawiający wyniki wyboru  
różnego rodzaju pivota dla iteracyjnego algorytmu Quick Sort**