



Fundusze Europejskie
Wiedza Edukacja Rozwój



Rzeczpospolita Polska



Unia Europejska
Europejski Fundusz Społeczny



**Zintegrowany Program Rozwoju
Akademii Górniczo-Hutniczej w Krakowie**
Nr umowy: POWR.03.05.00-00-Z307/17

Nazwa przedmiotu	Teoria współbieżności
Numer ćwiczenia	7
Temat ćwiczenia	Rozwiązywanie problemu pięciu filozofów w różnych paradygmatach programowania współbieżnego

Poziom studiów	1 stopień
Kierunek	Informatyka
Forma i tryb studiów	Stacjonarne
Semestr	5
Imię i nazwisko	Oscar Teezinga

Wprowadzenie

Wykorzystałem języki: Python, Java i JavaScript. W przypadku JS zastosowałem funkcję `setTimeout` realizując zadanie asynchronicznie, natomiast w Javie i Pythonie użyłem semaforów blokujących.

Realizowane przezem algorytmy to:

1. Wersja asymetryczna
2. Wersja z podnoszeniem obu widelców naraz
3. Wersja z kelnerem
4. Wersja z hierarchicznym przydzielaniem dostępu

Do rysowania wykresów w każdym języku zastosowałem inną bibliotekę, więc ciężko było zaprezentować wyniki na jednym wykresie bez zapisywania wyników do pliku .csv.

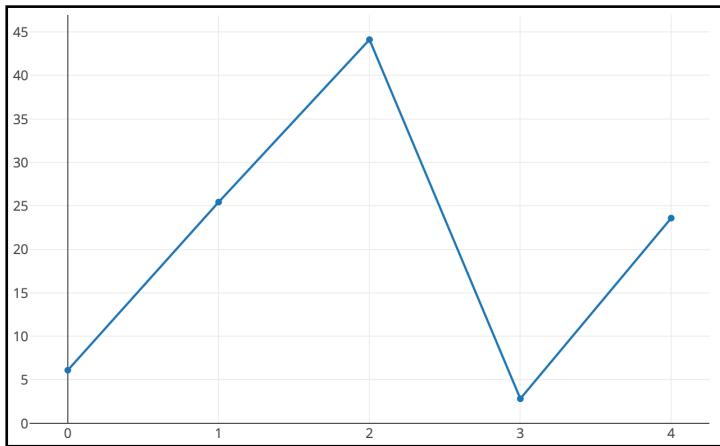
Dla wersji asynchronicznej zastosowałem algorytm BEB, przy czym w przypadku algorytmu z hierarchicznym dostępem oraz podnoszeniu obu widelców na raz musiałem ograniczyć zwiększenie czasu oczekiwania na kolejną próbę ze względu na mocną dyskryminację pewnych filozofów do zasobów, co znacznie wydłużało średni czas oczekiwania i sprawiało, że algorytmy te stawały się skrajnie nieefektywne.

Test przeprowadzałem dla 5, 50 i 250 filozofów, przy czym każdy jadł 100 razy, aby zwiększyć prawdopodobieństwo otrzymania dokładnych wyników.

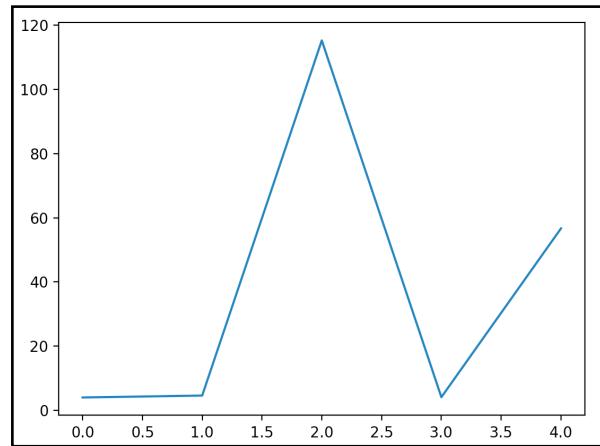
Wykresy dla Python i Javy były wyskalowane na 1e-6 sekundy, natomiast dla JavaScript na 1e-3 sekundy (JS nie jest w stanie zapisywać dokładniej, a w przypadku dwóch poprzednich `eps = 1e-3` mógł okazać się za mało dokładny).

W każdym przypadku czas jedzenia każdego filozofa oraz czas myślenia był równy 0 (stąd zapewne niesprawiedliwe wyniki dla Pythona).

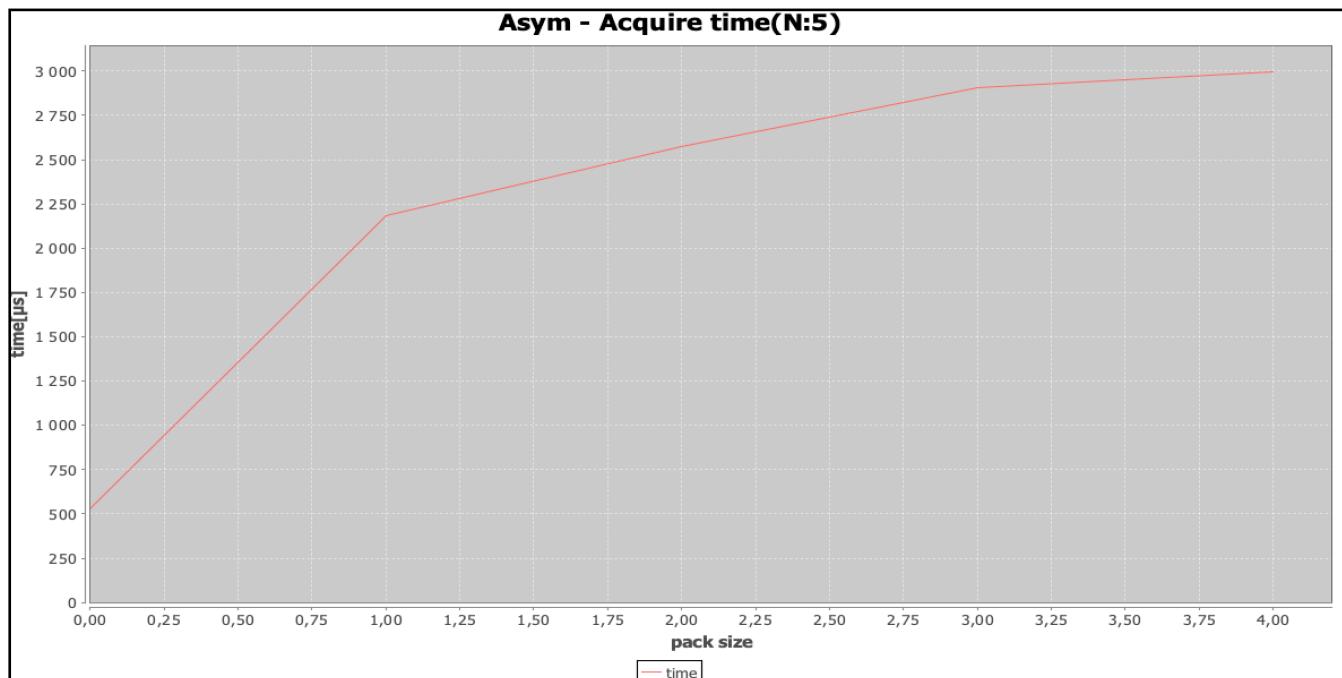
Wersja asymetryczna 5 filozofów



Wyk.: JavaScript dla wersji asymetrycznej



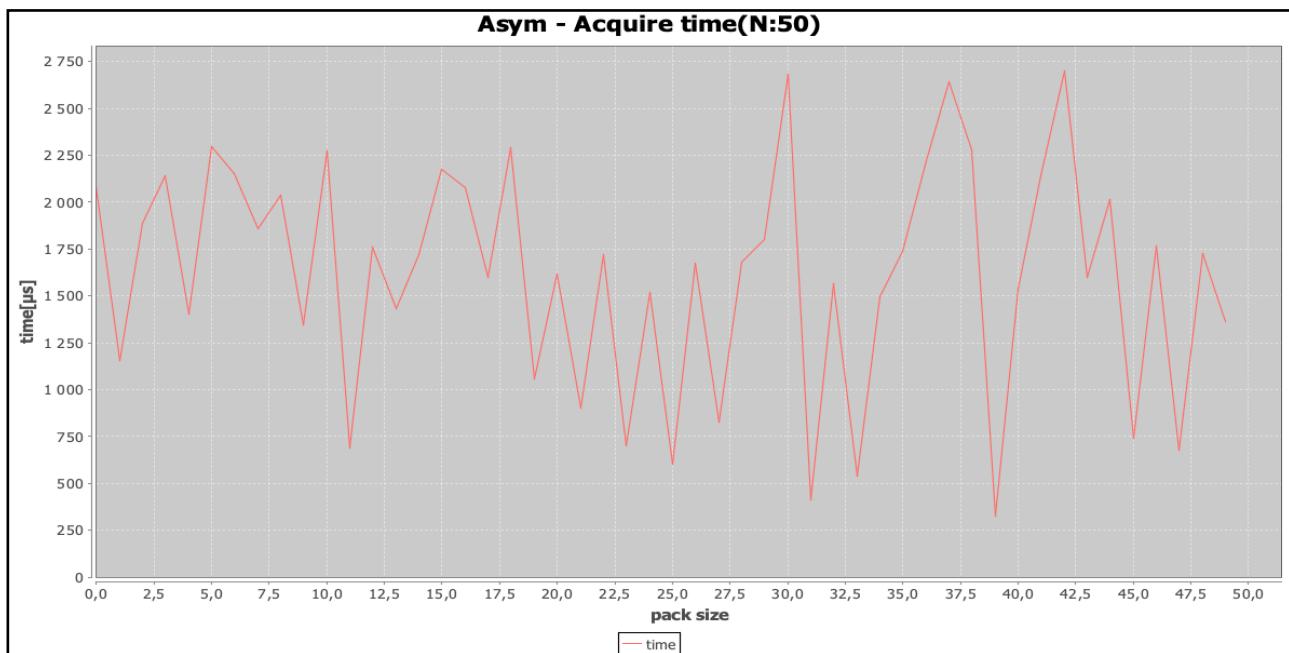
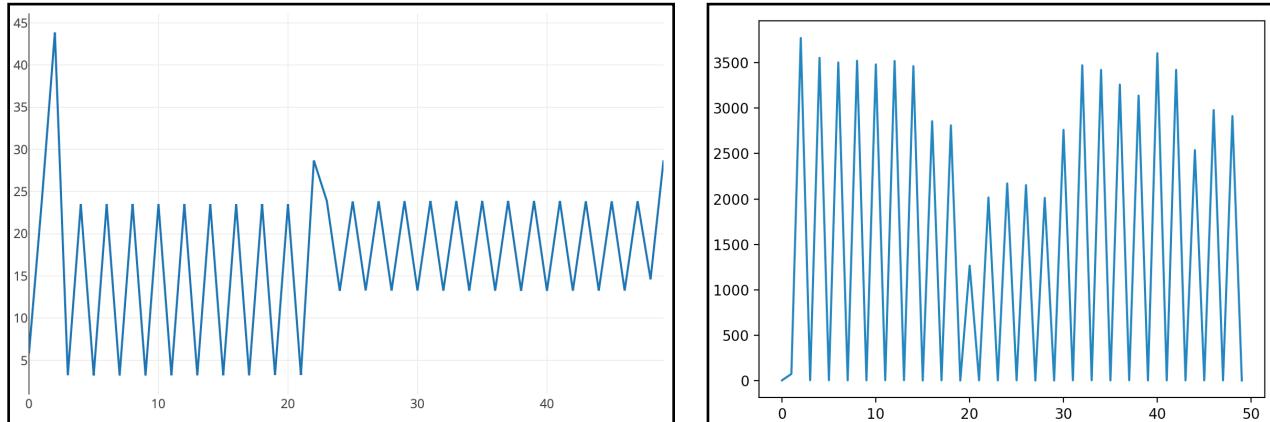
Wyk.: Python dla wersji asymetrycznej



Wyk.: Java dla wersji asymetrycznej

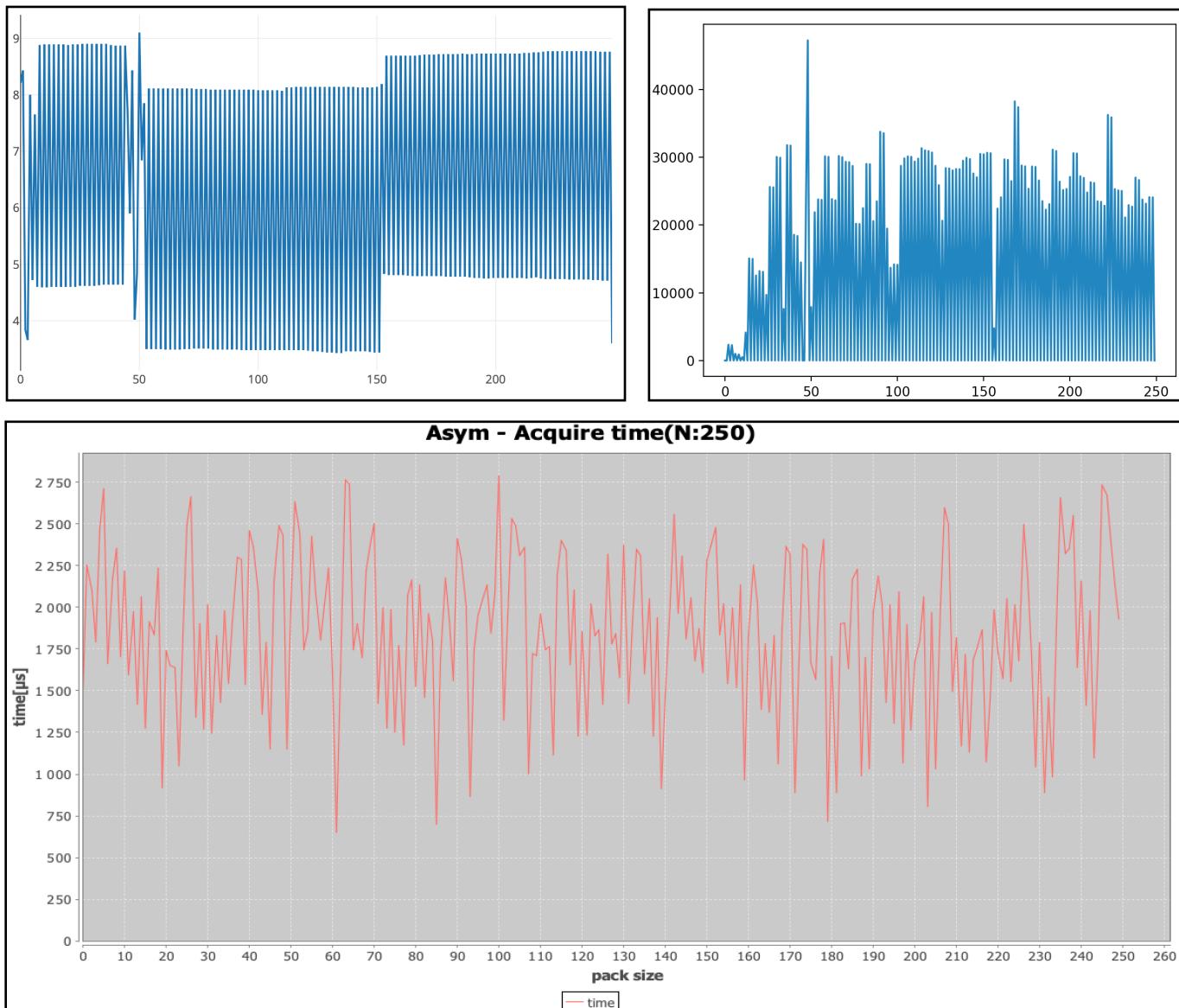
Jak widać na powyższych wykresach, właściwie dla tak małej ilości filozofów wyniki są losowe, przy czym dla JavaScript i Python widać głodzenie pewnych filozofów na rzecz innych.

Wersja asymetryczna 50 filozofów



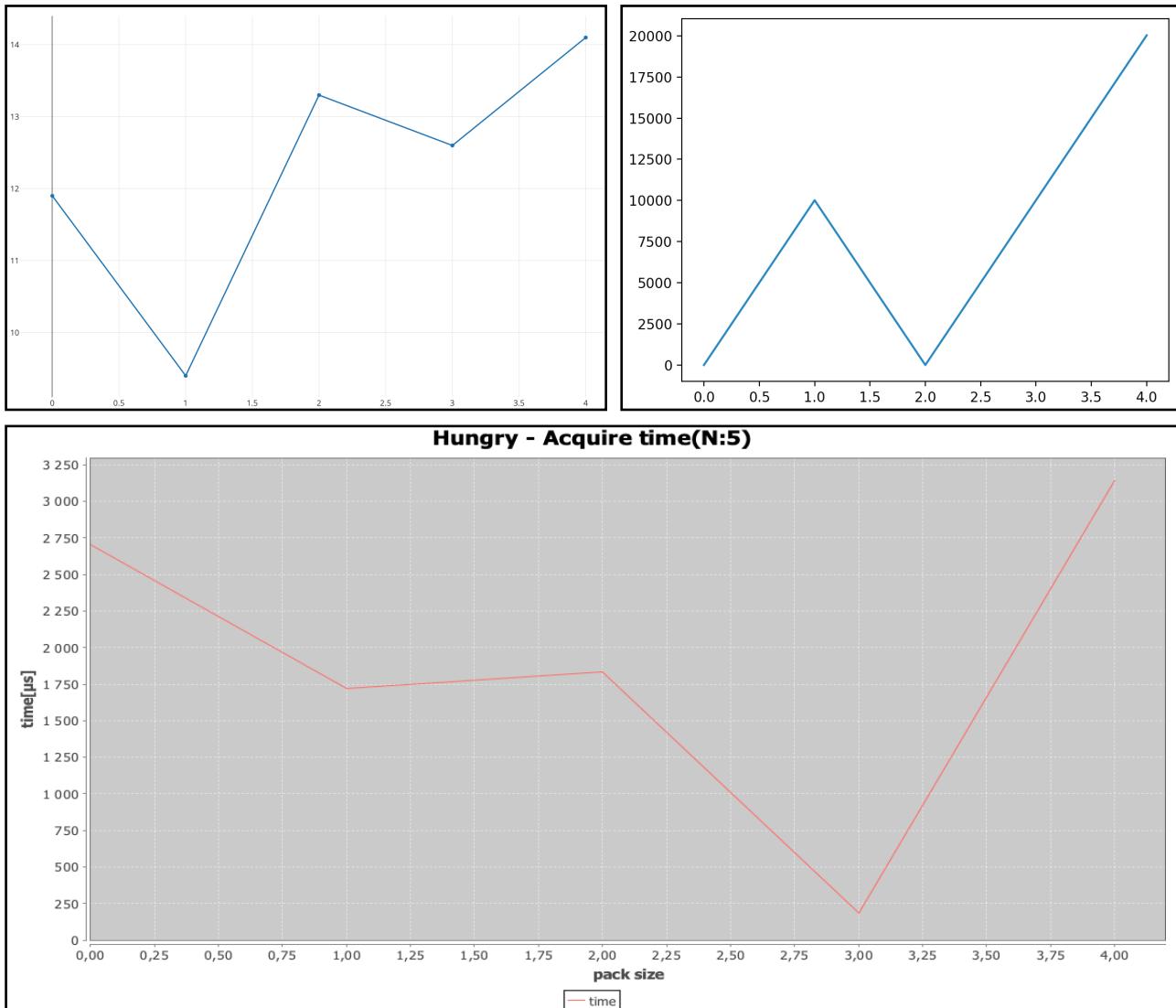
Dla większej ilości filozofów zaczyna być widoczny typowy dla wersji asymetrycznej okresowy charakter wykresu, w którym sąsiad uniemożliwia dostęp do widelca kolejnego. Widać również, że semafor Javy najlepiej zapobiega głodzeniu.

Wersja asymetryczna 250 filozofów



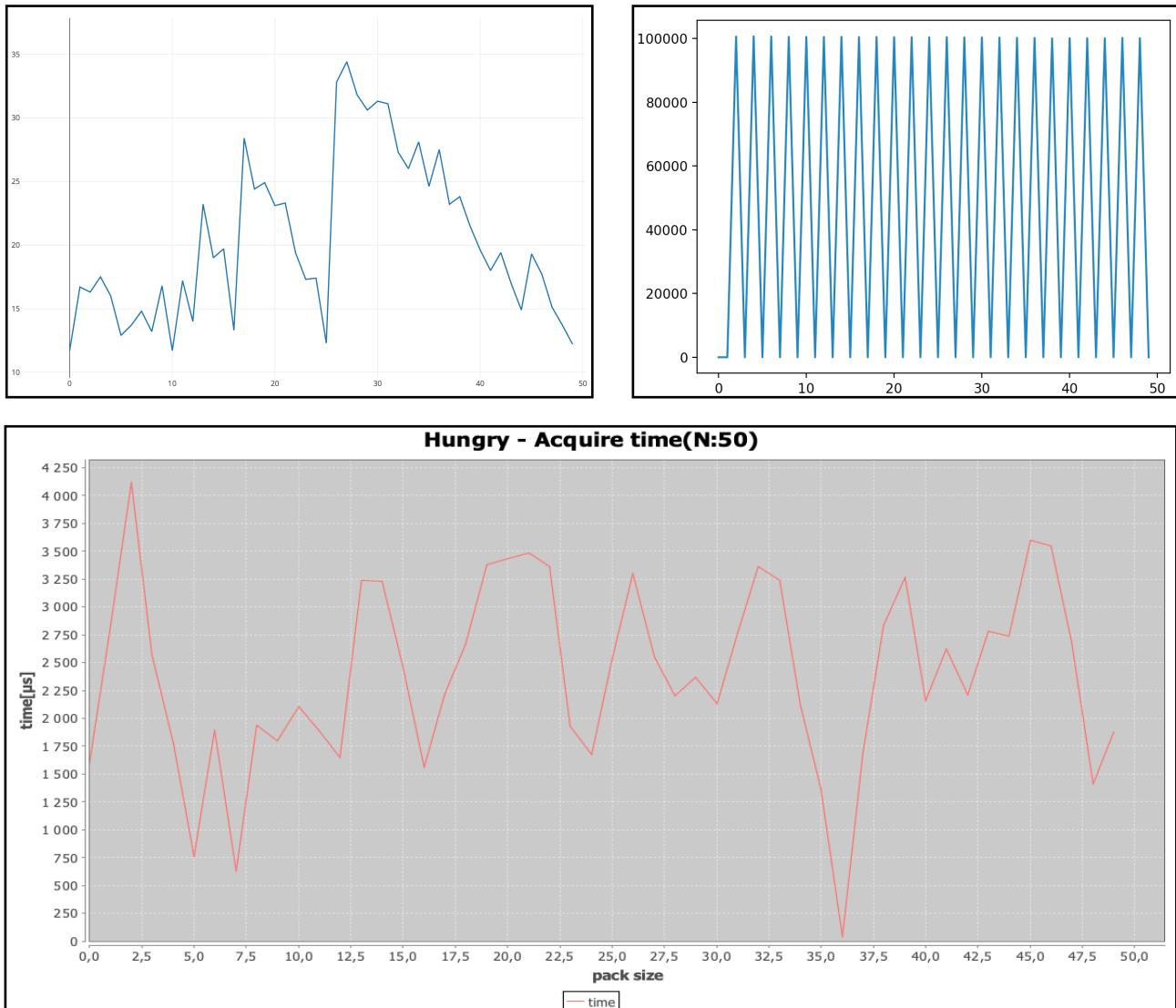
Dla 250 filozofów najlepiej uwydatnia się głodzenie sąsiadów dla wersji asymetrycznej. Warto zauważać, że wersja asynchroniczna w tej wersji osiąga najszybsze czasy.

Wersja z jednoczesnym pobraniem obu widelców 5 filozofów



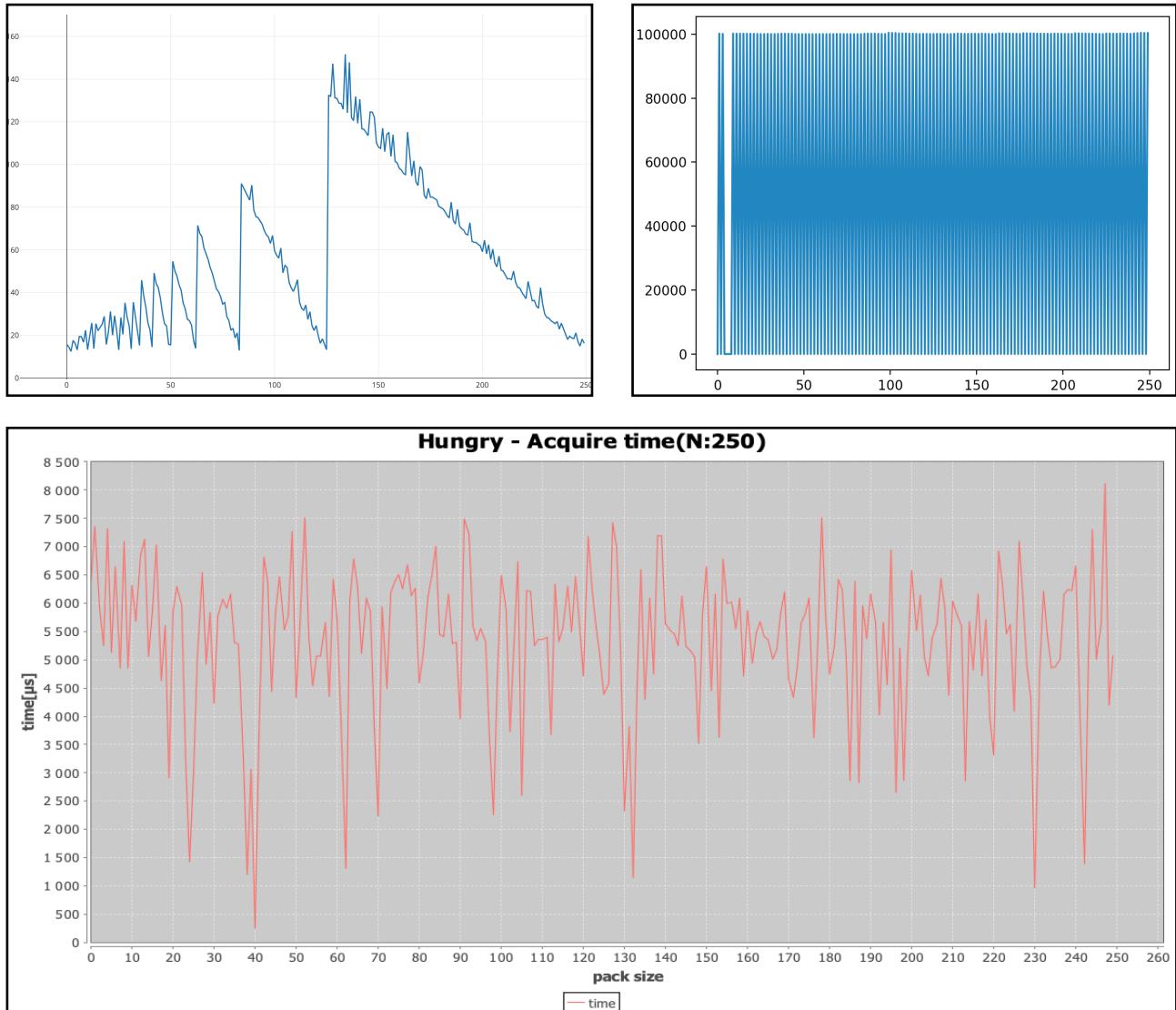
Sytuacja się powtarza, wydaje się, że wyniki są losowe, przy czym widać, że Python kiepsko czasowo radzi sobie z tym rozwiązaniem.

Wersja z jednoczesnym pobraniem obu widelców 50 filozofów



Python po raz kolejny fatalnie radzi sobie czasowo i w sposób ewidentny faworyzuje niektóre procesy. Podejście asynchroniczne generuje ciekawy wzór, który uwidocznii się dla większej ilości filozofów, natomiast Java wydaje się głodzić sąsiadów, jednak w nieznacznym stopniu.

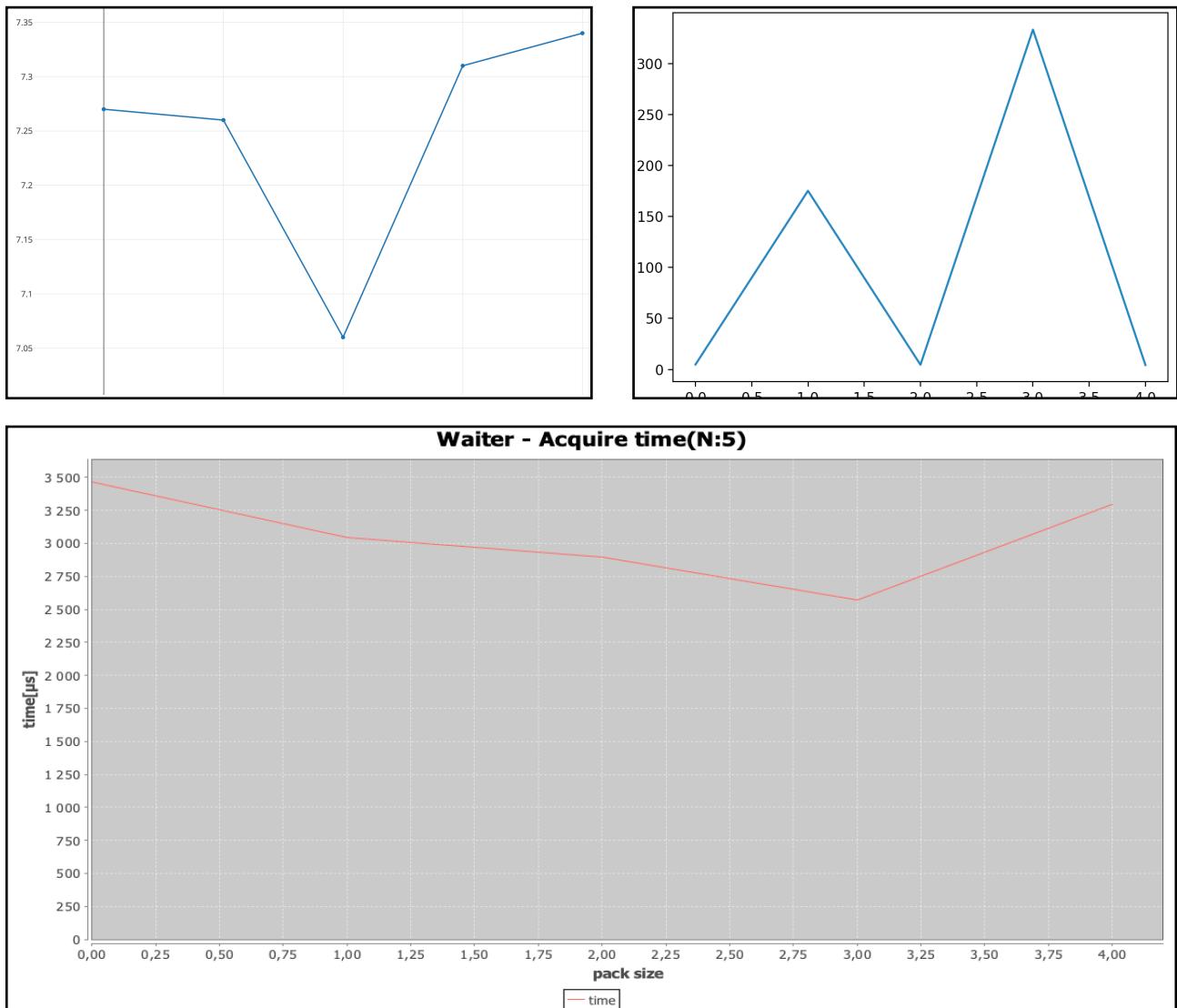
Wersja z jednoczesnym pobraniem obu widelców 250 filozofów



Bardzo ciekawy wykres dla Python i JS, natomiast Java generuje najbardziej równomierne rozłożenia czasów, osiągając w tym przypadku najlepszy wynik czasowy.

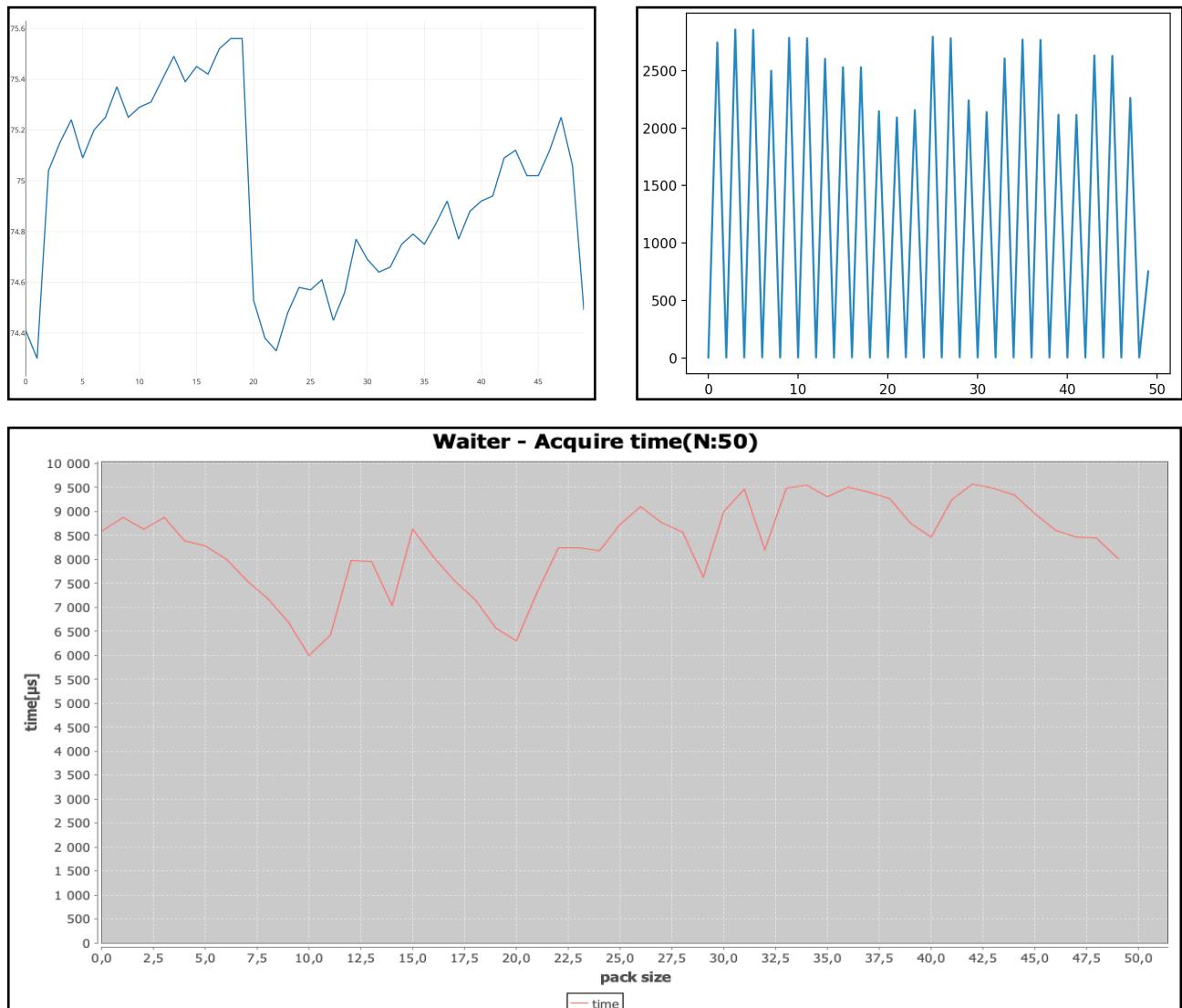
Interesujący wykres JS wynika z tego, że jeżeli jakiś filozof już dorwał się do obu widelców, to potem ze względu na algorytm BEB nie oddawał ich aż zakończył swoje działanie. Pythona nie umiem wytlumaczyć jak kiepskim dostępem filozofów do semaforów i skrajną dyskryminacją niektórych.

Wersja z kelnerem 5 filozofów



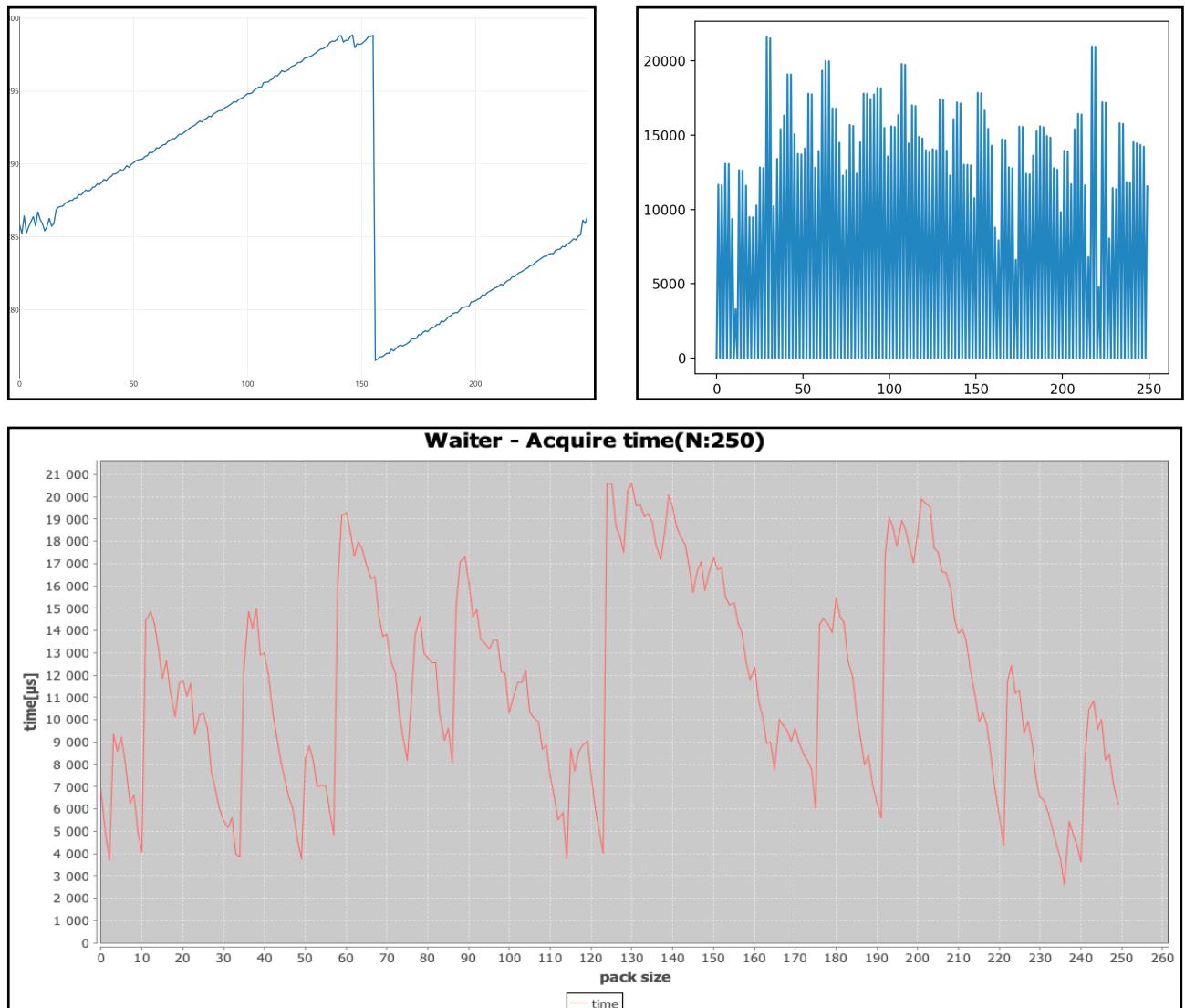
W tym przypadku nie ma mowy o losowych wynikach dla małej ilości filozofów: jest to wariant najbardziej sprawiedliwy. Java i JavaScript osiągnęły wręcz równy czas dla każdego filozofa. Natomiast tak jak wcześniej, Python zachowuje się skrajnie nieoczekiwane i działa zupełnie losowo.

Wersja z kelnerem 50 filozofów



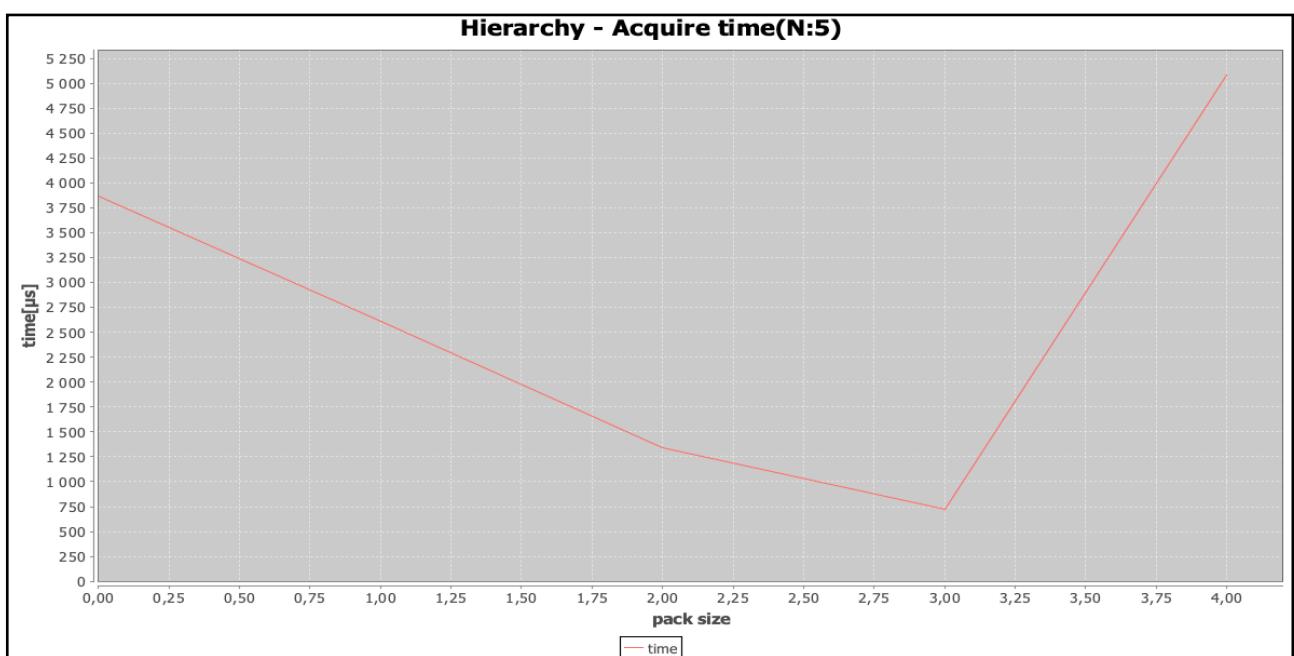
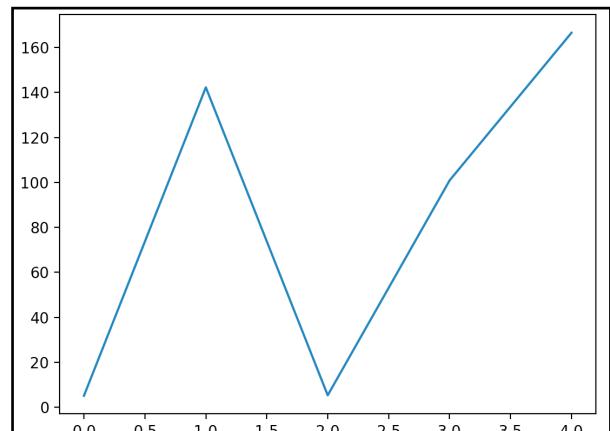
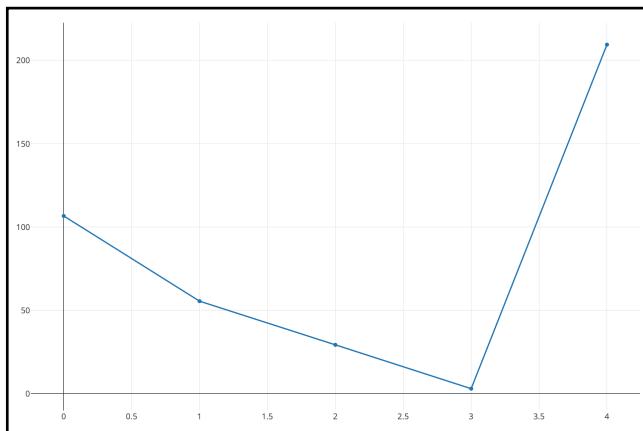
Ponownie, kelner w Java i JavaScript w sposób bardo sprawiedliwy rozporządza dostępem do widelców, generując piękne, równe wyniki dla każdego filozofa. Warto jednak zauważyć, że czasowo wypada to nieco gorzej. Python zachowuje się tak samo jak dla wersji asymetrycznej, co jest zupełnie nieoczekiwane.

Wersja z kelnerem 250 filozofów



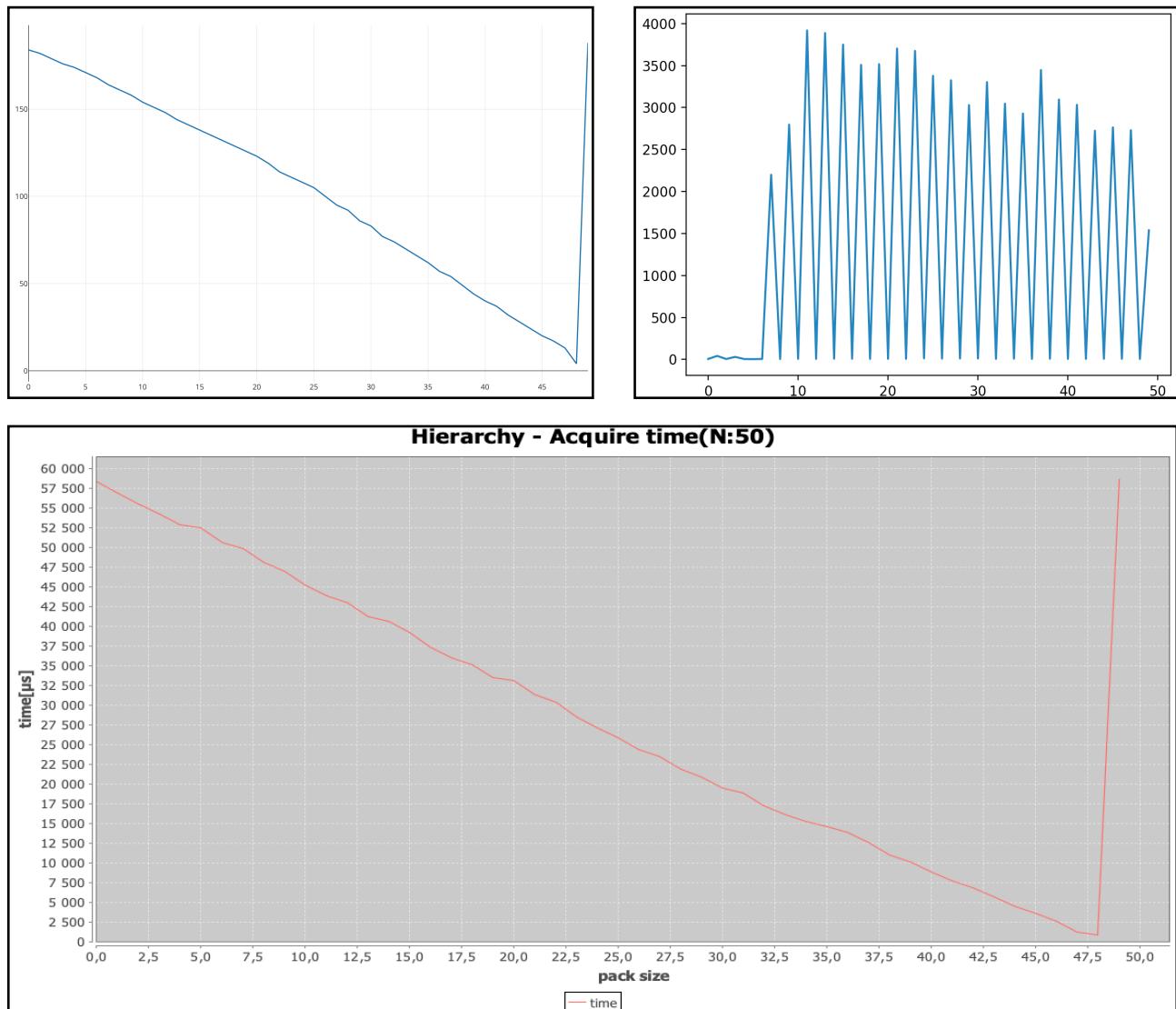
Tym razem widać przewagę JavaScriptu, czasy dla każdego filozofa są praktycznie równe, wręcz niewyobrażalne. Natomiast Java już nieco spowalnia i znacznie dyskryminuje pewnych filozofów. Python po raz kolejny, zupełnie dziwnie.

Wersja z hierarchią 5 filozofów



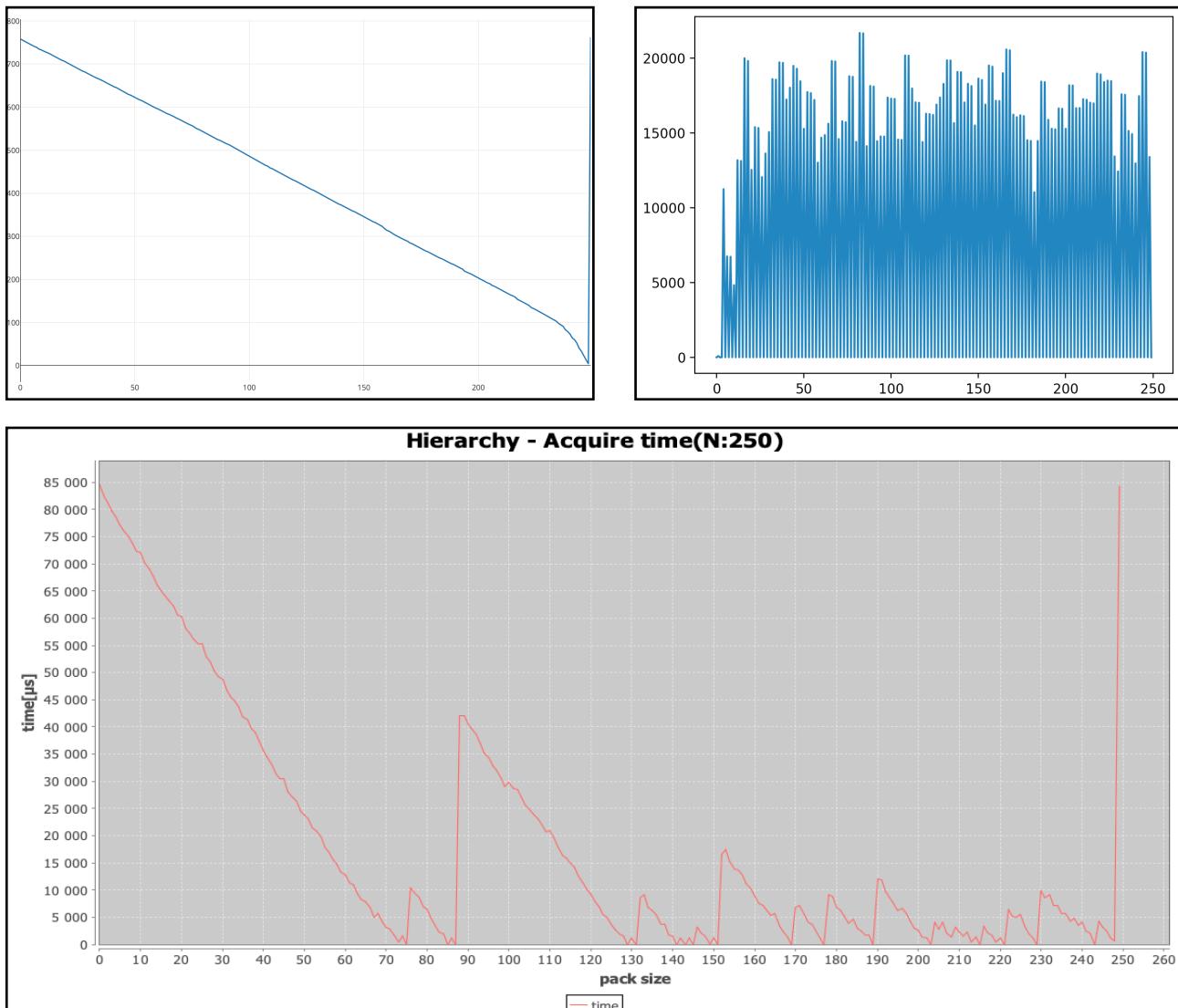
Specjalnie wybrałem ten algorytm, ponieważ widać tutaj doskonale jak głodzony jest ostatni filozof, który następnie musi czekać, aż zjedzą absolutnie wszyscy, żeby dostać drugi widelec o numerze 0.

Wersja z hierarchią 50 filozofów



Efekt się uwypukla, przy czym Python znów zachowuje się jakbyśmy oczekiwali dla wersji asymetrycznej.

Wersja z hierarchią 250 filozofów



Po raz kolejny widać głodzenie pewnych filozofów na końcu i oczekiwanie aż pewna ilość poprzedzająca go filozofów zwolni zasób. Do działania tego algorytmu niezbędne było zmniejszenie mnożnika oczekiwania dla BEB z 2 do 1.00001, inaczej ostatni filozof musiał czekać bardzo długo na swoją kolej. Python znów pokazuje taki sam wykres jak zwykle, zupełnie losowo z tendencją asymetryczną.

Podsumowanie

Z powyższych rozwiązań wydaje się, że każdy algorytm ma swoje wady i zalety, gdyż zdecydowanie najszybszy okazał się wariant asymetryczny, jednak najbardziej kompromisowym rozwiązaniem jest kelner. Natomiast językiem najszybszym, najbardziej wszechstronnym i przewidywalnym do programowania współbieżnego okazała się Java. Czasowo co prawda Python dla bardzo małej ilości filozofów konkurował z Javą, nie wspominając o JavaScircie: algorytm BEB w swoją pracę miał wpisany delay czasowy 1 ms wraz z propagacją tego czasu, która automatycznie rujnowała jakikolwiek możliwość czasowej rywalizacji z Javą.

Wydaje się, że Python nieoczekiwanie udziela dostępu do semaforów, przez to wykresy przedstawiają czasy zupełnie nieoczekiwane (bądź moja implementacja była nieprawidłowa).