Symulacja algorytmów planowania czasu procesora oraz wymiany stron

Autor: Karol Koenig

Planowanie czasu procesora

Opis algorytmów

LCFS (Last Come First Serve) – wersja niewywłaszczeniowa

LCFS w wersji niewywłaszczeniowej obsługuje procesy w odwrotnej kolejności ich przybycia. Oznacza to, że proces, który przybył najpóźniej, jest wykonywany jako pierwszy, a starsze procesy czekają na zakończenie bieżącego. Algorytm ten jest prosty w implementacji, ale może prowadzić do długiego czasu oczekiwania dla procesów przybyłych wcześniej (zjawisko głodzenia). Algorytm zaimplementowano za pomocą struktury danej stosu, zaimplementowanej w src/sim/ds.go.

LCFS (Last Come First Serve) – wersja wywłaszczeniowa

W wersji wywłaszczeniowej LCFS natychmiast przerywa wykonywanie bieżącego procesu, jeśli przybywa nowy proces. Nowy proces jest wykonywany jako pierwszy, a starsze procesy trafiają na koniec stosu. Algorytm ten maksymalizuje czas reakcji dla nowych procesów, ale może prowadzić do częstych przerwań i nierównomiernego obciążenia procesora. Algorytm zaimplementowano za pomocą struktury danej stosu, zaimplementowanej w src/sim/ds.go

SJF (Shortest Job First) – wersja niewywłaszczeniowa

SJF niewywłaszczeniowy wybiera proces o najkrótszym czasie wykonywania spośród dostępnych w kolejce i wykonuje go do końca. W przypadku równych czasów wykonania decyzja może być podejmowana na podstawie kolejności przybycia procesów. Algorytm ten minimalizuje średni czas oczekiwania, ale nie reaguje dynamicznie na nowe procesy o krótszym czasie wykonania. Algorytm zaimplementowano za pomocą struktury danej min heap zaimplementowanej w src/sim/process/process.go

SJF (Shortest Job First) – wersja wywłaszczeniowa

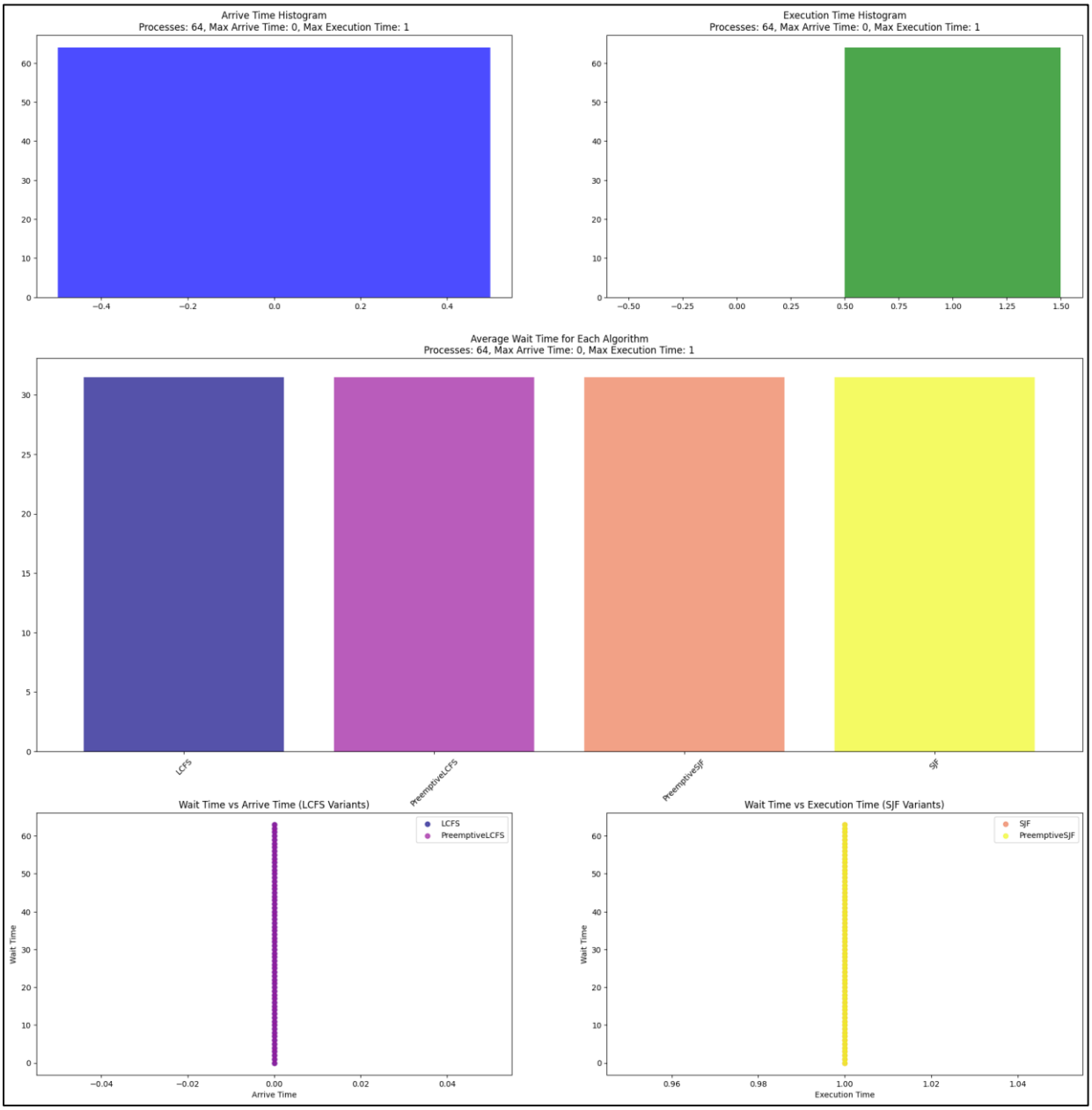
SJF wywłaszczeniowy zawsze wybiera proces o najkrótszym pozostałym czasie wykonywania, nawet jeśli wymaga to przerwania bieżącego procesu. Nowy proces o krótszym czasie wykonania od razu zastępuje procesy o dłuższym czasie w kolejce. Ten algorytm skutecznie minimalizuje czas oczekiwania, ale może prowadzić do większej liczby przerwań i zwiększenia narzutu systemowego. Algorytm zaimplementowano za pomocą struktury danej min heap zaimplementowanej w src/sim/process/process.go

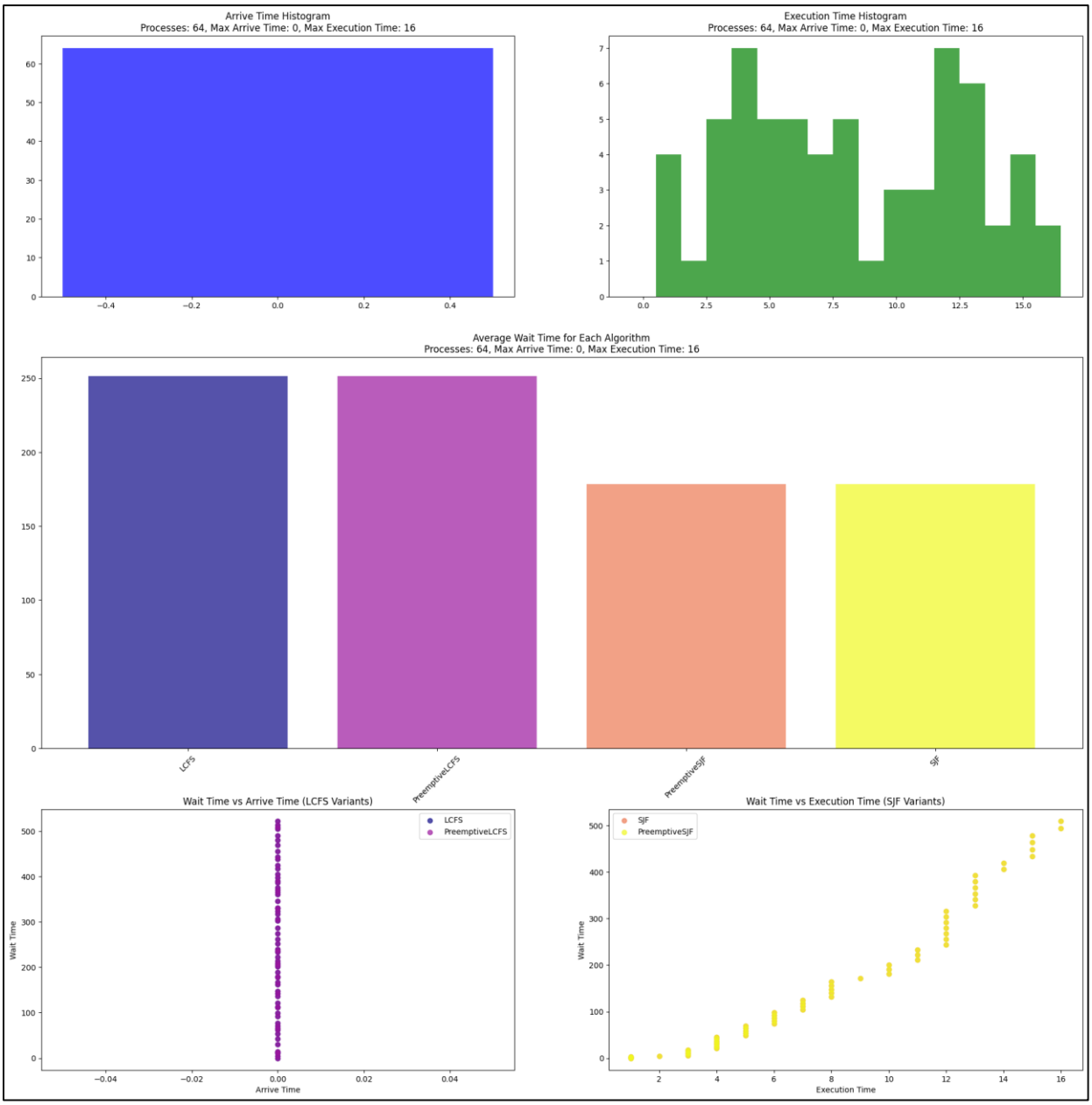
Opis procedury testowania algorytmów planowania czasu procesora

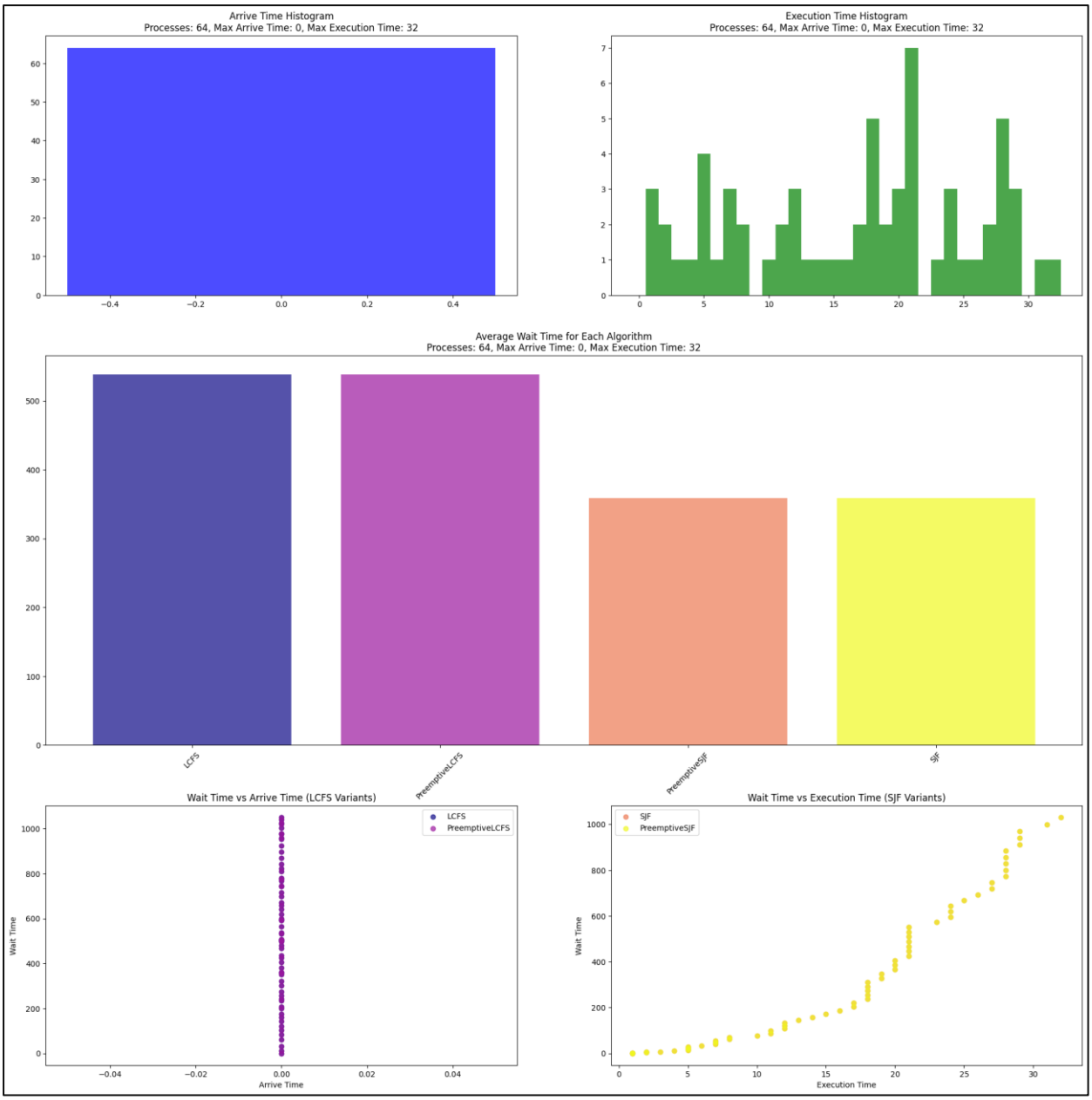
W testach generowane były zestawy 64, 128 i 256 procesów o czasach przyjścia równych 0, oraz z rozkładów równomiernych od 0 do 128 i od 0 do 512, i o czasach wykonania równych 1 i z rozkładów równomiernych od 1 do 16 i od 1 do 32. Łącznie daje to 27 zestawów danych wejściowych.

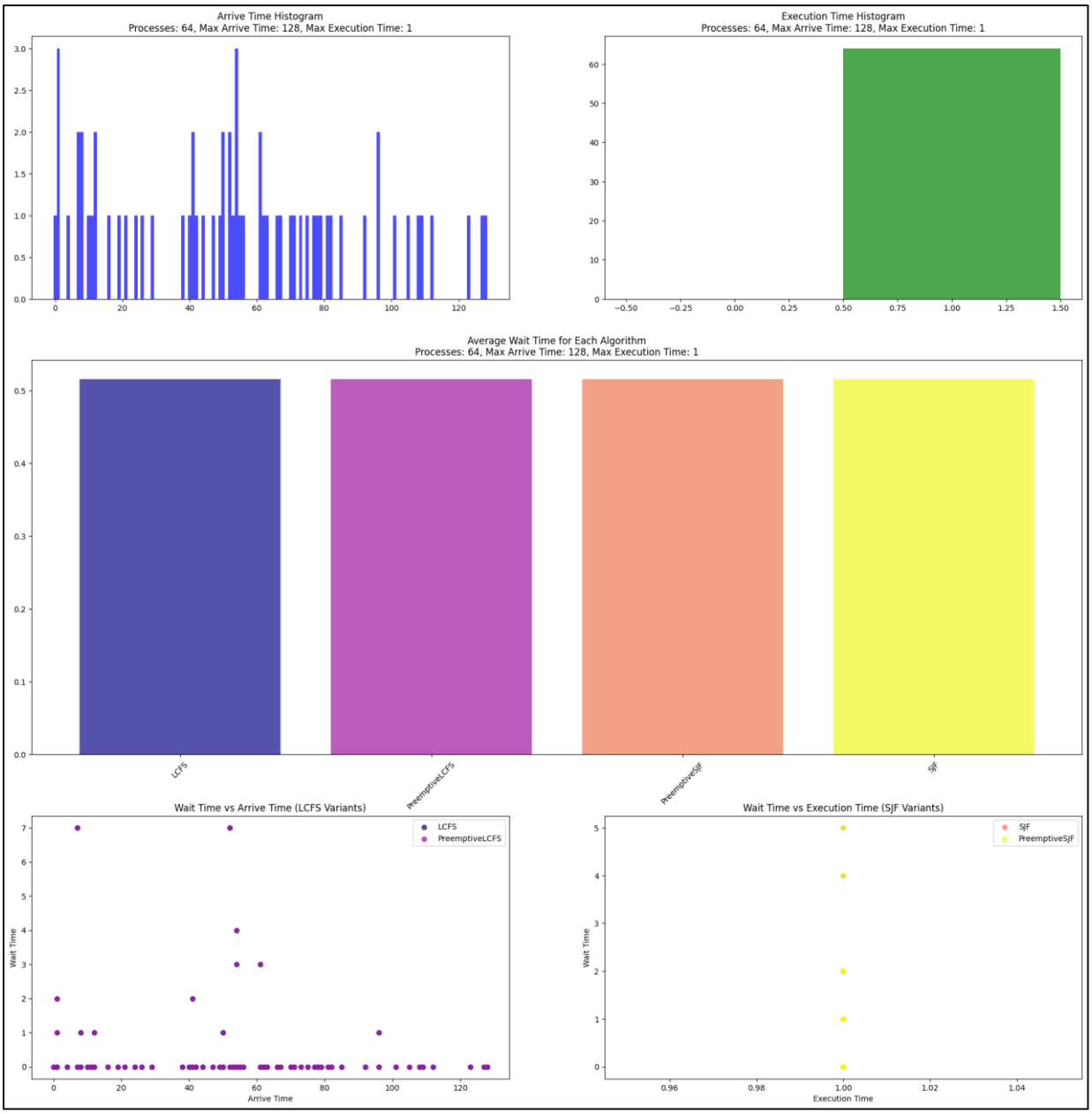
Wyniki testów algorytmów planowania czasu procesora:

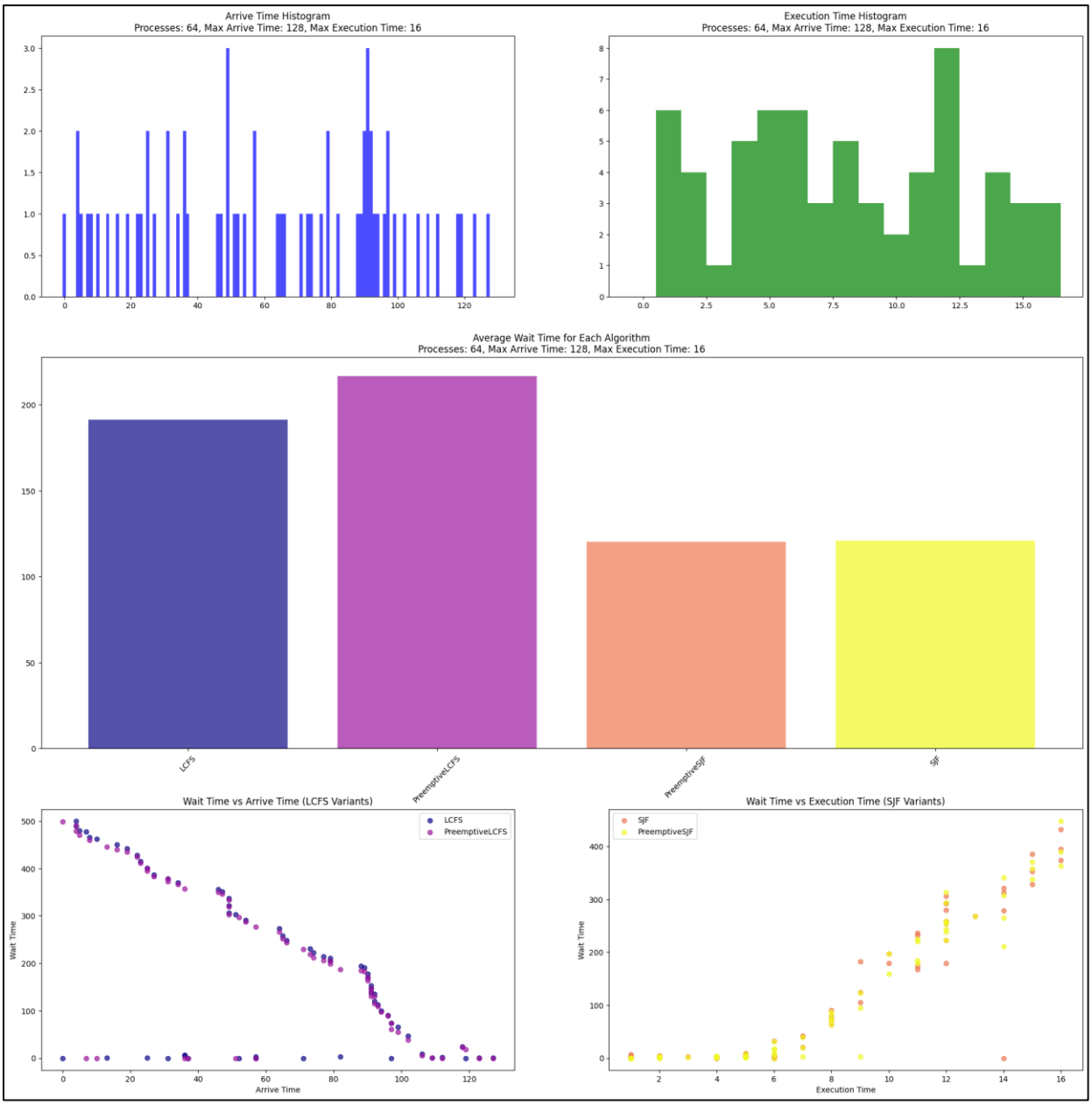
- *64 procesy:*

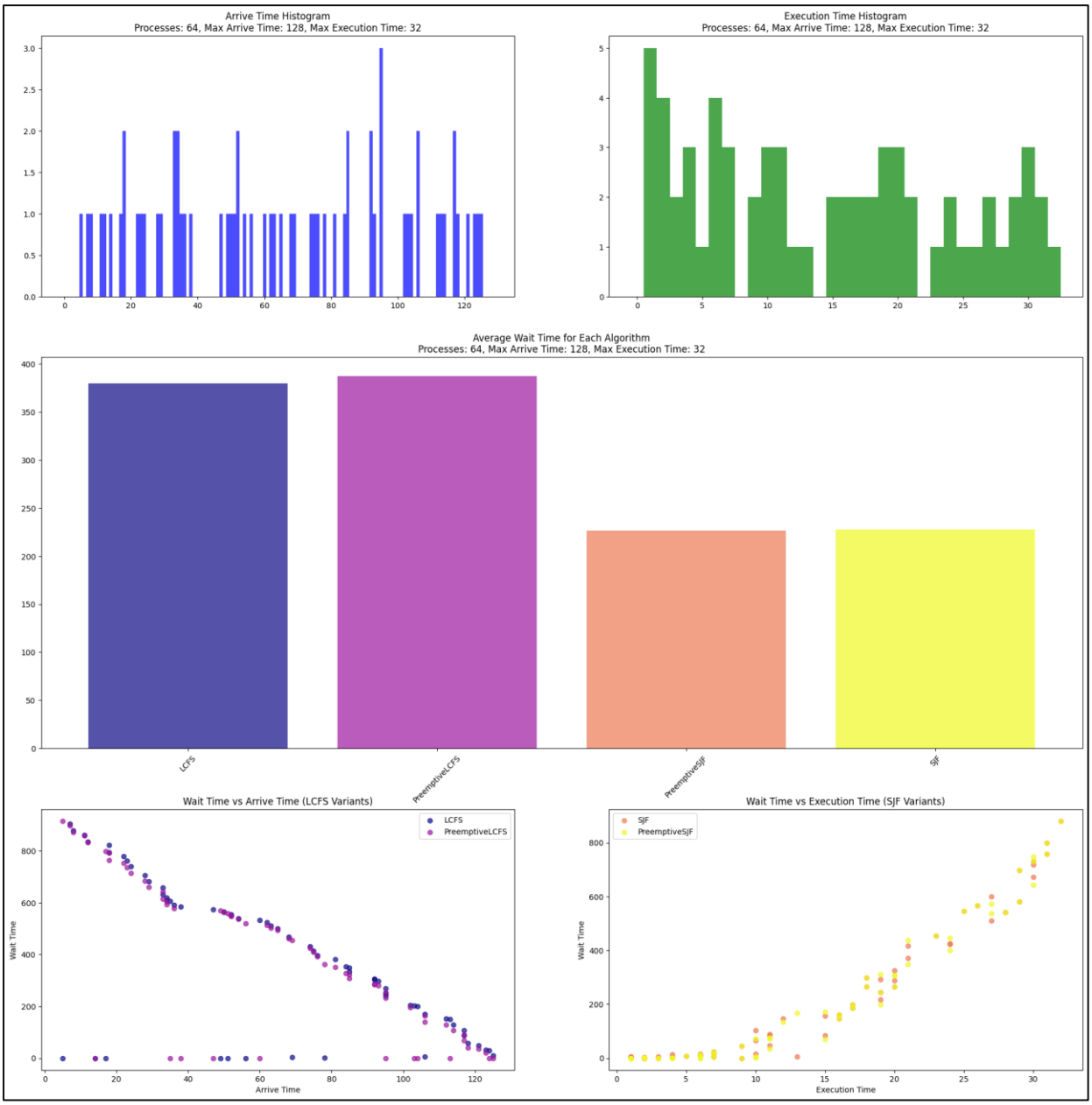


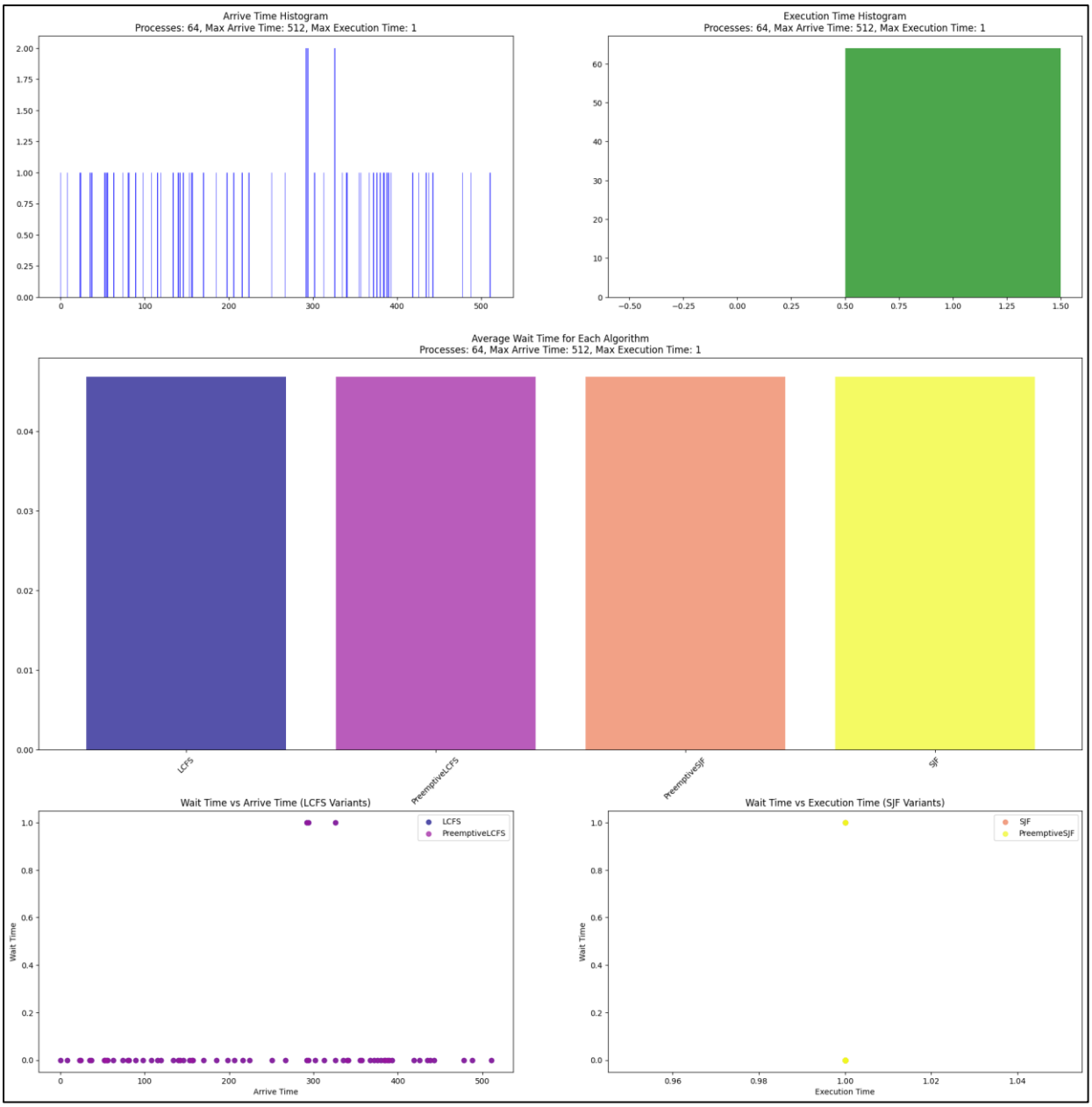

Arrive Time Histogram
Processes: 64, Max Arrive Time: 0, Max Execution Time: 16
Execution Time Histogram
Processes: 64, Max Arrive Time: 0, Max Execution Time: 16
Average Wait Time for Each Algorithm
Processes: 64, Max Arrive Time: 0, Max Execution Time: 16
LCFS
PreemptiveLCFS
PreemptiveSJF
SJF
Wait Time vs Arrive Time (LCFS Variants)
LCFS
PreemptiveLCFS
Wait Time
Arrive Time
Wait Time vs Execution Time (SJF Variants)
SJF
PreemptiveSJF
Wait Time
Execution Time


Arrive Time Histogram
Processes: 64, Max Arrive Time: 0, Max Execution Time: 32
Execution Time Histogram
Processes: 64, Max Arrive Time: 0, Max Execution Time: 32
Average Wait Time for Each Algorithm
Processes: 64, Max Arrive Time: 0, Max Execution Time: 32
LCFS
PreemptiveLCFS
PreemptiveSJF
SJF
Wait Time vs Arrive Time (LCFS Variants)
LCFS
PreemptiveLCFS
Wait Time
Arrive Time
Wait Time vs Execution Time (SJF Variants)
SJF
PreemptiveSJF
Wait Time
Execution Time

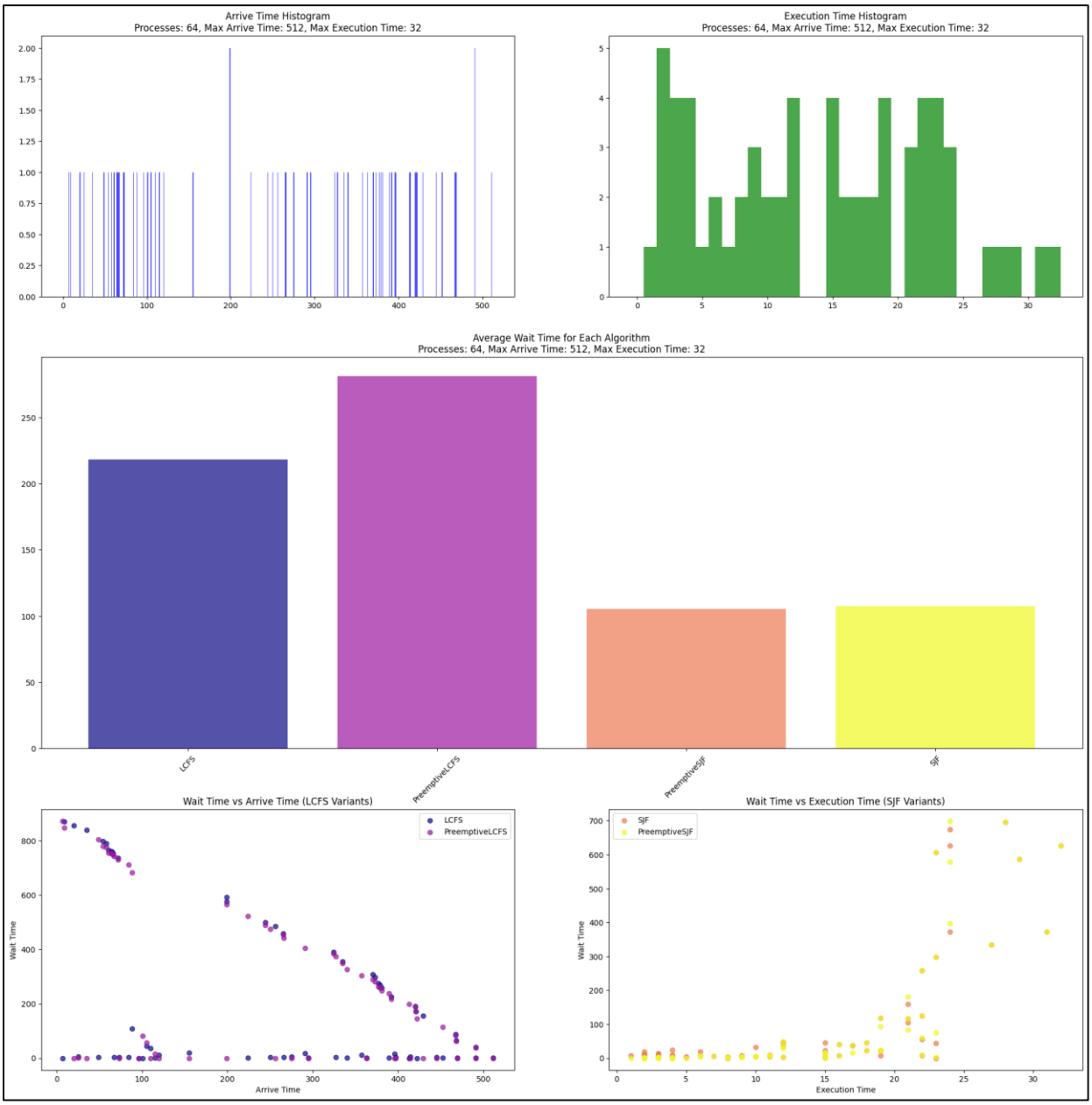

Arrive Time Histogram
Processes: 64, Max Arrive Time: 128, Max Execution Time: 1
Execution Time Histogram
Processes: 64, Max Arrive Time: 128, Max Execution Time: 1
Average Wait Time for Each Algorithm
Processes: 64, Max Arrive Time: 128, Max Execution Time: 1
LCFS
PreemptiveLCFS
PreemptiveSjf
Sjf
Wait Time vs Arrive Time (LCFS Variants)
LCFS
PreemptiveLCFS
Wait Time
Arrive Time
Wait Time vs Execution Time (SJF Variants)
SJF
PreemptiveSJF
Wait Time
Execution Time


Arrive Time Histogram
Processes: 64, Max Arrive Time: 128, Max Execution Time: 16
Execution Time Histogram
Processes: 64, Max Arrive Time: 128, Max Execution Time: 16
Average Wait Time for Each Algorithm
Processes: 64, Max Arrive Time: 128, Max Execution Time: 16
LCFS
PreemptiveLCFS
PreemptiveSJF
SJF
Wait Time vs Arrive Time (LCFS Variants)
LCFS
PreemptiveLCFS
Wait Time
Arrive Time
Wait Time vs Execution Time (SJF Variants)
SJF
PreemptiveSJF
Wait Time
Execution Time

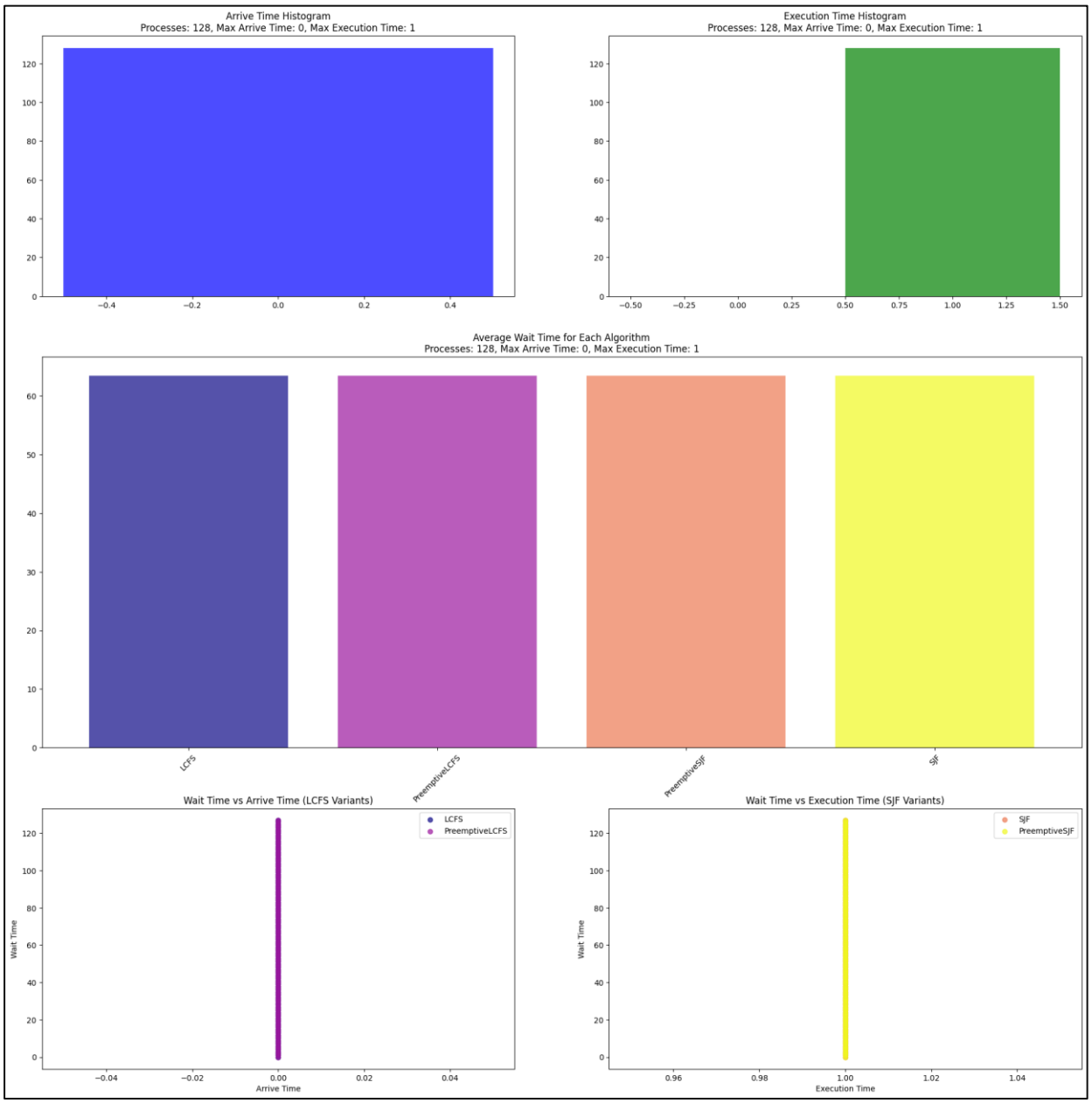

Arrive Time Histogram
Processes: 64, Max Arrive Time: 128, Max Execution Time: 32
Execution Time Histogram
Processes: 64, Max Arrive Time: 128, Max Execution Time: 32
Average Wait Time for Each Algorithm
Processes: 64, Max Arrive Time: 128, Max Execution Time: 32
LCFS
PreemptiveLCFS
PreemptiveSJF
SJF
Wait Time vs Arrive Time (LCFS Variants)
LCFS
PreemptiveLCFS
Wait Time
Arrive Time
Wait Time vs Execution Time (SJF Variants)
SJF
PreemptiveSJF
Wait Time
Execution Time

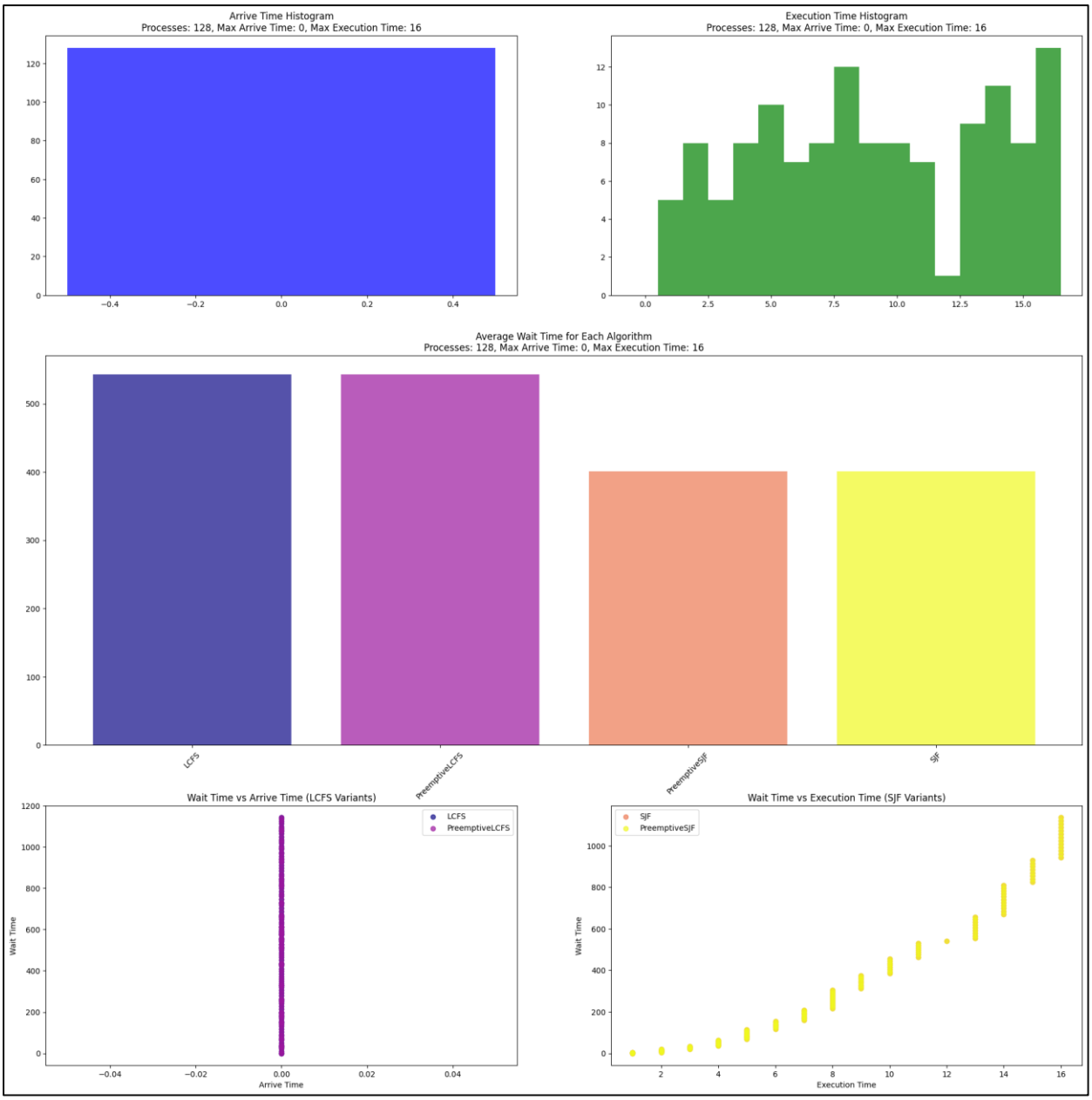

Arrive Time Histogram
Processes: 64, Max Arrive Time: 512, Max Execution Time: 1
Execution Time Histogram
Processes: 64, Max Arrive Time: 512, Max Execution Time: 1
Average Wait Time for Each Algorithm
Processes: 64, Max Arrive Time: 512, Max Execution Time: 1
LCFS
PreemptiveLCFS
PreemptiveSJF
SJF
Wait Time vs Arrive Time (LCFS Variants)
Wait Time
Arrive Time
Wait Time vs Execution Time (SJF Variants)
Execution Time

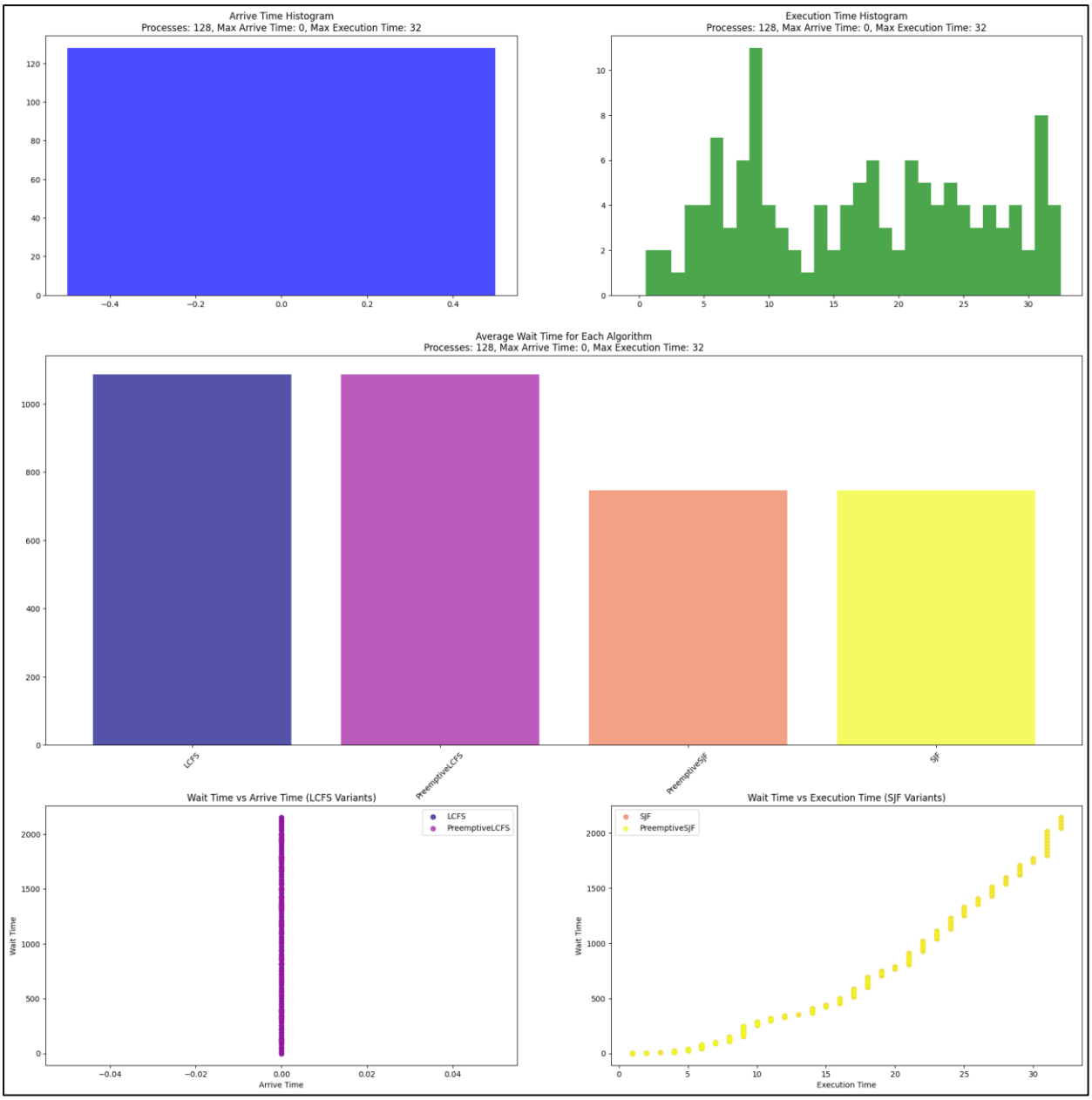

Arrive Time Histogram
Processes: 64, Max Arrive Time: 512, Max Execution Time: 16
Execution Time Histogram
Processes: 64, Max Arrive Time: 512, Max Execution Time: 16
Average Wait Time for Each Algorithm
Processes: 64, Max Arrive Time: 512, Max Execution Time: 16
LCFS
PreemptiveLCFS
PreemptiveSJF
SJF
Wait Time vs Arrive Time (LCFS Variants)
LCFS
PreemptiveLCFS
Wait Time
Arrive Time
Wait Time vs Execution Time (SJF Variants)
SJF
PreemptiveSJF
Wait Time
Execution Time

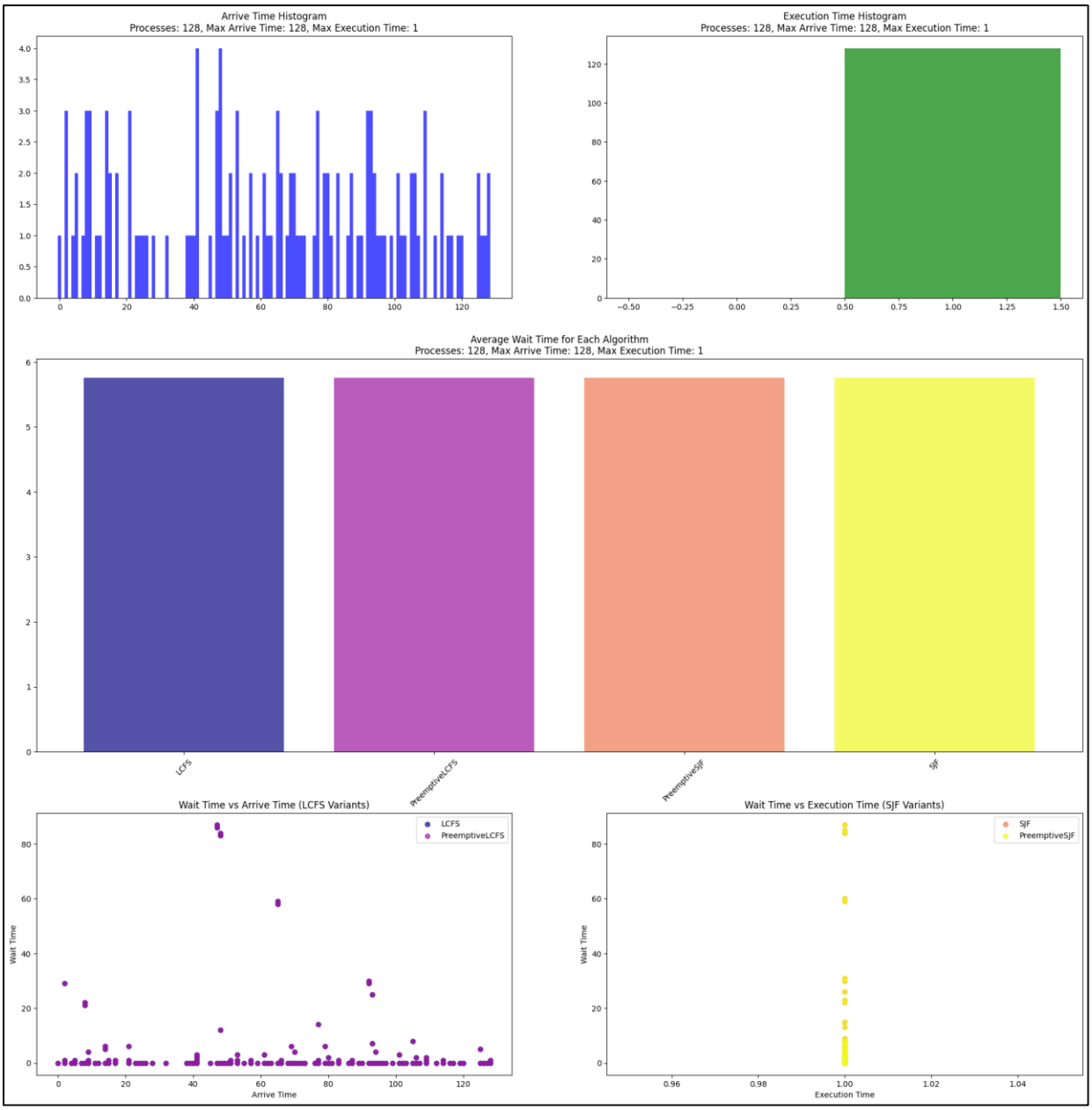

Arrive Time Histogram
Processes: 64, Max Arrive Time: 512, Max Execution Time: 32
Execution Time Histogram
Processes: 64, Max Arrive Time: 512, Max Execution Time: 32
Average Wait Time for Each Algorithm
Processes: 64, Max Arrive Time: 512, Max Execution Time: 32
LCFS
PreemptiveLCFS
PreemptiveSJF
SJF
Wait Time vs Arrive Time (LCFS Variants)
LCFS
PreemptiveLCFS
Wait Time
Arrive Time
Wait Time vs Execution Time (SJF Variants)
SJF
PreemptiveSJF
Wait Time
Execution Time

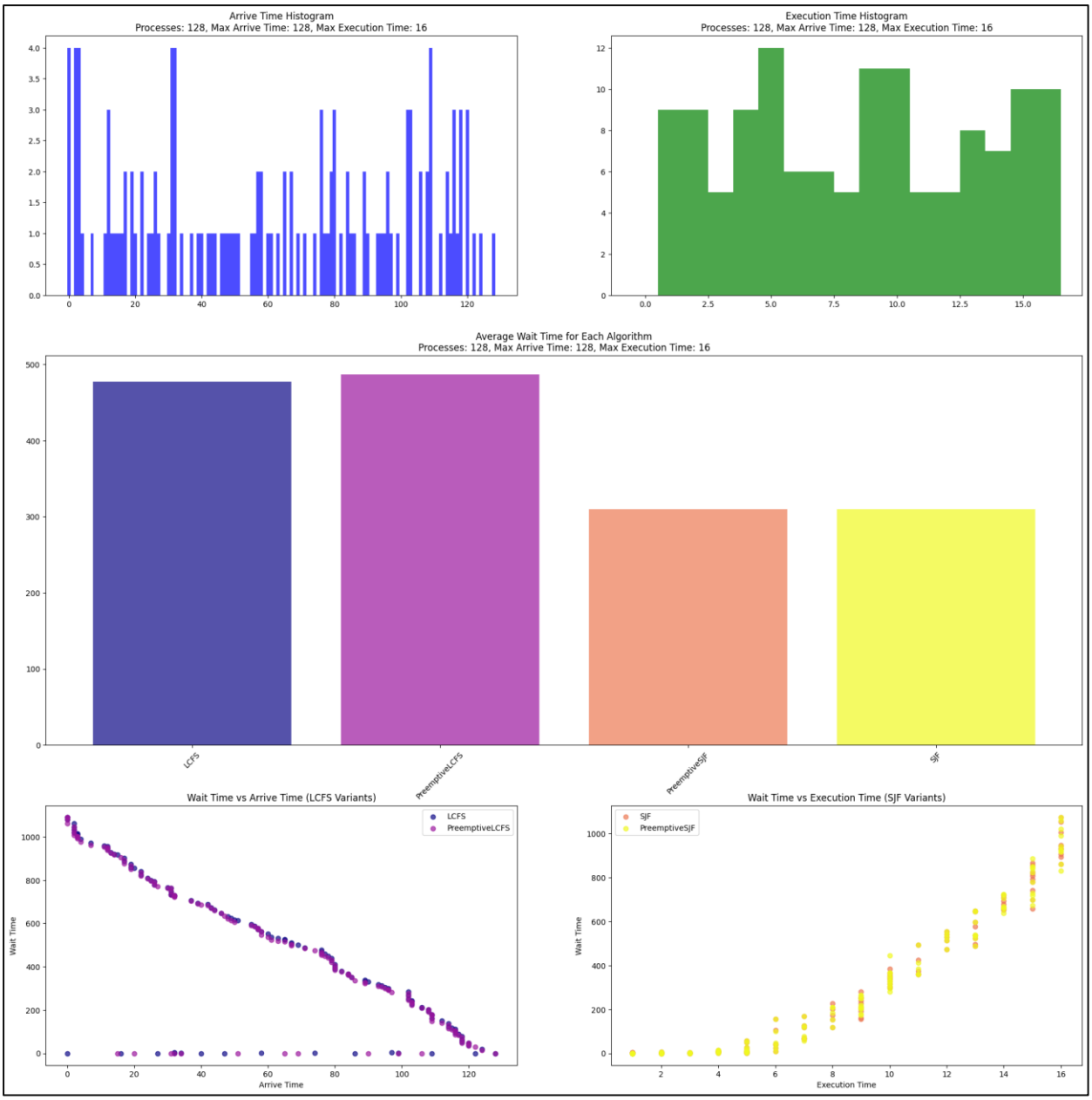
- *128 procesów:*

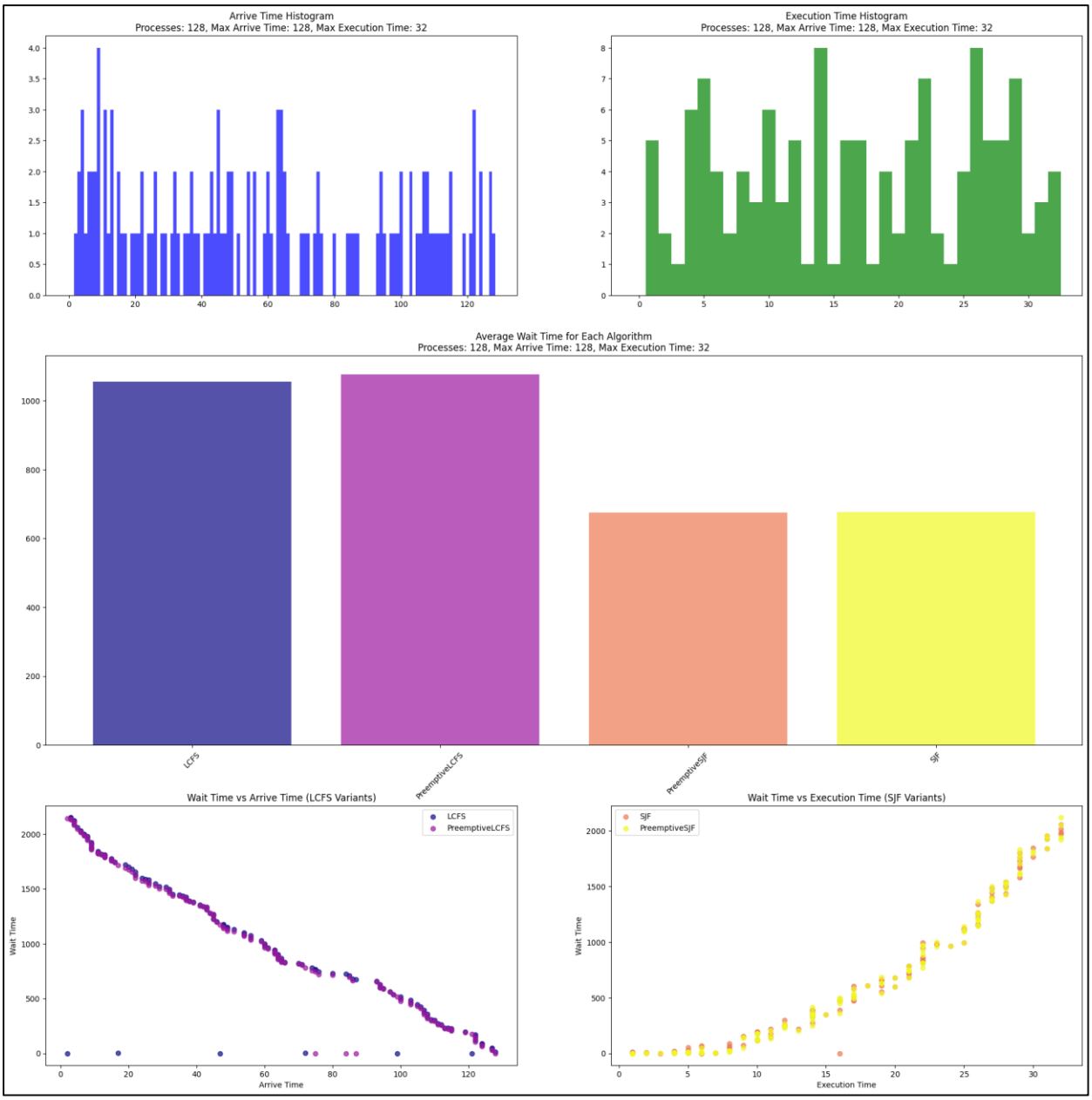


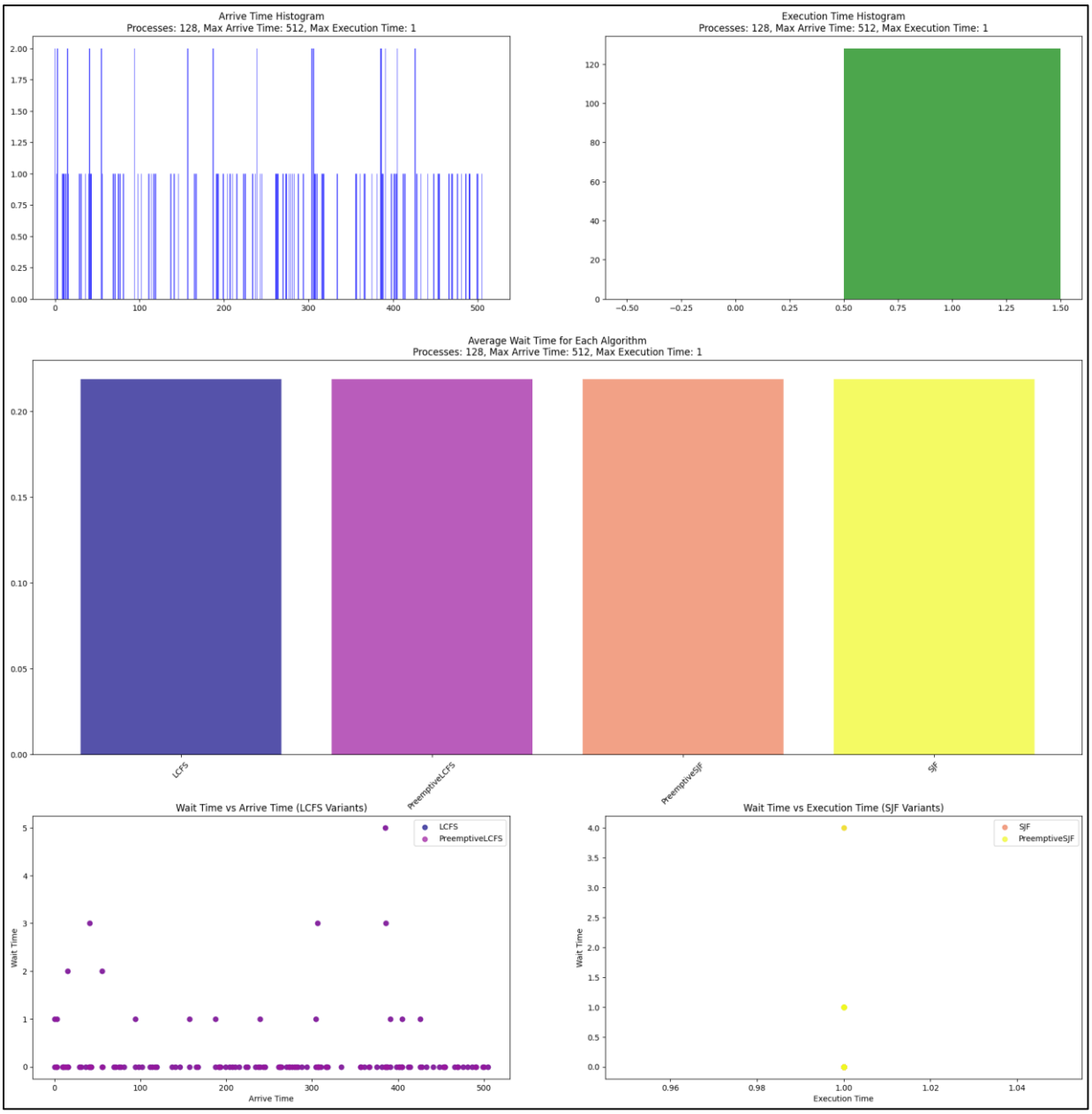

Arrive Time Histogram
Processes: 128, Max Arrive Time: 0, Max Execution Time: 16
Execution Time Histogram
Processes: 128, Max Arrive Time: 0, Max Execution Time: 16
Average Wait Time for Each Algorithm
Processes: 128, Max Arrive Time: 0, Max Execution Time: 16
LCFS
PreemptiveLCFS
PreemptiveSJF
SJF
Wait Time vs Arrive Time (LCFS Variants)
LCFS
PreemptiveLCFS
Wait Time
Arrive Time
Wait Time vs Execution Time (SJF Variants)
SJF
PreemptiveSJF
Wait Time
Execution Time

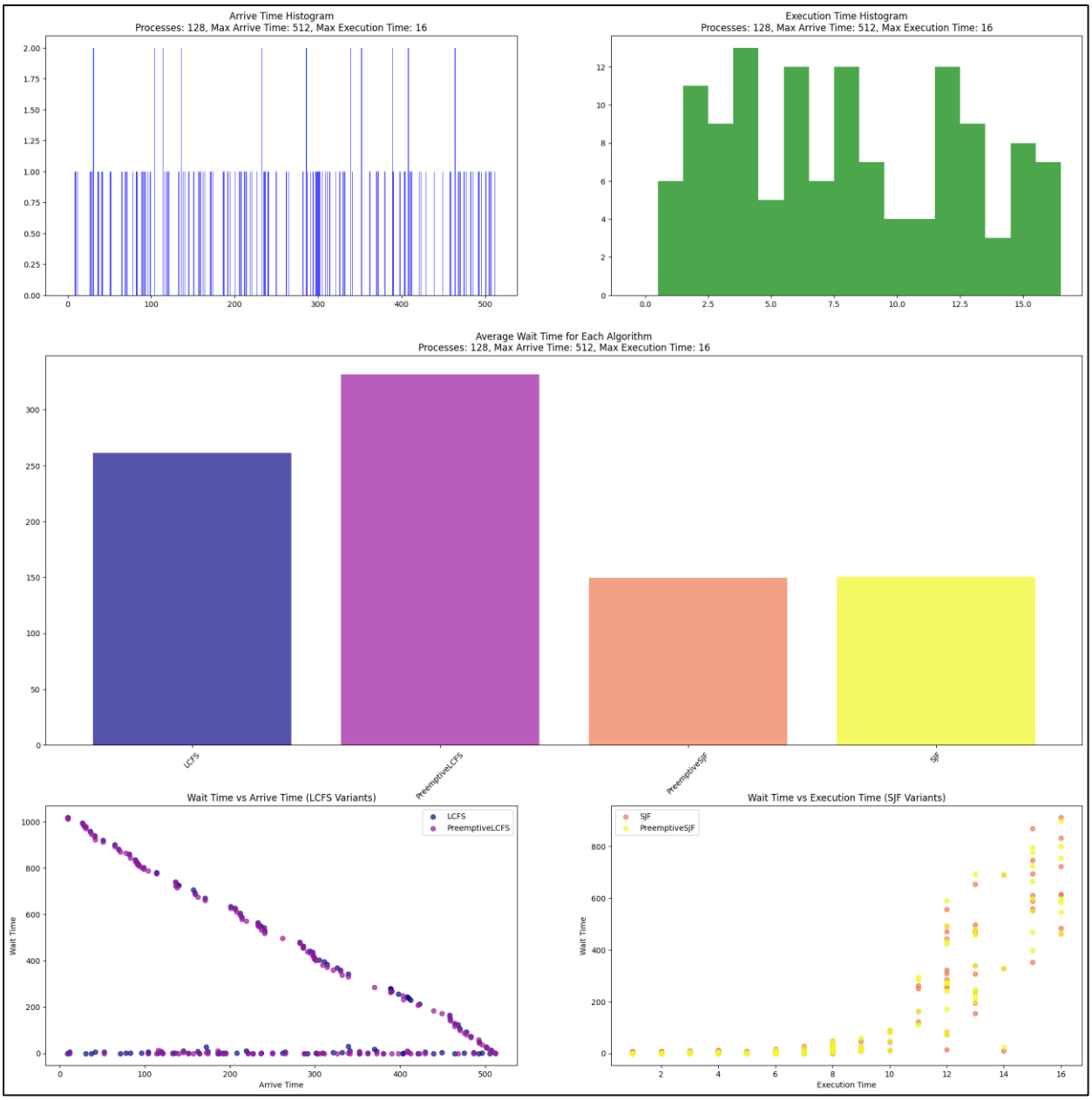

Arrive Time Histogram
Processes: 128, Max Arrive Time: 0, Max Execution Time: 32
Execution Time Histogram
Processes: 128, Max Arrive Time: 0, Max Execution Time: 32
Average Wait Time for Each Algorithm
Processes: 128, Max Arrive Time: 0, Max Execution Time: 32
LCFS
PreemptiveLCFS
PreemptiveSJF
SJF
Wait Time vs Arrive Time (LCFS Variants)
LCFS
PreemptiveLCFS
Wait Time
Arrive Time
Wait Time vs Execution Time (SJF Variants)
SJF
PreemptiveSJF
Wait Time
Execution Time

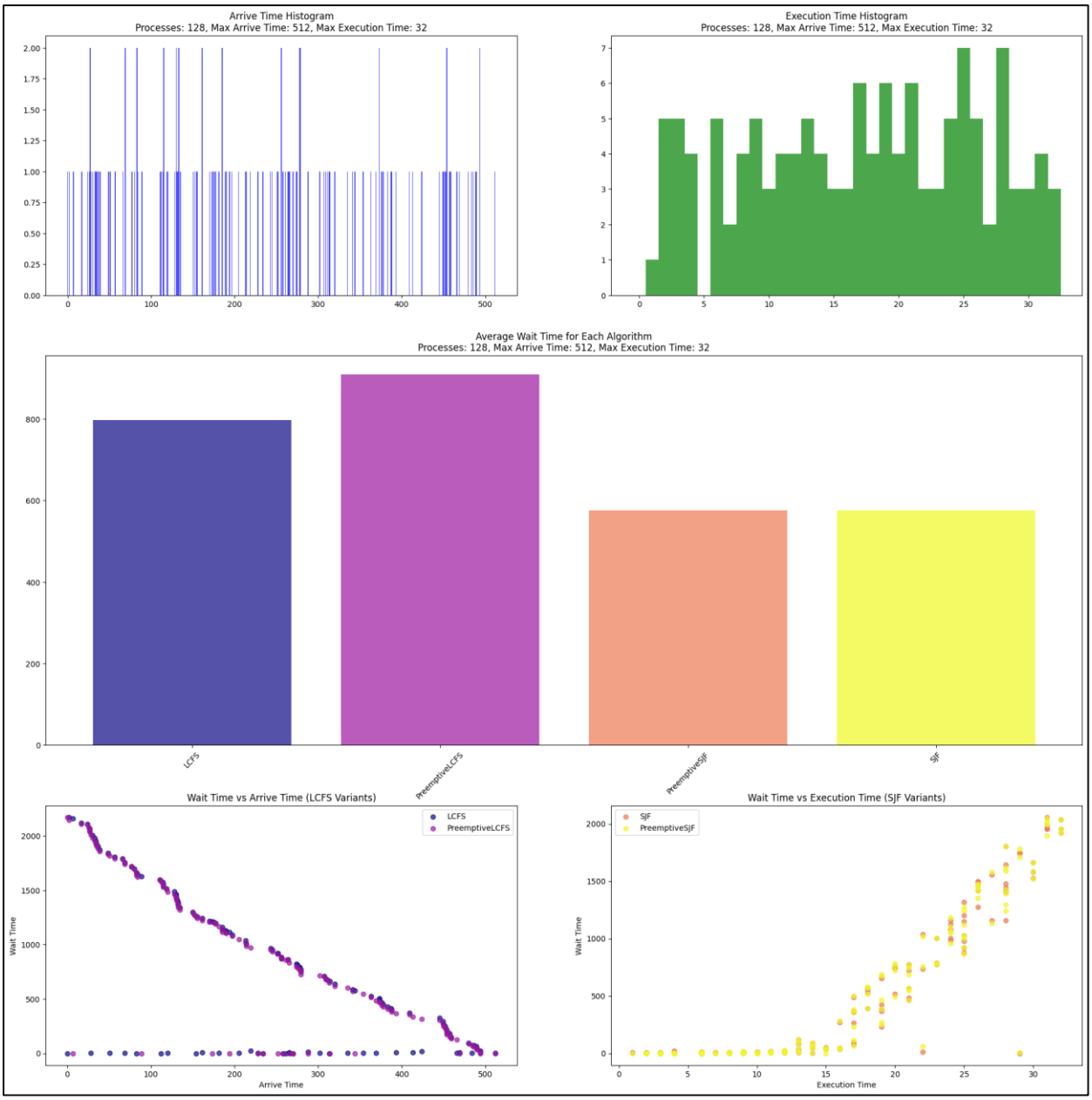

Arrive Time Histogram
Processes: 128, Max Arrive Time: 128, Max Execution Time: 1
Execution Time Histogram
Processes: 128, Max Arrive Time: 128, Max Execution Time: 1
Average Wait Time for Each Algorithm
Processes: 128, Max Arrive Time: 128, Max Execution Time: 1
LCFS
PreemptiveLCFS
PreemptiveSJF
SJF
Wait Time vs Arrive Time (LCFS Variants)
LCFS
PreemptiveLCFS
Wait Time
Arrive Time
Wait Time vs Execution Time (SJF Variants)
SJF
PreemptiveSJF
Wait Time
Execution Time


Arrive Time Histogram
Processes: 128, Max Arrive Time: 128, Max Execution Time: 16
Execution Time Histogram
Processes: 128, Max Arrive Time: 128, Max Execution Time: 16
Average Wait Time for Each Algorithm
Processes: 128, Max Arrive Time: 128, Max Execution Time: 16
LCFS
PreemptiveLCFS
PreemptiveSJF
SJF
Wait Time vs Arrive Time (LCFS Variants)
LCFS
PreemptiveLCFS
Wait Time
Arrive Time
Wait Time vs Execution Time (SJF Variants)
SJF
PreemptiveSJF
Wait Time
Execution Time

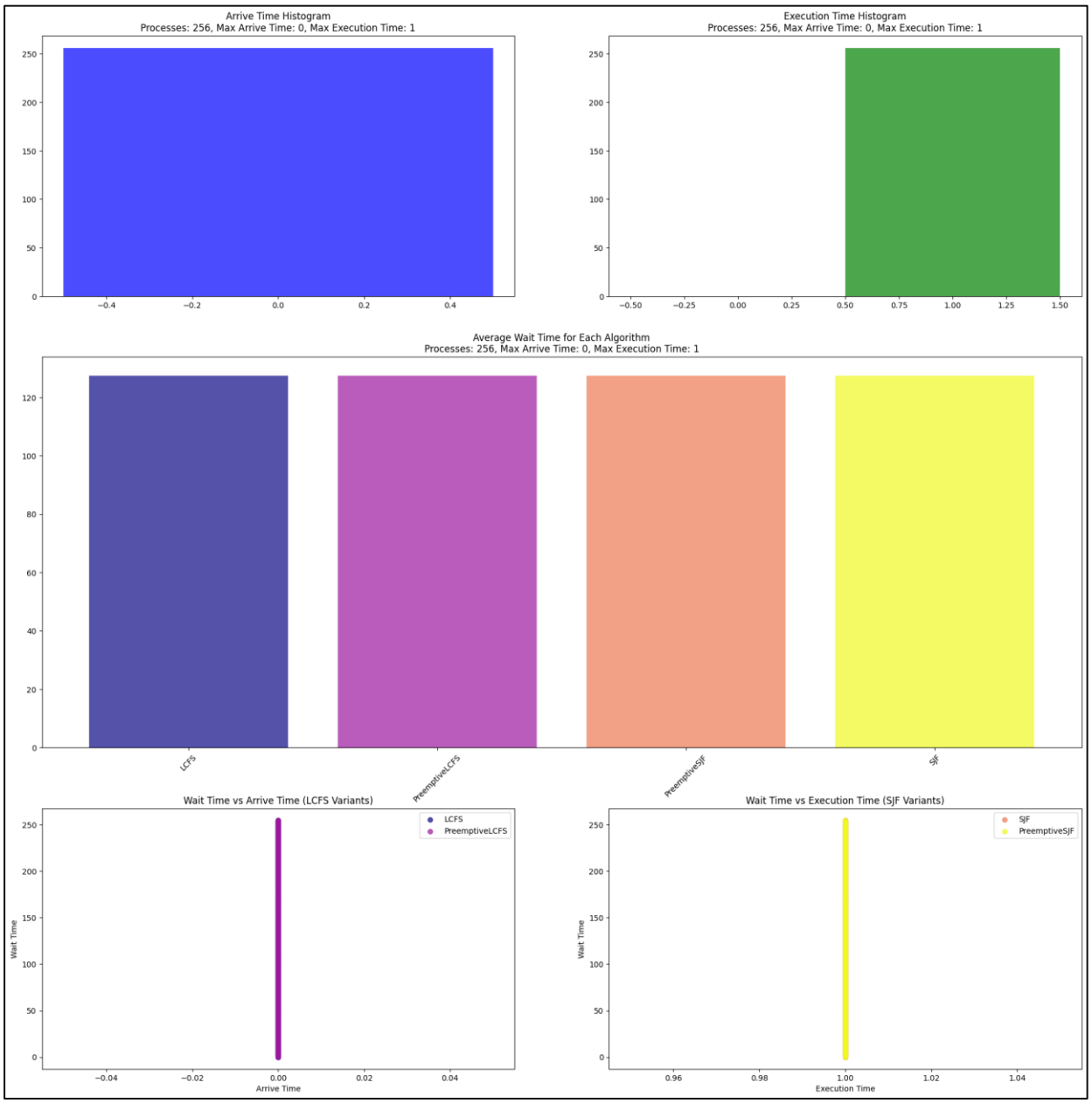

Arrive Time Histogram
Processes: 128, Max Arrive Time: 128, Max Execution Time: 32
Execution Time Histogram
Processes: 128, Max Arrive Time: 128, Max Execution Time: 32
Average Wait Time for Each Algorithm
Processes: 128, Max Arrive Time: 128, Max Execution Time: 32
LCFS
PreemptiveLCFS
PreemptiveSJF
SJF
Wait Time vs Arrive Time (LCFS Variants)
LCFS
PreemptiveLCFS
Wait Time
Arrive Time
Wait Time vs Execution Time (SJF Variants)
SJF
PreemptiveSJF
Wait Time
Execution Time

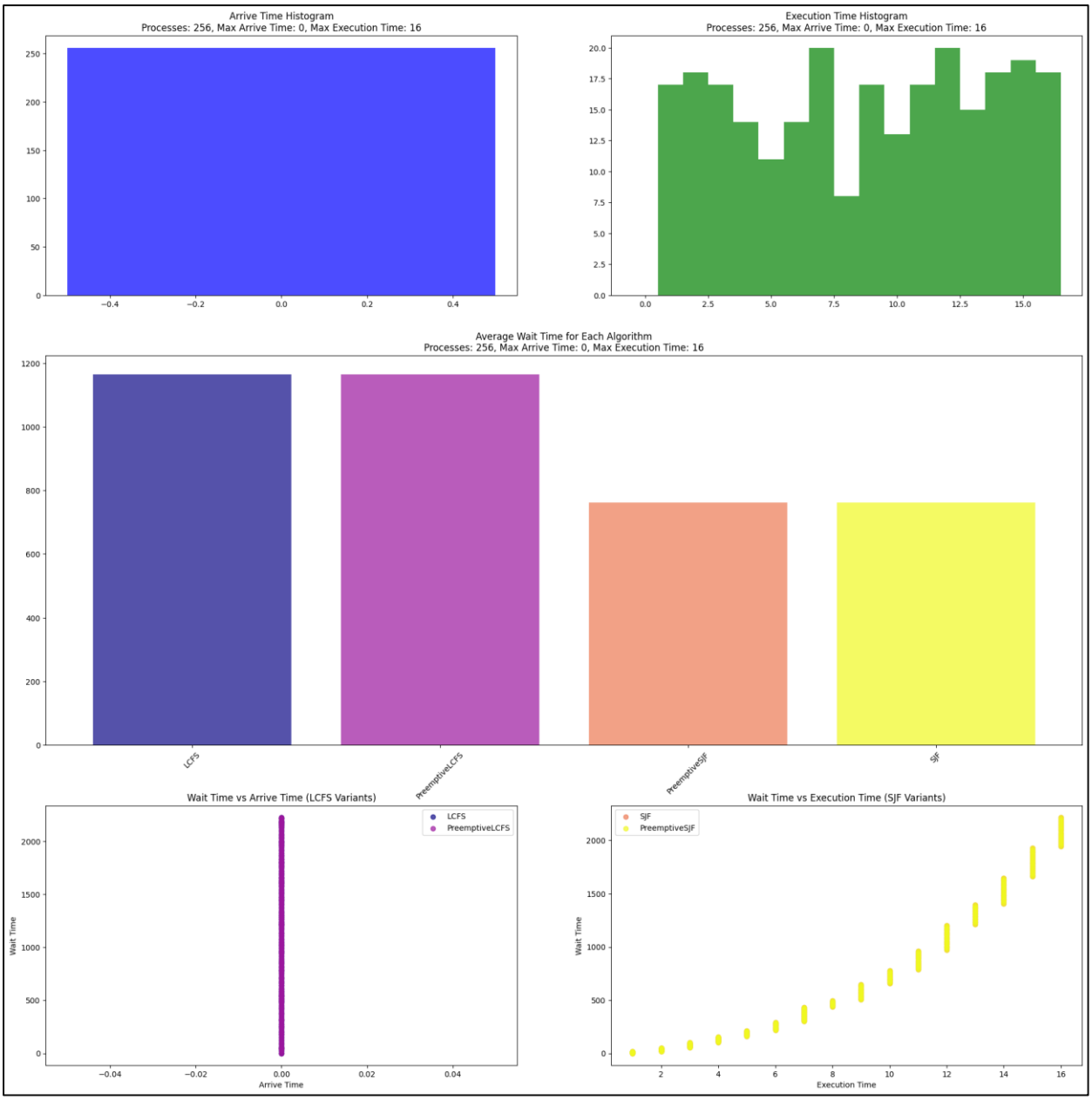

Arrive Time Histogram
Processes: 128, Max Arrive Time: 512, Max Execution Time: 1
Execution Time Histogram
Processes: 128, Max Arrive Time: 512, Max Execution Time: 1
Average Wait Time for Each Algorithm
Processes: 128, Max Arrive Time: 512, Max Execution Time: 1
LCFS
PreemptiveLCFS
PreemptiveSJF
SJF
Wait Time vs Arrive Time (LCFS Variants)
Wait Time
Arrive Time
Wait Time vs Execution Time (SJF Variants)
Wait Time
Execution Time

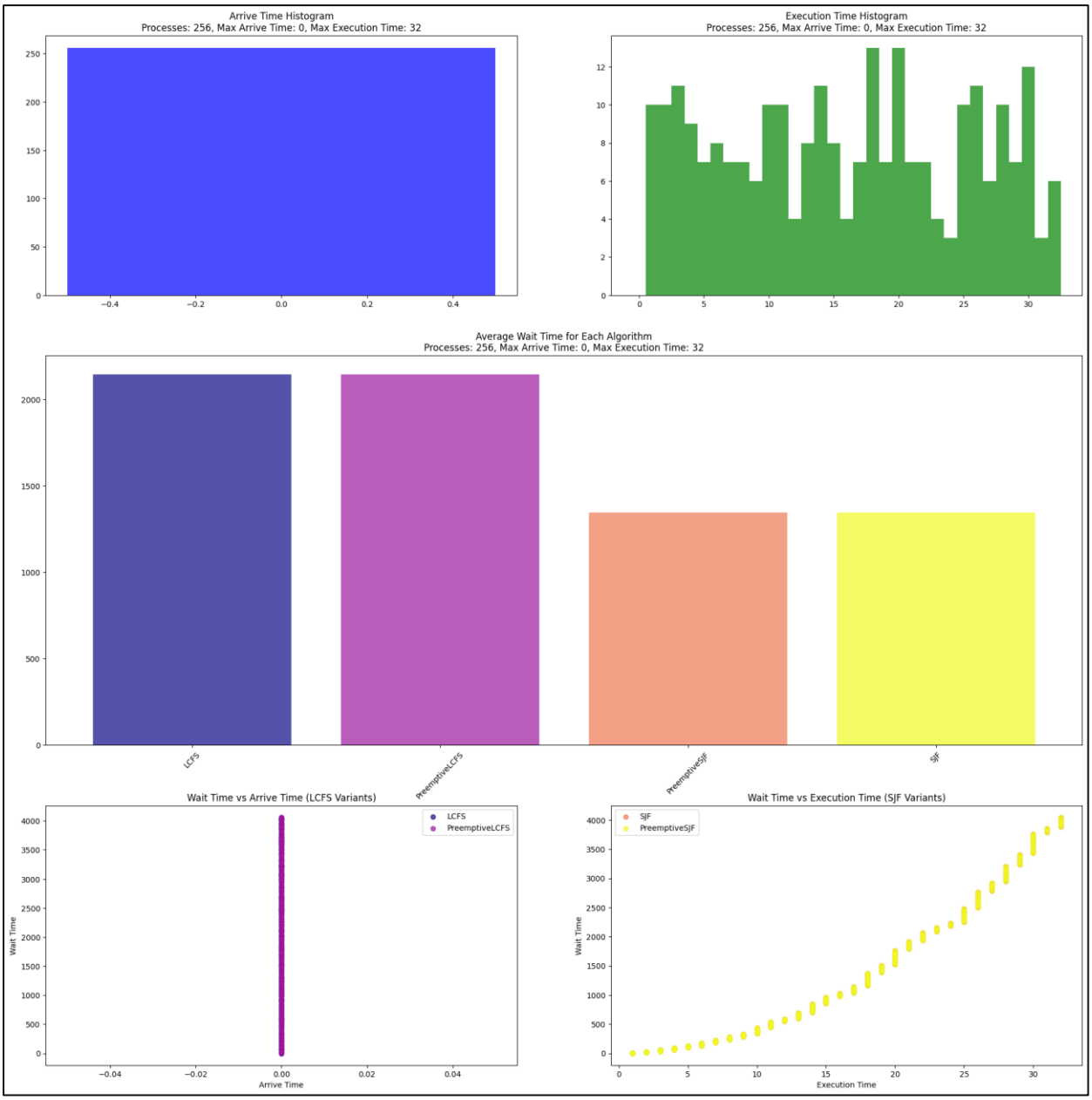

Arrive Time Histogram
Processes: 128, Max Arrive Time: 512, Max Execution Time: 16
Execution Time Histogram
Processes: 128, Max Arrive Time: 512, Max Execution Time: 16
Average Wait Time for Each Algorithm
Processes: 128, Max Arrive Time: 512, Max Execution Time: 16
LCFS
PreemptiveLCFS
PreemptiveSJF
SJF
Wait Time vs Arrive Time (LCFS Variants)
LCFS
PreemptiveLCFS
Wait Time
Arrive Time
Wait Time vs Execution Time (SJF Variants)
SJF
PreemptiveSJF
Wait Time
Execution Time

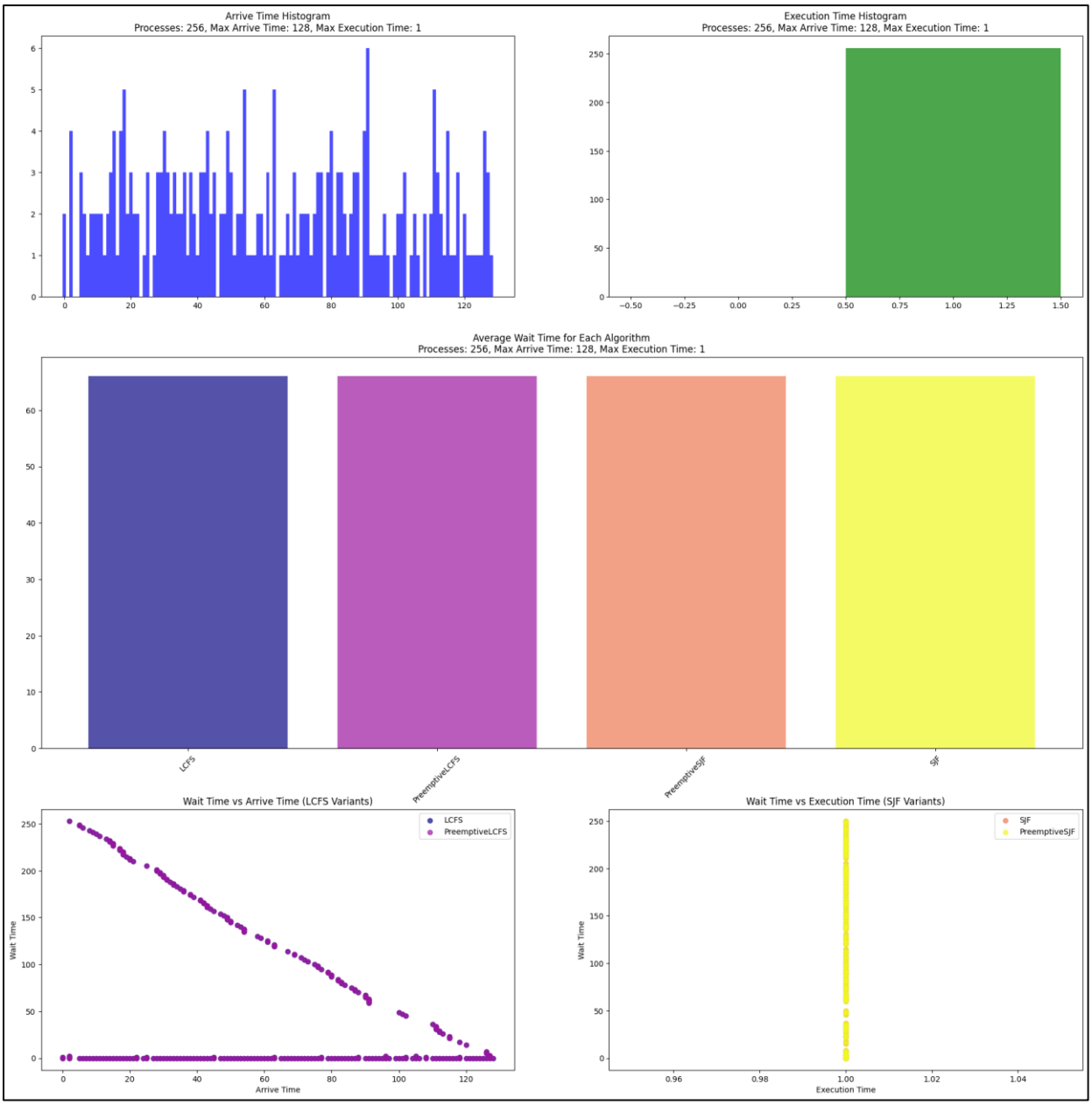

Arrive Time Histogram
Processes: 128, Max Arrive Time: 512, Max Execution Time: 32
Execution Time Histogram
Processes: 128, Max Arrive Time: 512, Max Execution Time: 32
Average Wait Time for Each Algorithm
Processes: 128, Max Arrive Time: 512, Max Execution Time: 32
LCFS
PreemptiveLCFS
PreemptiveSJF
SJF
Wait Time vs Arrive Time (LCFS Variants)
LCFS
PreemptiveLCFS
Wait Time
Arrive Time
Wait Time vs Execution Time (SJF Variants)
SJF
PreemptiveSJF
Wait Time
Execution Time

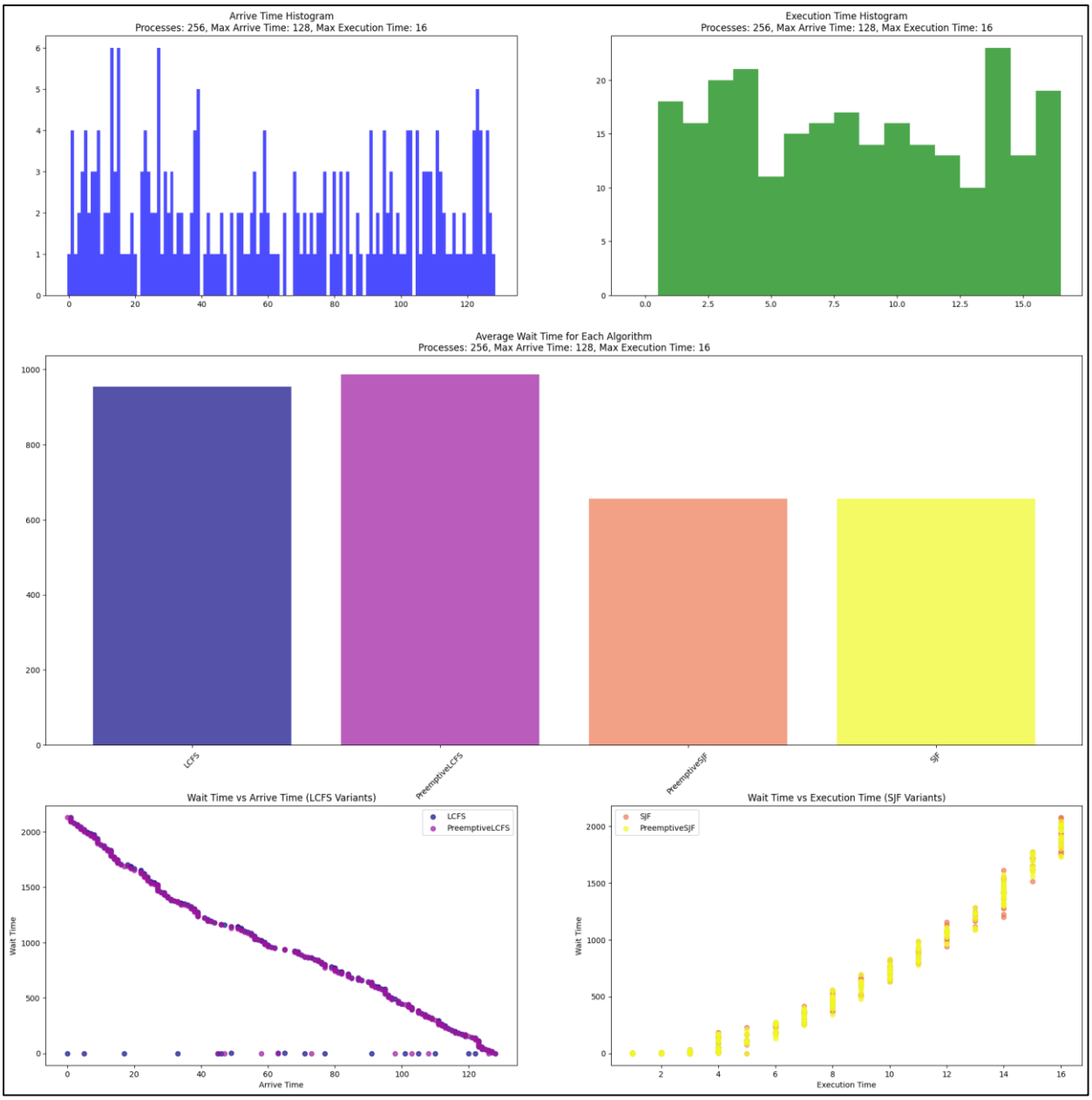
- *256 procesów:*

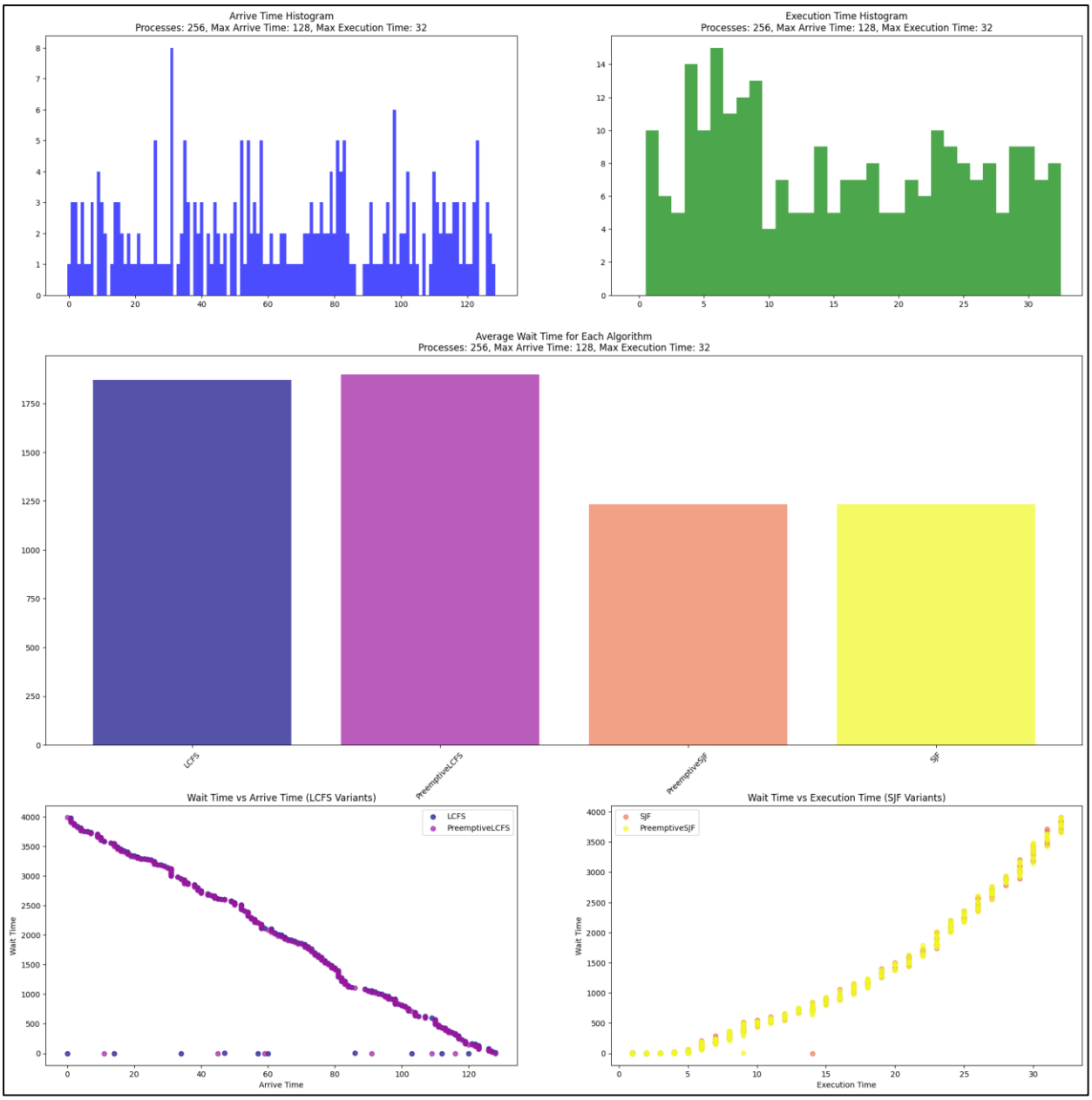


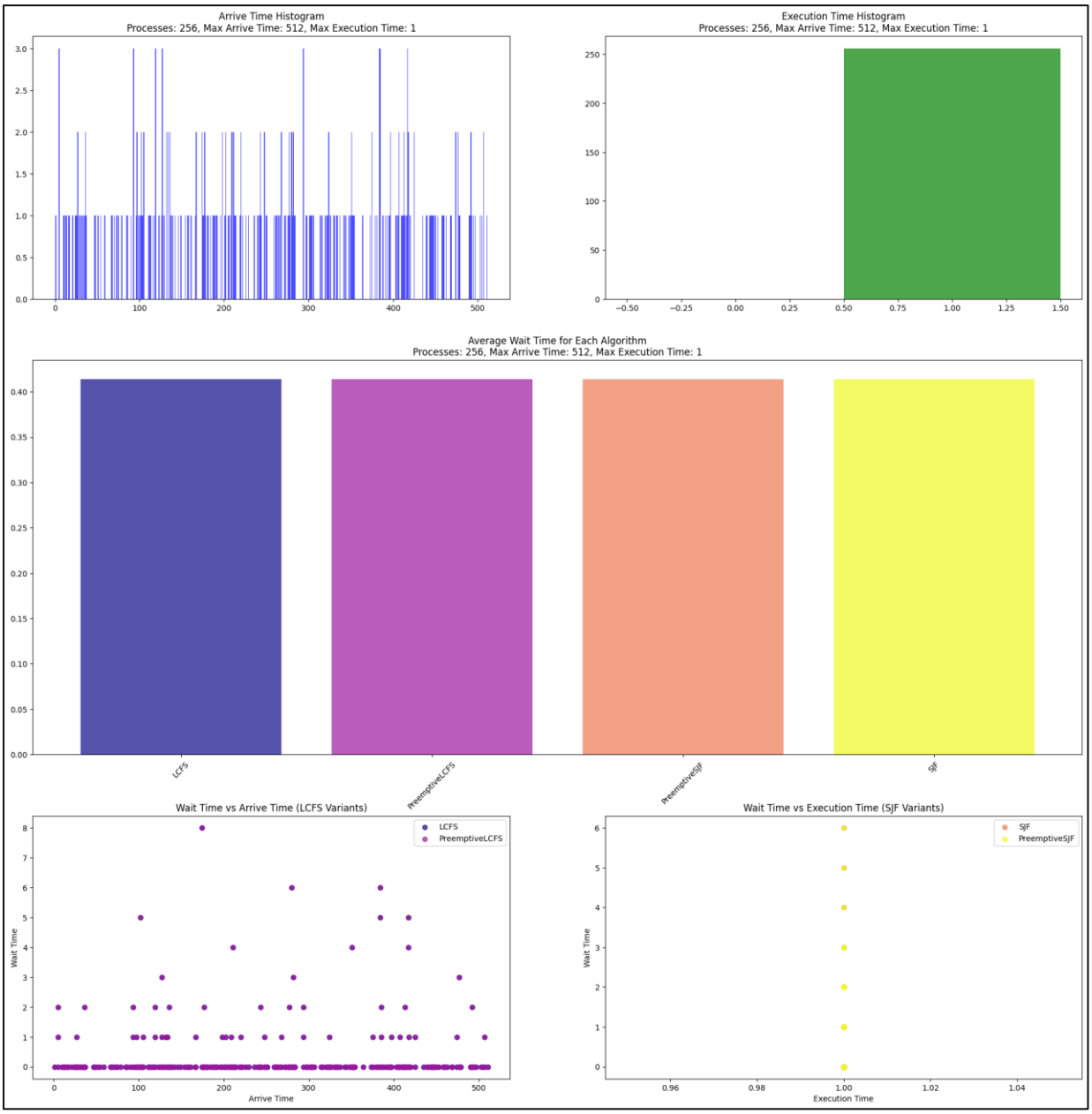

Arrive Time Histogram
Processes: 256, Max Arrive Time: 0, Max Execution Time: 16
Execution Time Histogram
Processes: 256, Max Arrive Time: 0, Max Execution Time: 16
Average Wait Time for Each Algorithm
Processes: 256, Max Arrive Time: 0, Max Execution Time: 16
LCFS
PreemptiveLCFS
PreemptiveSJF
SJF
Wait Time vs Arrive Time (LCFS Variants)
LCFS
PreemptiveLCFS
Wait Time
Arrive Time
Wait Time vs Execution Time (SJF Variants)
SJF
PreemptiveSJF
Wait Time
Execution Time

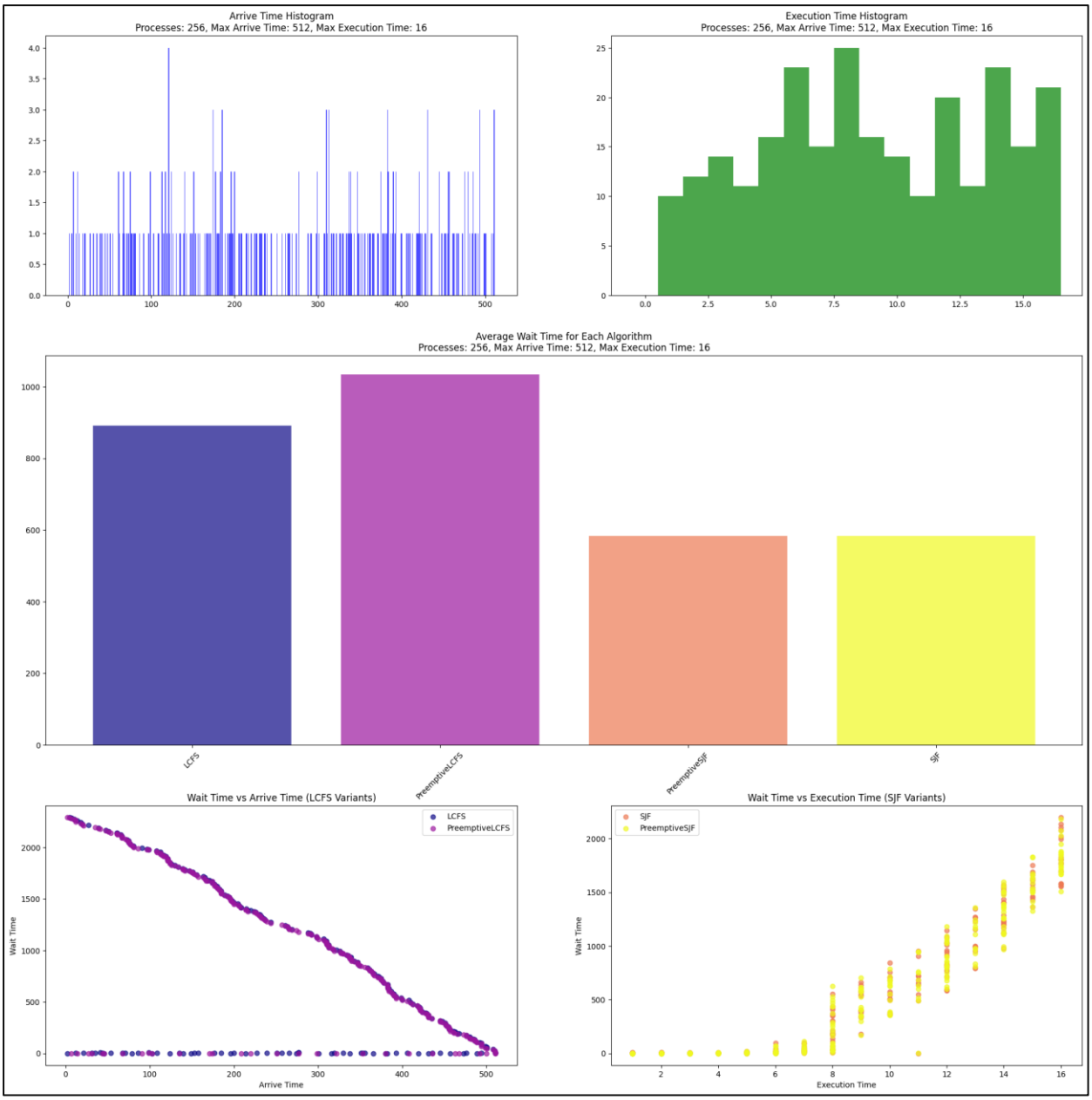

Arrive Time Histogram
Processes: 256, Max Arrive Time: 0, Max Execution Time: 32
Execution Time Histogram
Processes: 256, Max Arrive Time: 0, Max Execution Time: 32
Average Wait Time for Each Algorithm
Processes: 256, Max Arrive Time: 0, Max Execution Time: 32
LCFS
PreemptiveLCFS
PreemptiveSJF
SJF
Wait Time vs Arrive Time (LCFS Variants)
LCFS
PreemptiveLCFS
Wait Time
Arrive Time
Wait Time vs Execution Time (SJF Variants)
SJF
PreemptiveSJF
Wait Time
Execution Time

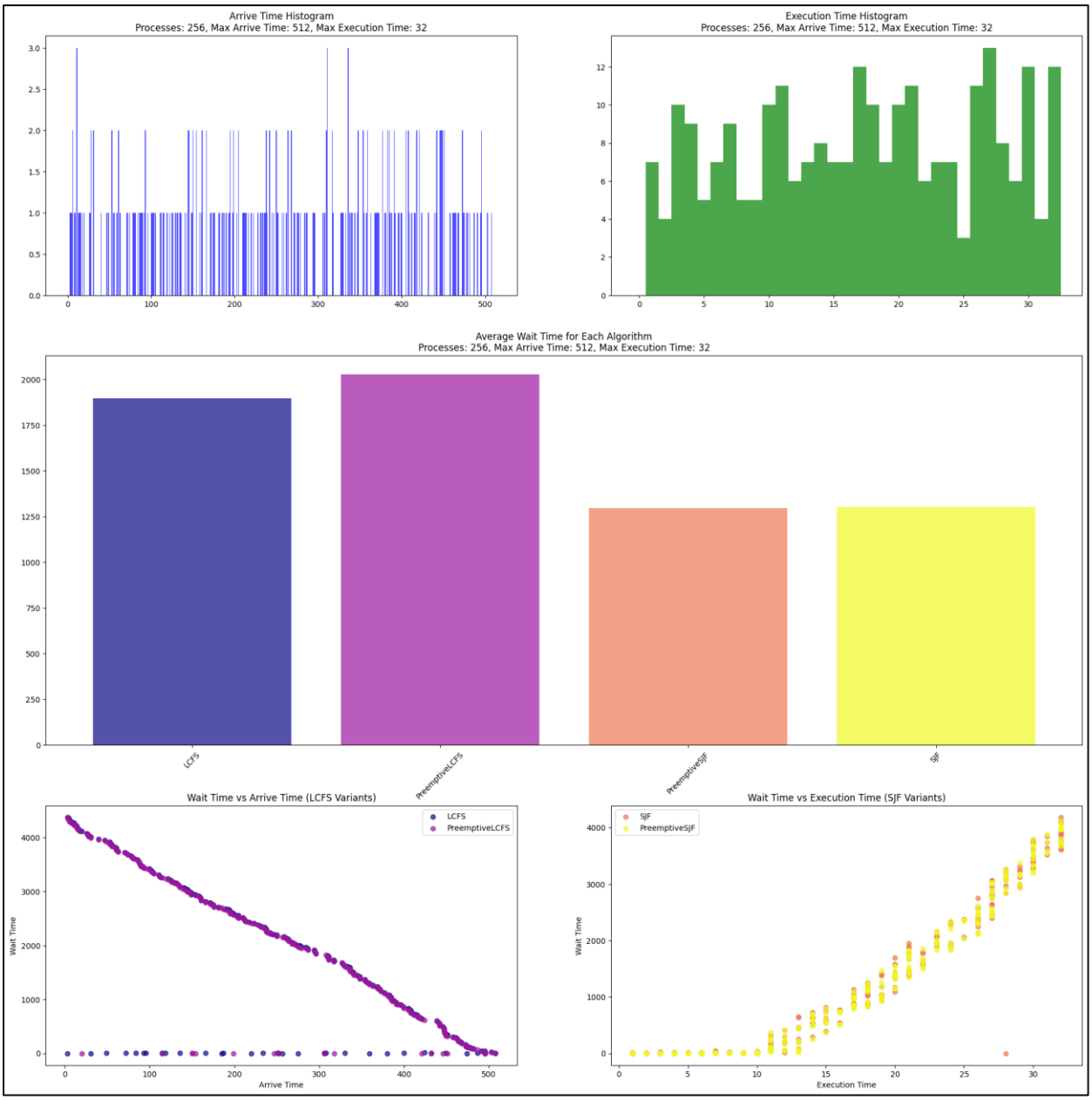

Arrive Time Histogram
Processes: 256, Max Arrive Time: 128, Max Execution Time: 1
Execution Time Histogram
Processes: 256, Max Arrive Time: 128, Max Execution Time: 1
Average Wait Time for Each Algorithm
Processes: 256, Max Arrive Time: 128, Max Execution Time: 1
LCFS
PreemptiveLCFS
PreemptiveSJF
SJF
Wait Time vs Arrive Time (LCFS Variants)
LCFS
PreemptiveLCFS
Wait Time
Arrive Time
Wait Time vs Execution Time (SJF Variants)
SJF
PreemptiveSJF
Wait Time
Execution Time


Arrive Time Histogram
Processes: 256, Max Arrive Time: 128, Max Execution Time: 16
Execution Time Histogram
Processes: 256, Max Arrive Time: 128, Max Execution Time: 16
Average Wait Time for Each Algorithm
Processes: 256, Max Arrive Time: 128, Max Execution Time: 16
LCFS
PreemptiveLCFS
PreemptiveSJF
SJF
Wait Time vs Arrive Time (LCFS Variants)
LCFS
PreemptiveLCFS
Wait Time
Arrive Time
Wait Time vs Execution Time (SJF Variants)
SJF
PreemptiveSJF
Wait Time
Execution Time


Arrive Time Histogram
Processes: 256, Max Arrive Time: 128, Max Execution Time: 32
Execution Time Histogram
Processes: 256, Max Arrive Time: 128, Max Execution Time: 32
Average Wait Time for Each Algorithm
Processes: 256, Max Arrive Time: 128, Max Execution Time: 32
LCFS
PreemptiveLCFS
PreemptiveSJF
SJF
Wait Time vs Arrive Time (LCFS Variants)
LCFS
PreemptiveLCFS
Wait Time
Arrive Time
Wait Time vs Execution Time (SJF Variants)
SJF
PreemptiveSJF
Wait Time
Execution Time


Arrive Time Histogram
Processes: 256, Max Arrive Time: 512, Max Execution Time: 1
Execution Time Histogram
Processes: 256, Max Arrive Time: 512, Max Execution Time: 1
Average Wait Time for Each Algorithm
Processes: 256, Max Arrive Time: 512, Max Execution Time: 1
LCFS
PreemptiveLCFS
PreemptiveSjf
Sjf
Wait Time vs Arrive Time (LCFS Variants)
LCFS
PreemptiveLCFS
Wait Time
Arrive Time
Wait Time vs Execution Time (SJF Variants)
SJF
PreemptiveSJF
Wait Time
Execution Time


Arrive Time Histogram
Processes: 256, Max Arrive Time: 512, Max Execution Time: 16
Execution Time Histogram
Processes: 256, Max Arrive Time: 512, Max Execution Time: 16
Average Wait Time for Each Algorithm
Processes: 256, Max Arrive Time: 512, Max Execution Time: 16
LCFS
PreemptiveLCFS
PreemptiveSJF
SJF
Wait Time vs Arrive Time (LCFS Variants)
LCFS
PreemptiveLCFS
Wait Time
Arrive Time
Wait Time vs Execution Time (SJF Variants)
SJF
PreemptiveSJF
Wait Time
Execution Time



Wnioski:

Dla zestawów danych wejściowych gdzie czas przybycia wynosi zero wszystkie algorytmy działają identycznie. W sytuacjach gdy czas wykonania wszystkich procesów wynosił 1 sytuacja była analogiczna.

W pozostałych przypadkach widzimy że średni czas oczekiwania dla obydwu wariantów SJF jest znacznie niższy niż dla LCFS.

Różnica między SJF a Preemptive SJF jeżeli chodzi o średnie czasy wykonania jest bardzo niewielka, różnice można zauważyć na wykresach czasu oczekiwania jako funkcji czasu wykonania, gdzie można zauważyć nieco krótsze czasy oczekiwania procesów o dłuższym czasie wykonania w porównaniu z wersją wywłaszczeniową.

Jeżeli chodzi o różnice między wariantami LCFS, można zauważyć że wersja wywłaszczeniowa prowadzi do wyższej średniej wartości czasu oczekiwania dla znaczącej części zestawów danych wejściowych. Na wykresie czasu oczekiwania jako funkcji czasu przybycia możemy zauważyć, że wersja wywłaszczeniowa ma znacznie mniej procesów które odbiegają od głównej linii trendu, co oznacza że procesy które przybyły wcześnie bardzo żadko zostają wykonane szybko.

SJF jest lepszym algorytmem planowania czasu procesora ze względu na znacznie niższy średni czas oczekiwania. Jeżeli zależy nam na procesach o bardzo krótkim czasie wykonania, należy rozważyć wersję wywłaszczeniową, natomiast jeżeli chcemy skierować nacisk na nieco dłuższe procesy wersja niewywłaszczeniowa również jest świetnym wyborem.

Wymiana stron

Opis algorytmów:

FIFO (First In, First Out)

Algorytm FIFO wymienia stronę, która najdłużej znajduje się w pamięci, niezależnie od jej częstotliwości użycia. Każda nowa strona jest dodawana do kolejki, a gdy pamięć jest pełna, strona na początku kolejki jest usuwana. Algorytm jest prosty i szybki w implementacji, ale nie uwzględnia wzorców dostępu, co może prowadzić do tzw. "anomalii Belady'ego". Algorytm zaimplementowano używając struktury danej kolejki, zaimplementowanej w src/sim/ds.go

LFU (Least Frequently Used)

LFU wymienia stronę, która była najmniej używana w danym okresie. Każda strona ma licznik, który zwiększa się przy każdym dostępie do niej, a strona z najniższym licznikiem jest usuwana z pamięci. Algorytm skutecznie dostosowuje się do częstotliwości dostępu, ale jego implementacja wymaga dodatkowej pamięci i czasu na aktualizację liczników. Algorytm zaimplementowano używając struktury danej min heap, zaimplementowanej w src/sim/page/page.go

LFU bez czyszczenia licznika użyć

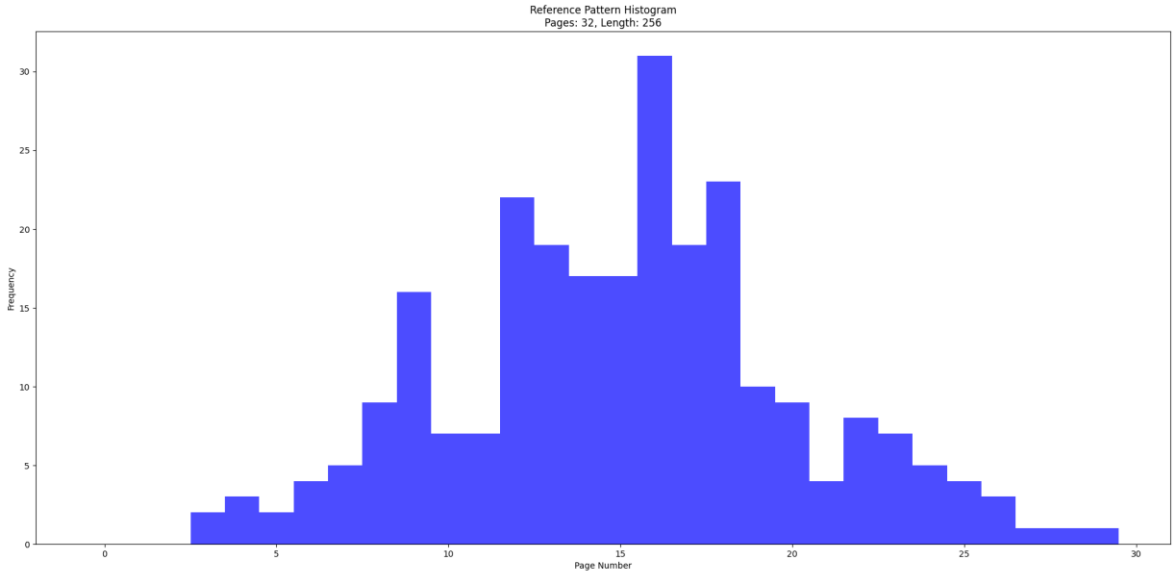
W tej wersji LFU licznik użycia strony nie jest resetowany, gdy strona zostaje przeniesiona do pamięci swap. Po ponownym załadowaniu strony do pamięci RAM, jej licznik nadal odzwierciedla historię użycia. Rozwiązanie to lepiej uwzględnia długoterminowe wzorce dostępu, ale może prowadzić do utrzymywania w pamięci stron rzadko używanych, jeśli były one intensywnie wykorzystywane w przeszłości. Algorytm zaimplementowano używając struktury danej min heap, zaimplementowanej w src/sim/page/page.go

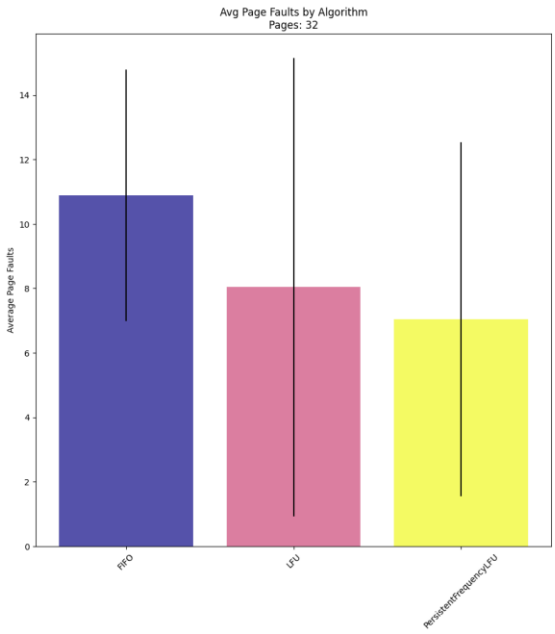
Opis procedury testowania algorytmów wymiany stron:

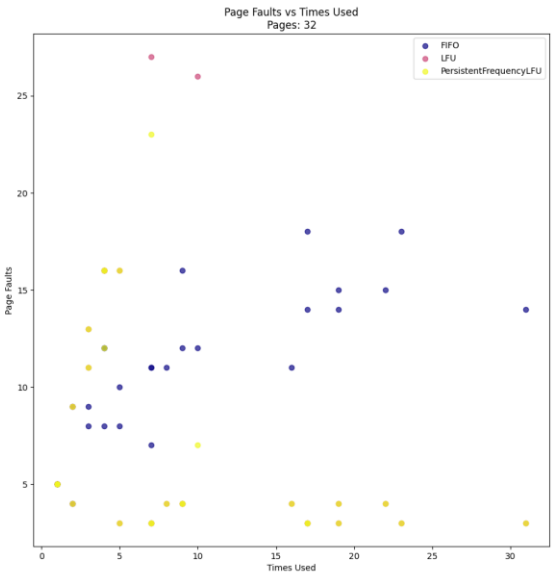
W testach generowane były sekwencje dostępów do 32, 64 i 128 stron o długościach 256, 512 i 1024. Rozmiar ramki pamięci, tzn. Ilość stron które maksymalnie mogły zmieścić się w pamięci w 1 momencie wynosił dla wszystkich testów 16. Dla każdej pozycji z sekwencji dostępów strona była wybierana z rozkładu normalnego o średniej w środku przedziału indeksów stron, (od 0 do n-1 gdzie n to ilość stron) i odchyleniu standardowym równym 1/3 średniej, co sprawia że przedział pokrywa 3 odchylenia standardowe rozkładu. Niestety tylko takie rozwiązanie było możliwe przez to, że tylko jedna funkcja generująca wartości losowe o rozkładzie normalnym znajduje się w bibliotece go std/math/randv2 i są to wartości float64, przez co trzeba było dokonać konwersji oraz obcięcia wartości.

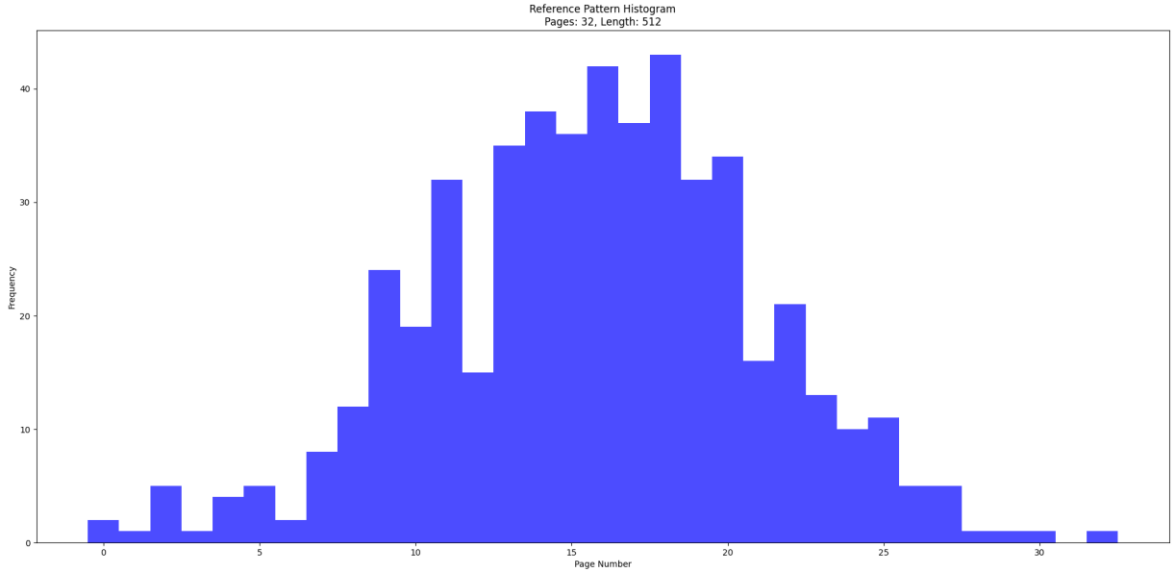
Wyniki testów algorytmów wymiany stron

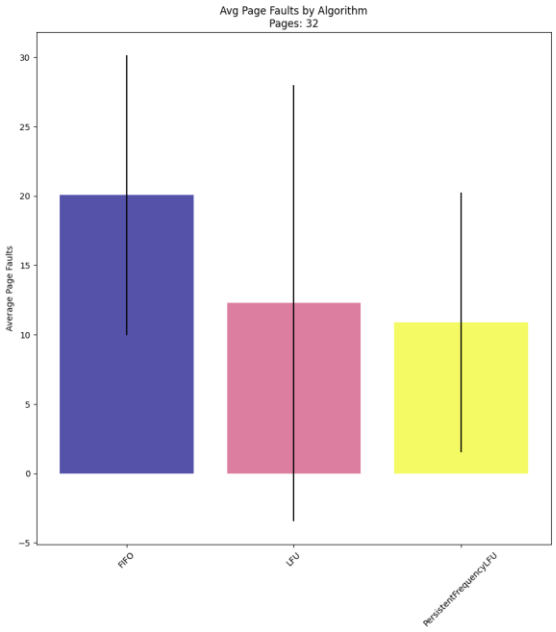
- *32 strony:*

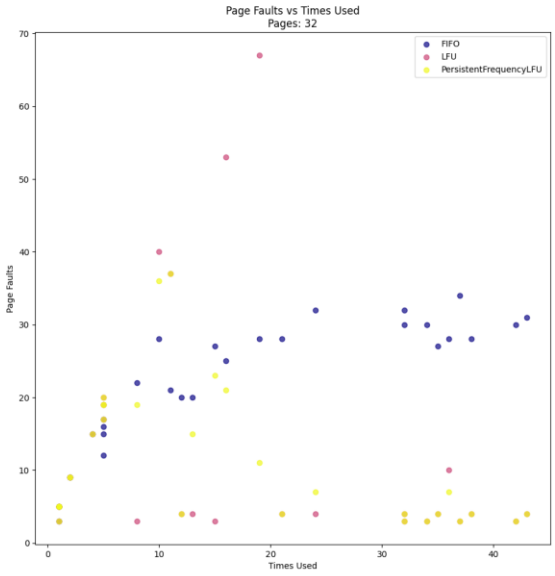


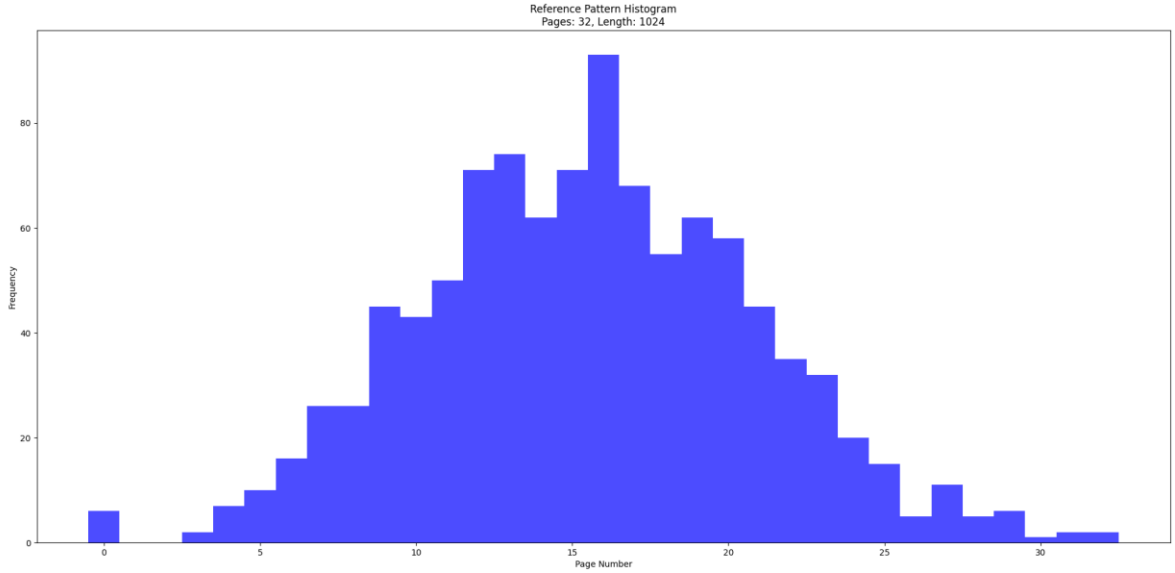


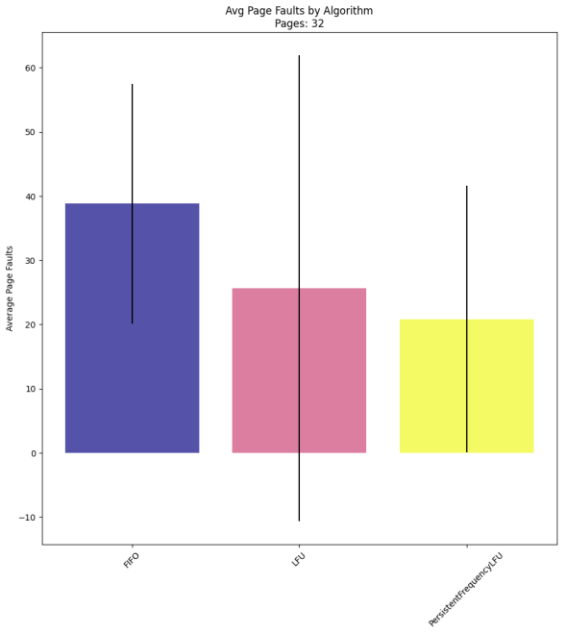


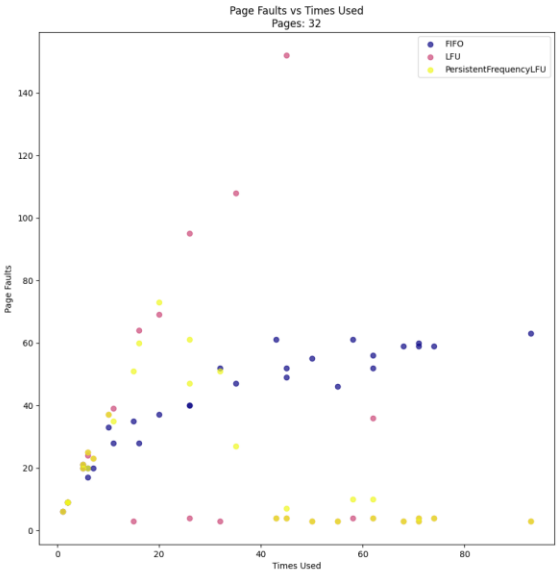

Reference Pattern Histogram
Pages: 32, Length: 512
Frequency
Page Number
0
5
10
15
20
25
30
40


Avg Page Faults by Algorithm
Pages: 32
Average Page Faults
FIFO
LFU
PersistentFrequencyLFU

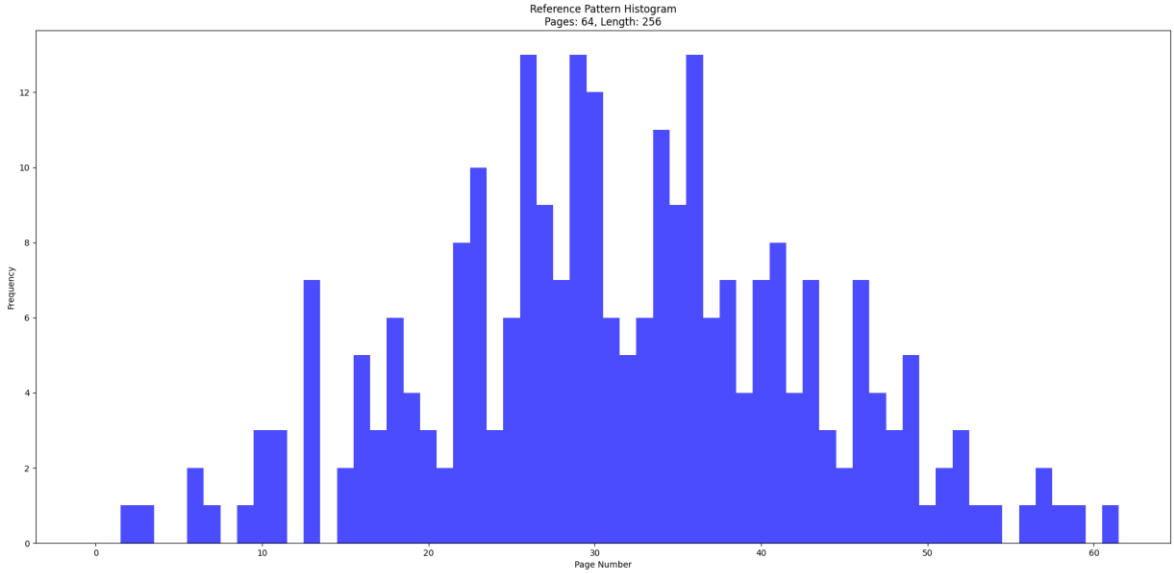

Page Faults vs Times Used
Pages: 32
FIFO
LFU
PersistentFrequencyLFU
Page Faults
Times Used

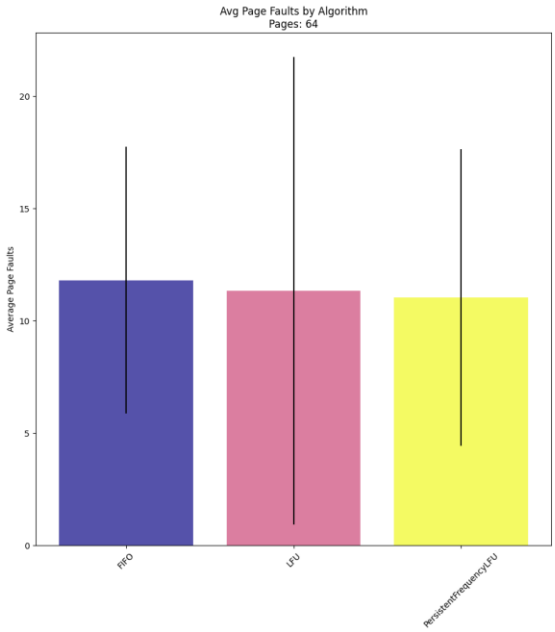

Reference Pattern Histogram
Pages: 32, Length: 1024
Frequency
Page Number

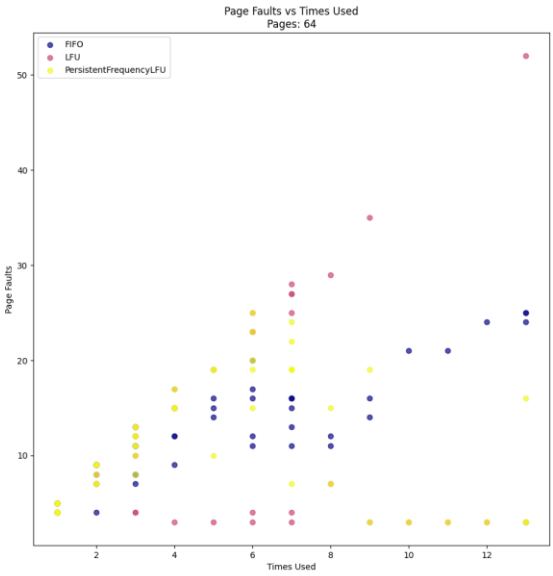

Avg Page Faults by Algorithm
Pages: 32
Average Page Faults
FIFO
LFU
PersistentFrequencyLFU

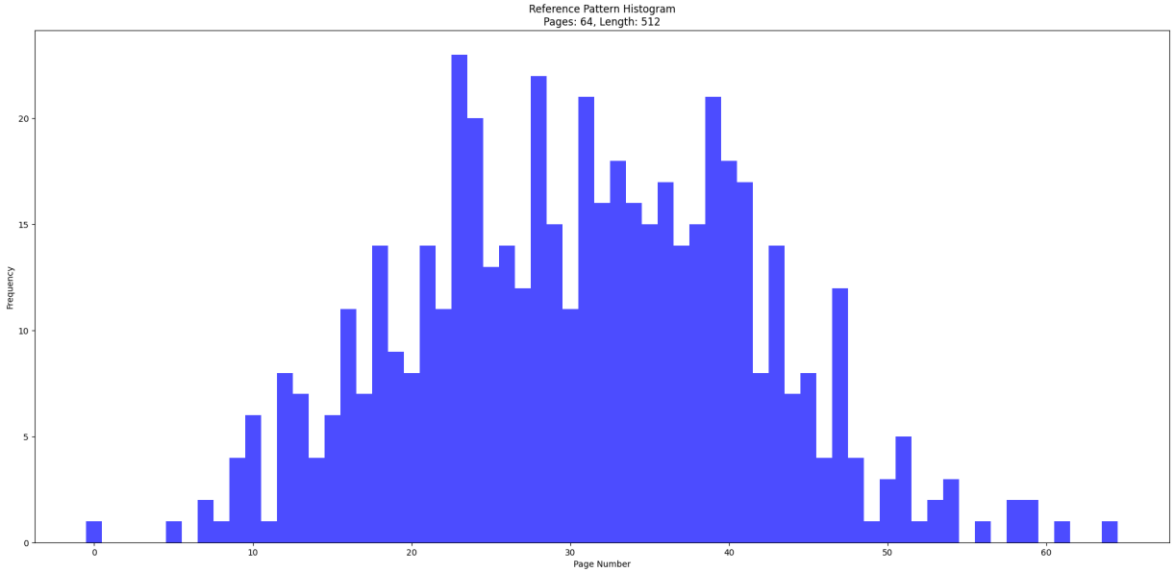

Page Faults vs Times Used
Pages: 32
FIFO
LFU
PersistentFrequencyLFU
Page Faults
Times Used

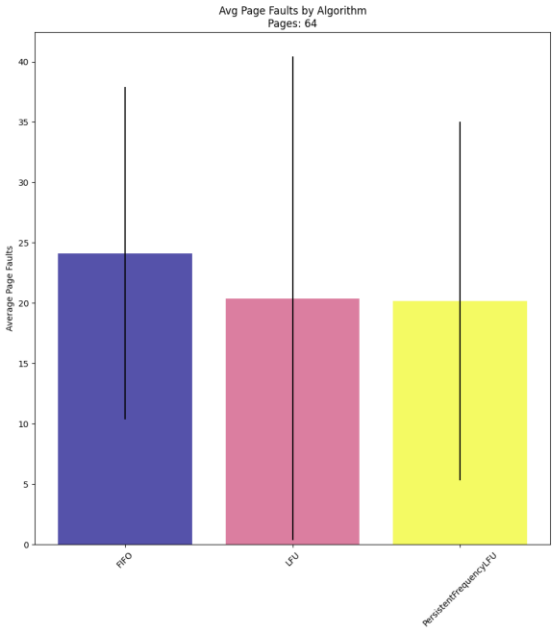
- *64 strony:*

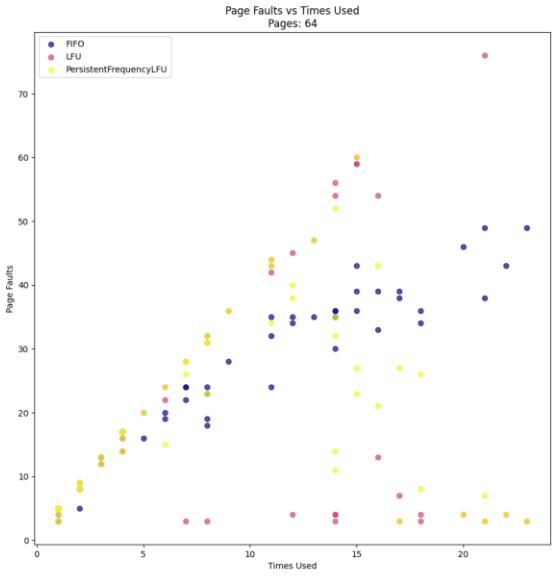

Reference Pattern Histogram
Pages: 64, Length: 256
Frequency
Page Number

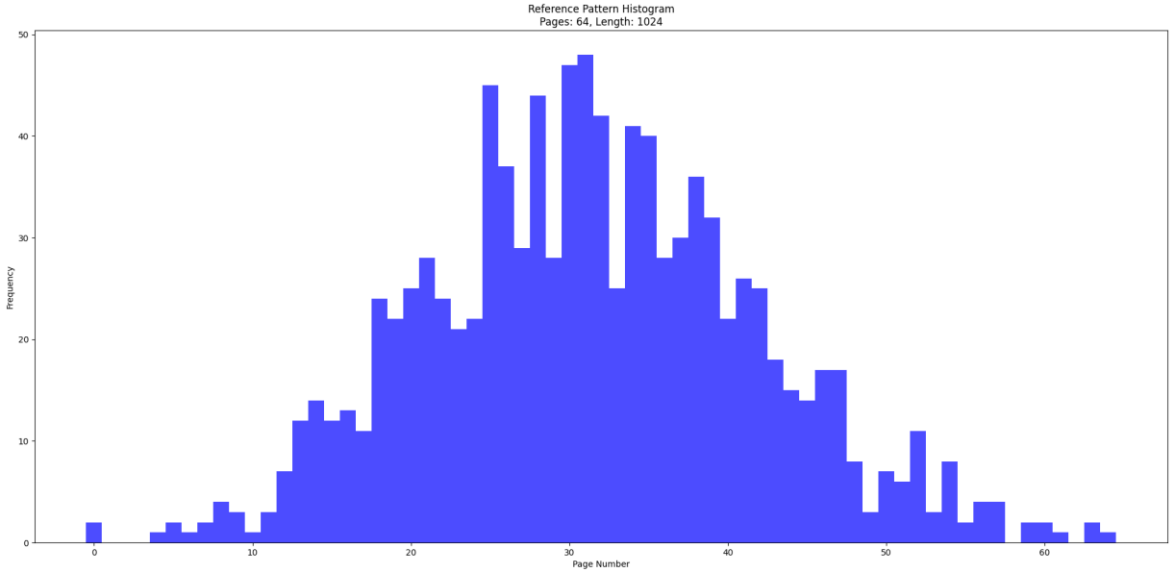

Avg Page Faults by Algorithm
Pages: 64
Average Page Faults
FIFO
LFU
PersistentFrequencyLFU

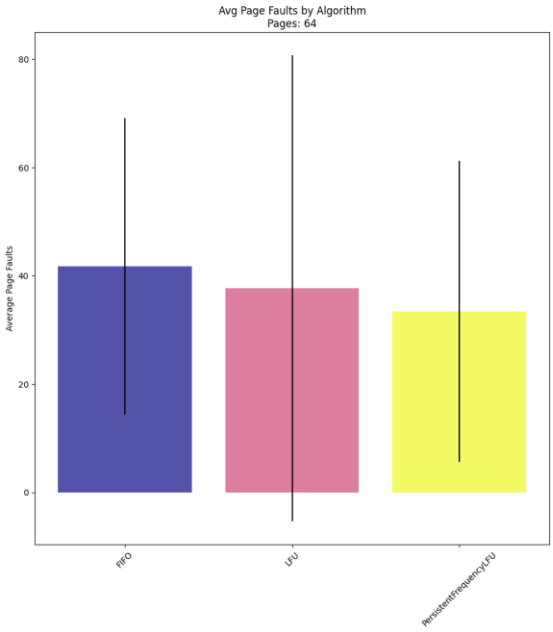

Page Faults vs Times Used
Pages: 64
FIFO
LFU
PersistentFrequencyLFU
Page Faults
Times Used

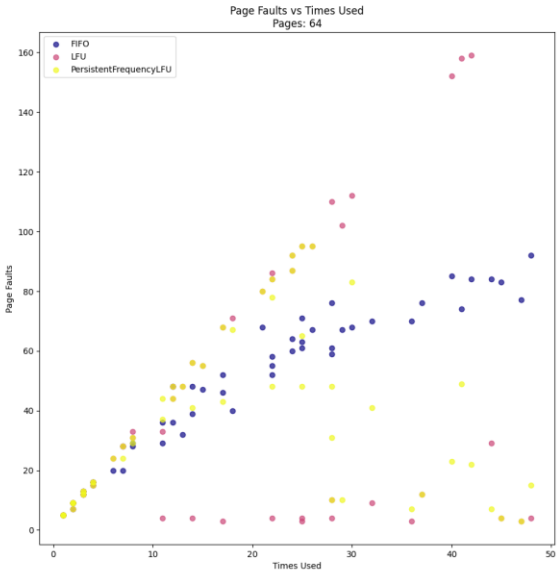

Reference Pattern Histogram
Pages: 64, Length: 512
Frequency
Page Number


Avg Page Faults by Algorithm
Pages: 64
Average Page Faults
FIFO
LFU
PersistentFrequencyLFU

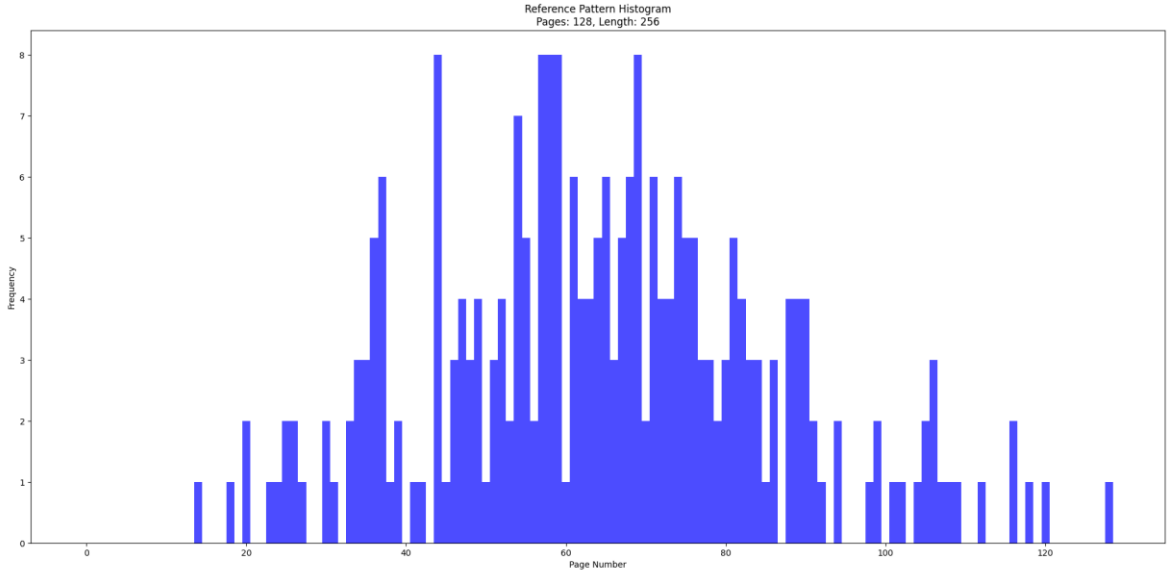

Page Faults vs Times Used
Pages: 64
FIFO
LFU
PersistentFrequencyLFU
Page Faults
Times Used

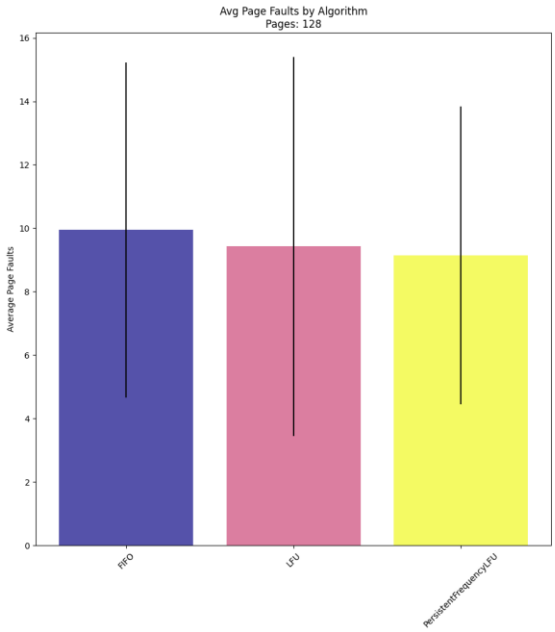

Reference Pattern Histogram
Pages: 64, Length: 1024
Frequency
Page Number

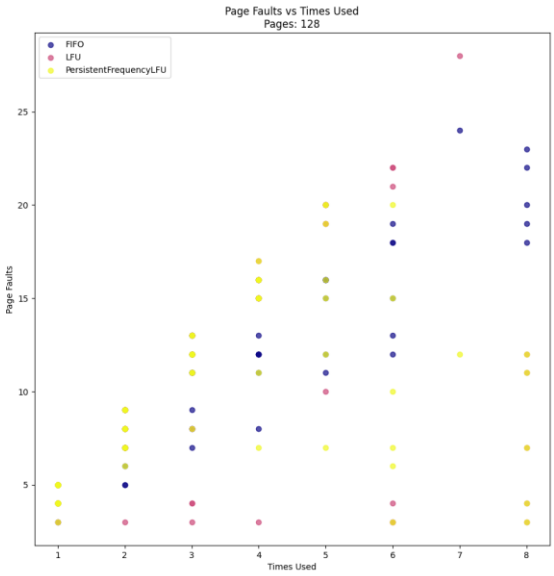

Avg Page Faults by Algorithm
Pages: 64
Average Page Faults
FIFO
LFU
PersistentFrequencyLFU

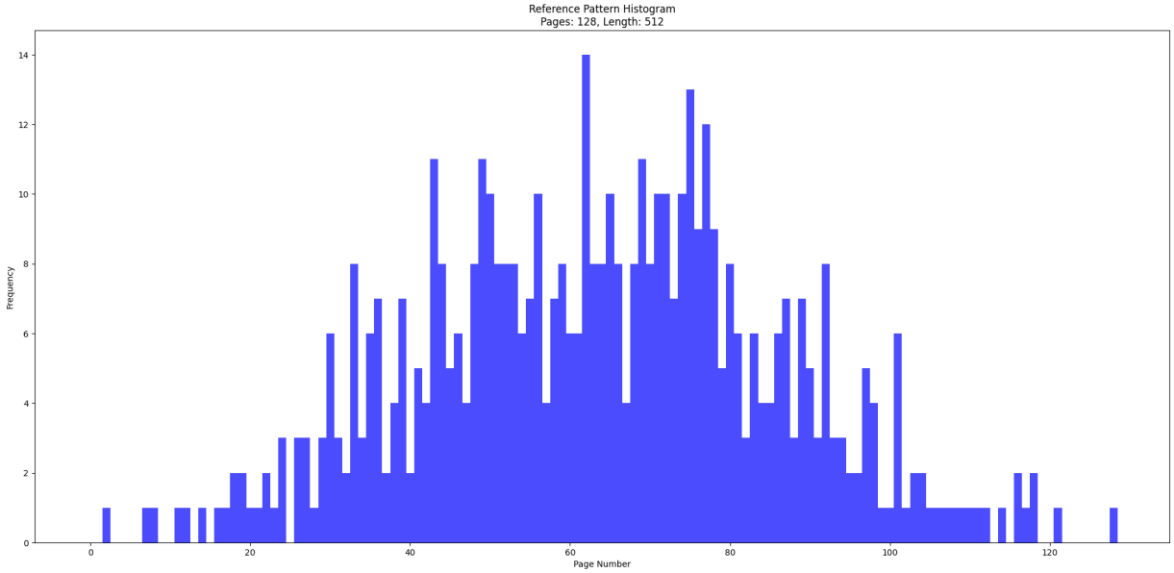

Page Faults vs Times Used
Pages: 64
FIFO
LFU
PersistentFrequencyLFU
Page Faults
Times Used

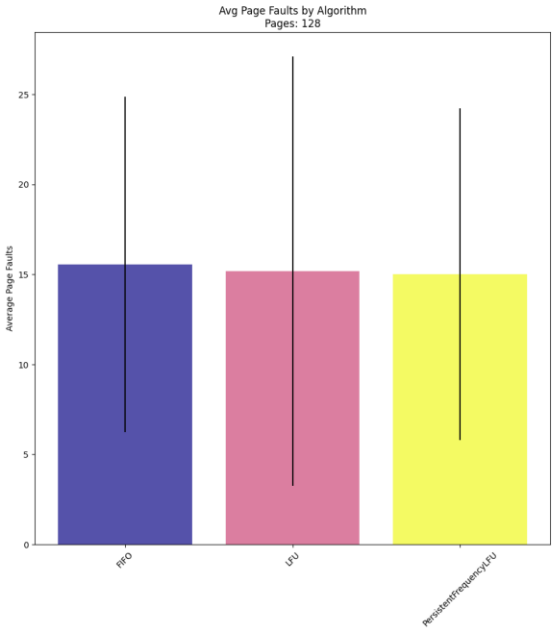
- *128 stron:*

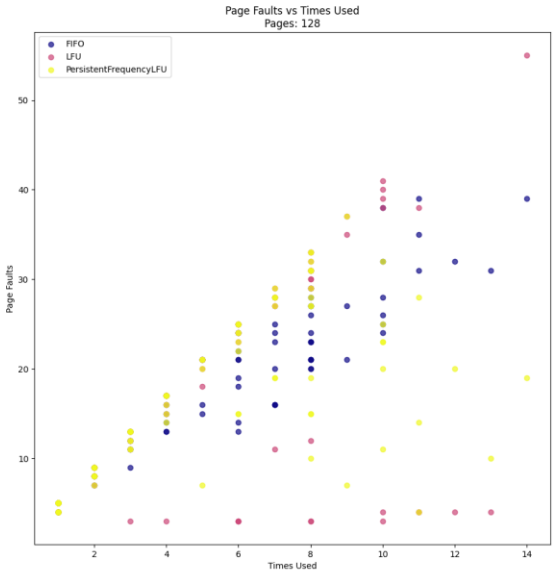

Reference Pattern Histogram
Pages: 128, Length: 256
Frequency
Page Number

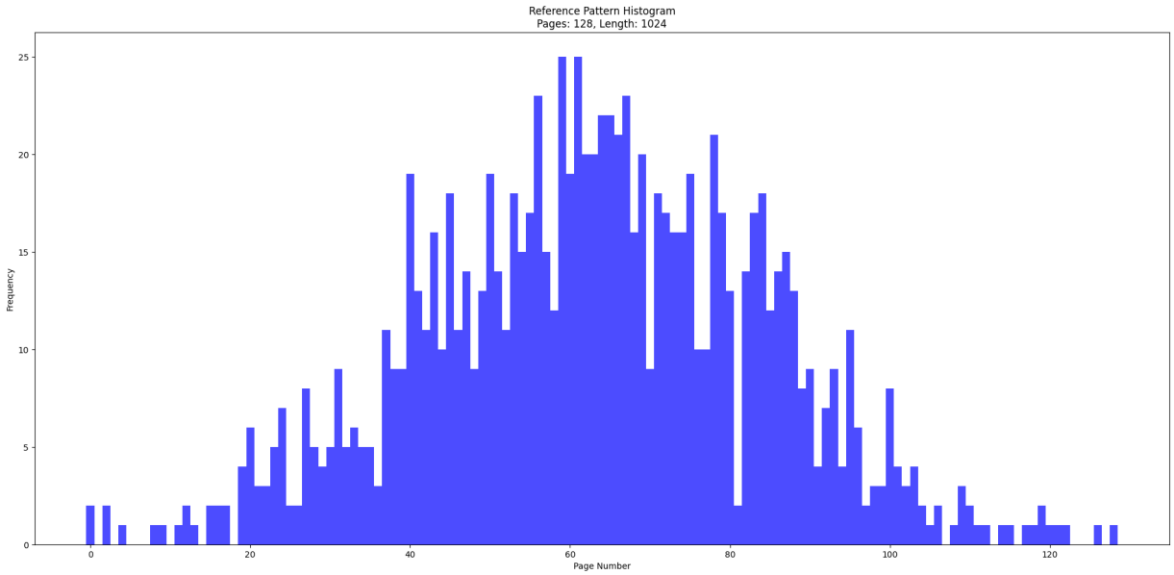

Avg Page Faults by Algorithm
Pages: 128
Average Page Faults
FIFO
LFU
PersistentFrequencyLFU

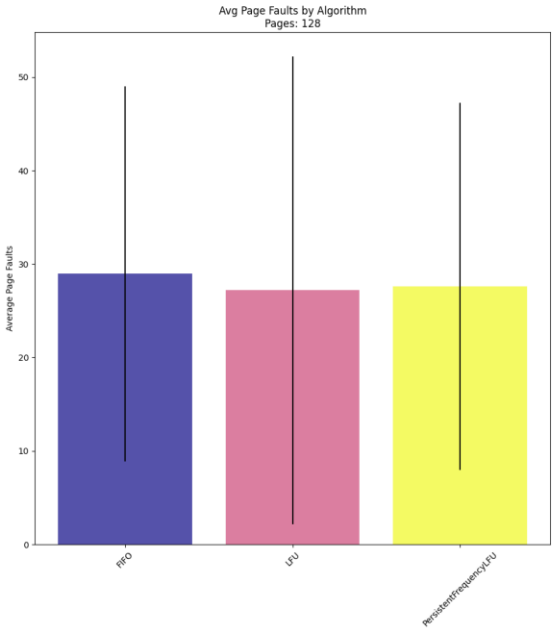

Page Faults vs Times Used
Pages: 128
FIFO
LFU
PersistentFrequencyLFU
Page Faults
Times Used

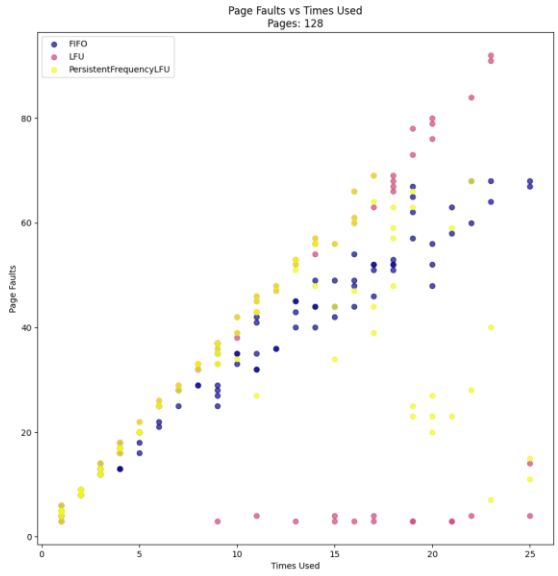

Reference Pattern Histogram
Pages: 128, Length: 512
Frequency
Page Number


Avg Page Faults by Algorithm
Pages: 128
Average Page Faults
FIFO
LFU
PersistentFrequencyLFU


Page Faults vs Times Used
Pages: 128
FIFO
LFU
PersistentFrequencyLFU
Page Faults
Times Used







Wnioski:

W większości wyników można zauważyć że najniższą wartość średnich błędów stronnicowania osiąga algorytm LFU z długotrwałym licznikiem, niewiele gorszy bądź podobny był algorytm LFU, a na końcu plasował się FIFO.

Przewaga LFU z długotrwałym licznikiem polegała na tym, że zastosowane sekwencje były sztucznie wygenerowane z rozkładu normalnego. W rzeczywistych przypadkach będzie występowało zjawisko zmian lokalności, co przekłada się na miejsca w sekwencji dostępów gdzie zagęszczone jest dużo dostępów do małej ilości najbardziej potrzebnych stron, po czym program kończy część w której potrzebował danych na tych stronach, i inne strony mają zagęszczenie dostępu. W takiej sytuacji algorytm LFU poradziłby sobie lepiej, bo szybciej wykryłby że nastąpiła zmiana. Natomiast przy równomiernie rozłożonych dostępach do najbardziej pożądanych stron w rozkładzie normalnym wersja z długotrwałym licznikiem radzi

sobie lepiej, ponieważ statystycznie jeżeli dotychczas strona pojawiła się ponadprzeciętnie wiele razy, to najpewniej będzie to kontynuować na przestrzeni sekwencji.

Algorytm FIFO radzi sobie nieco gorzej, co prawda jego stosunek błędów do ilości użyć strony spada przy częściej używanych stronach, zwłaszcza dla długich sekwencji dostępu, i w sytuacjach gdzie rozmiar ramki jest proporcjonalnie duży w stosunku do ilości stron, ale nie może on konkurować z wersjami LFU dla najczęściej używanych stron, ponieważ tam ich ilość błędów stron jest znacznie niższa.

Na wykresach słupkowych czarną linią zaznaczone są estymatory odchylenia standardowego ilości błędów strony dla poszczególnych algorytmów. Można zauważyć, że FIFO ma często najniższe odchylenia, natomiast w tym przypadku nie jest to pozytywne, ponieważ oznacza to że mniej priorytetyzuje on pewne strony, a konkretnie te najbardziej używane. LFU często ma największą wartość odchylenia standardowego, co może wynikać z dużej ilości błędów które ten algorytm powoduje w sytuacjach gdy użycia strony nie są zlokalizowane, a bardziej równomiernie rozłożone w sekwencji.

Na wykresach punktowych można zauważyć znaczną przewagę wariantów LFU nad FIFO dla częściej używanych stron. LFU ma często używane strony które znajdują się znacznie niżej od wykresu wersji z długotrwałym licznikiem, natomiast widać też odchylenia w drugą stronę spowodowane brakiem lokalności.

Wynik testu wskazuje na LFU z długotrwałym licznikiem jako najlepszy algorytm, ale warunki testu nie są zgodne z rzeczywistymi, gdzie algorytm LFU najpewniej poradziłby sobie lepiej.