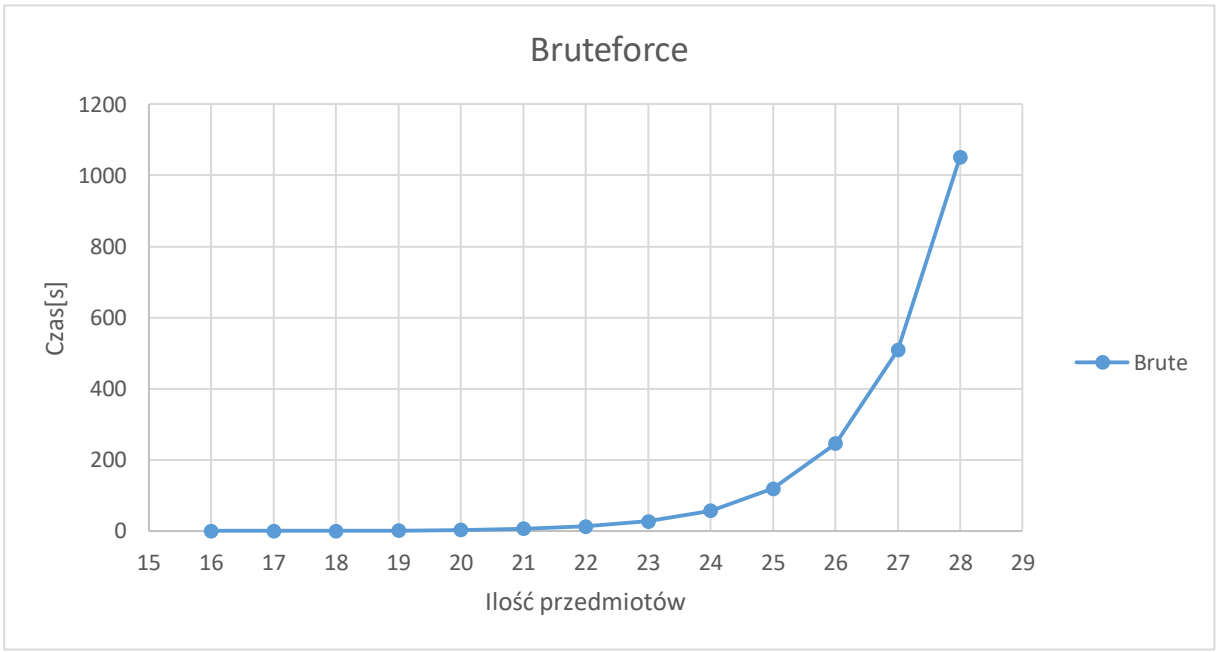
Algorytmy i Struktury Danych – Programowanie Dynamiczne

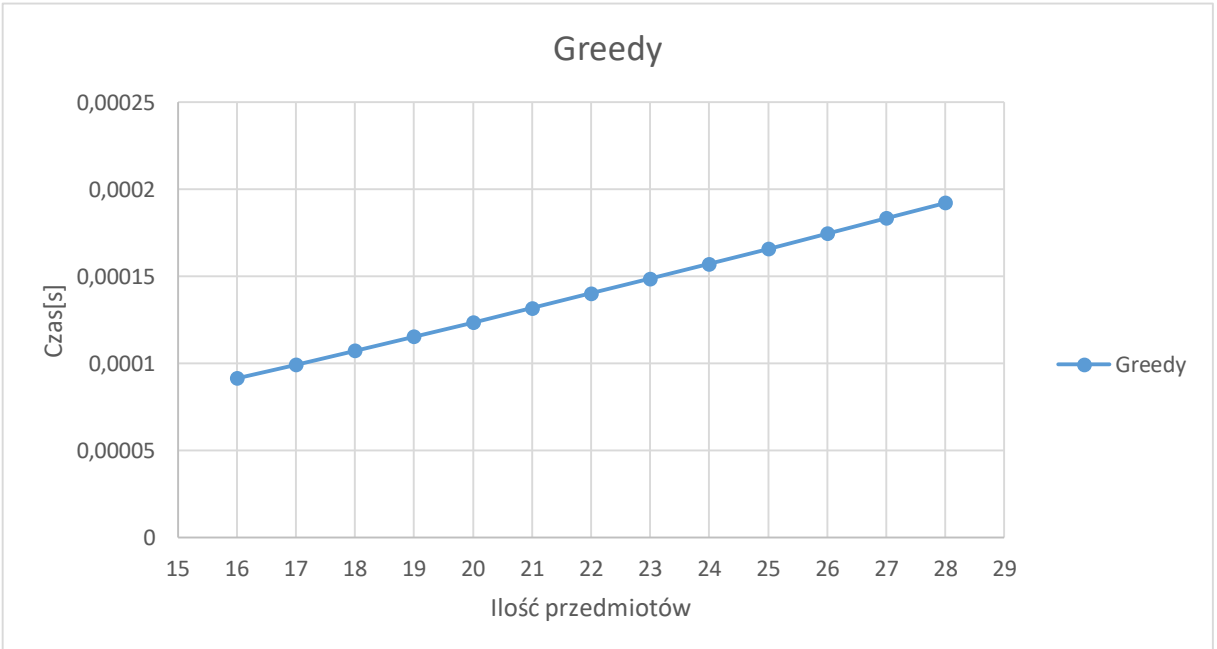
Filip Rosiak

1. Wykres t=f(n) zależności czasu obliczeń t od liczby n przedmiotów, przy stałej pojemności plecaka (pojemność plecaka 700).

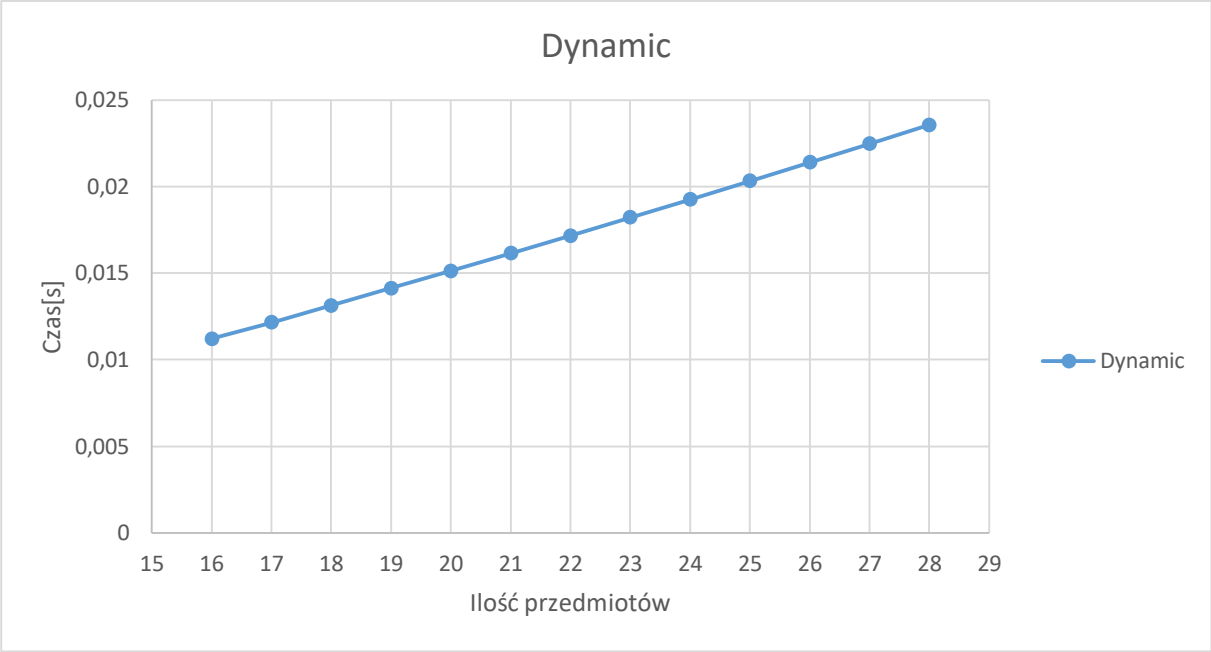
a)



b)

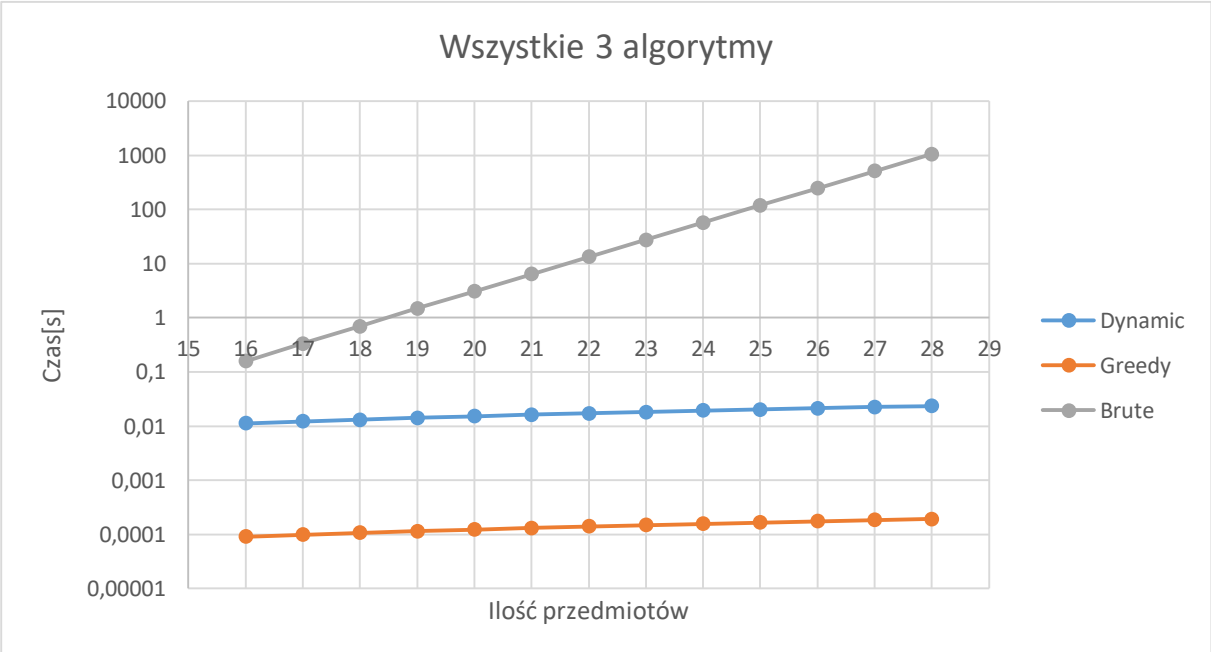


c)



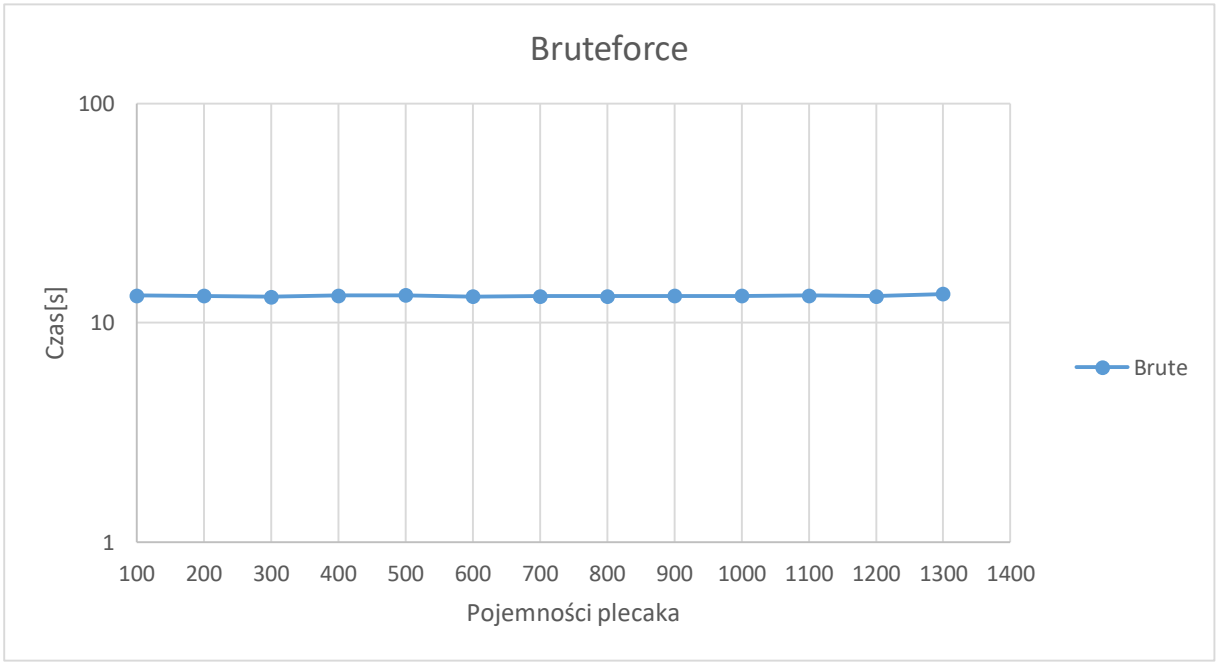
2. Wykres t=f(n) zależności czasu obliczeń t od liczby n przedmiotów, przy stałej pojemności plecaka (pojemności plecaka 700).

a)

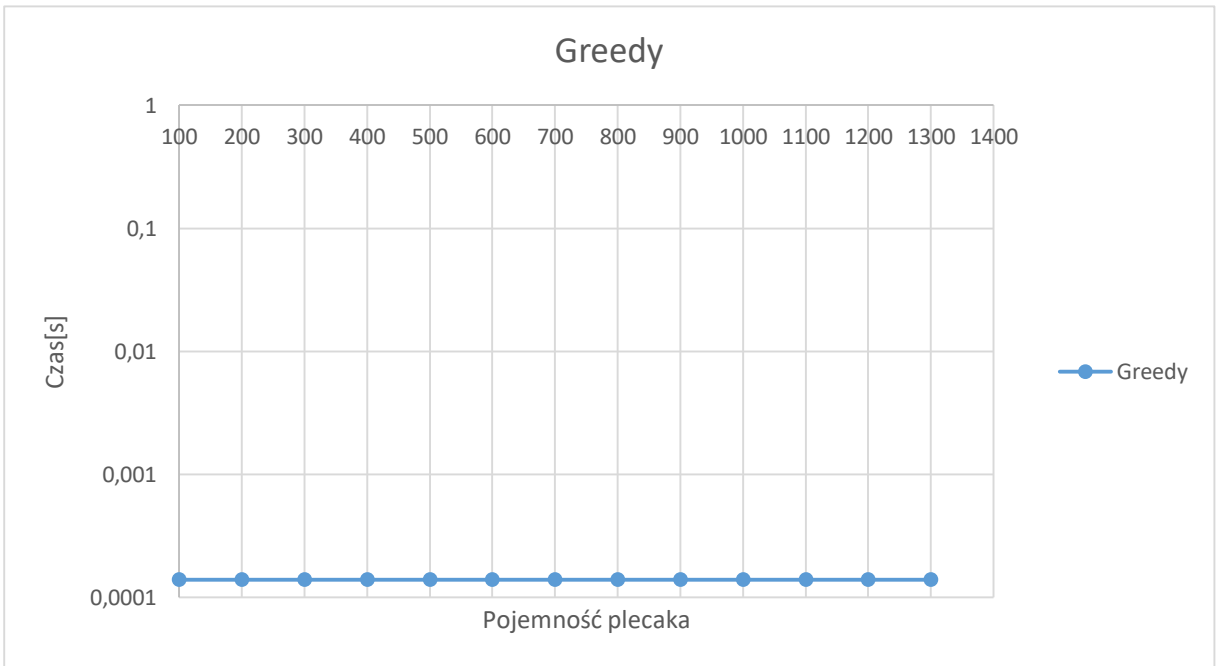


3. Wykres t=f(b) zależności czasu obliczeń t od pojemności plecaka b, przy stałej ilości przedmiotów n (ilość przedmiotów 22).

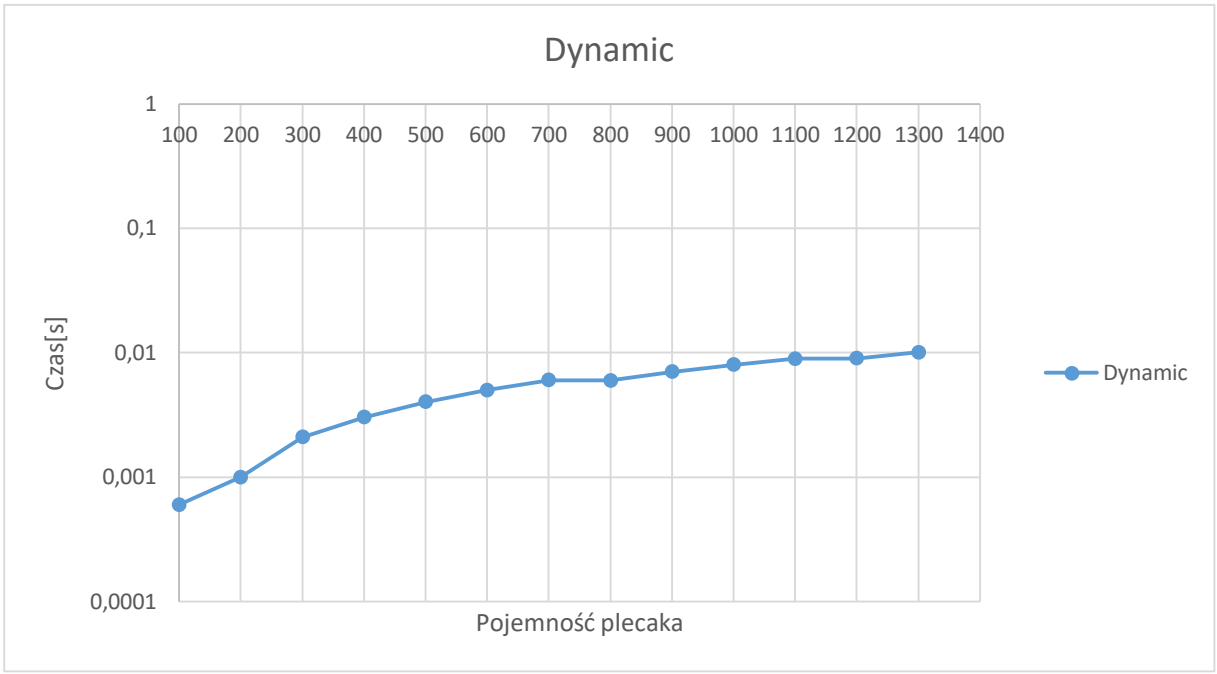
a)



b)

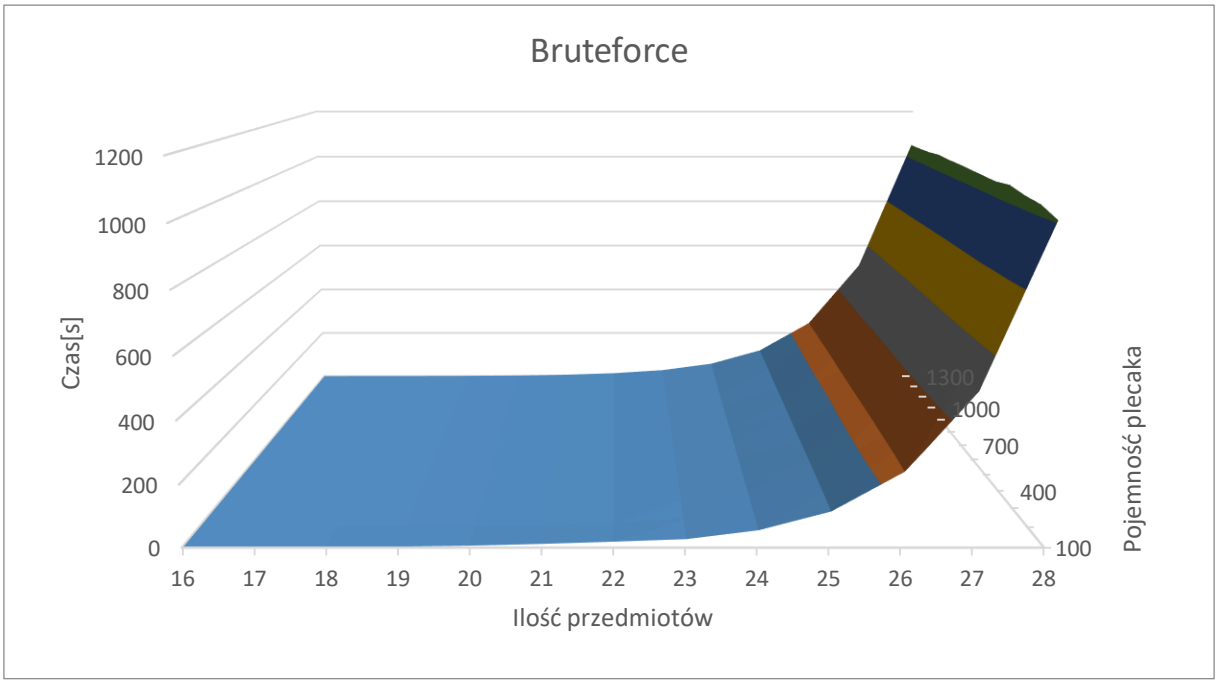


c)

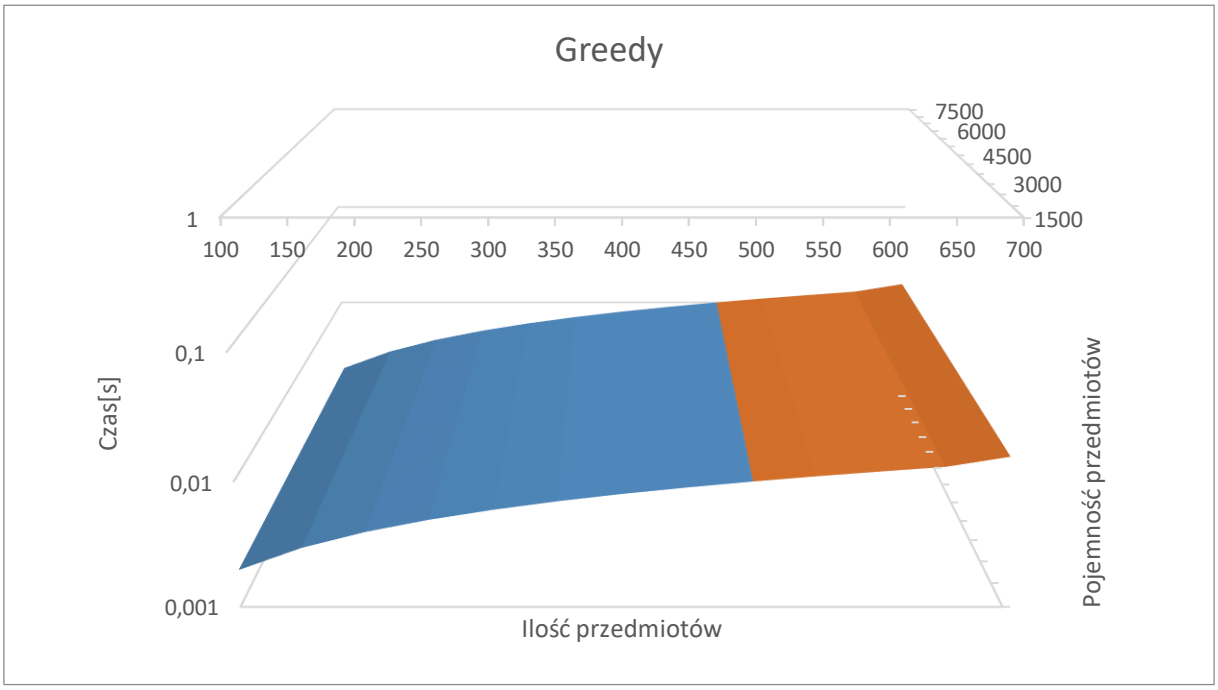


4. Wykresy t=f(n,b) zależności czasu obliczeń t od liczby n przedmiotów i pojemności plecaka b.

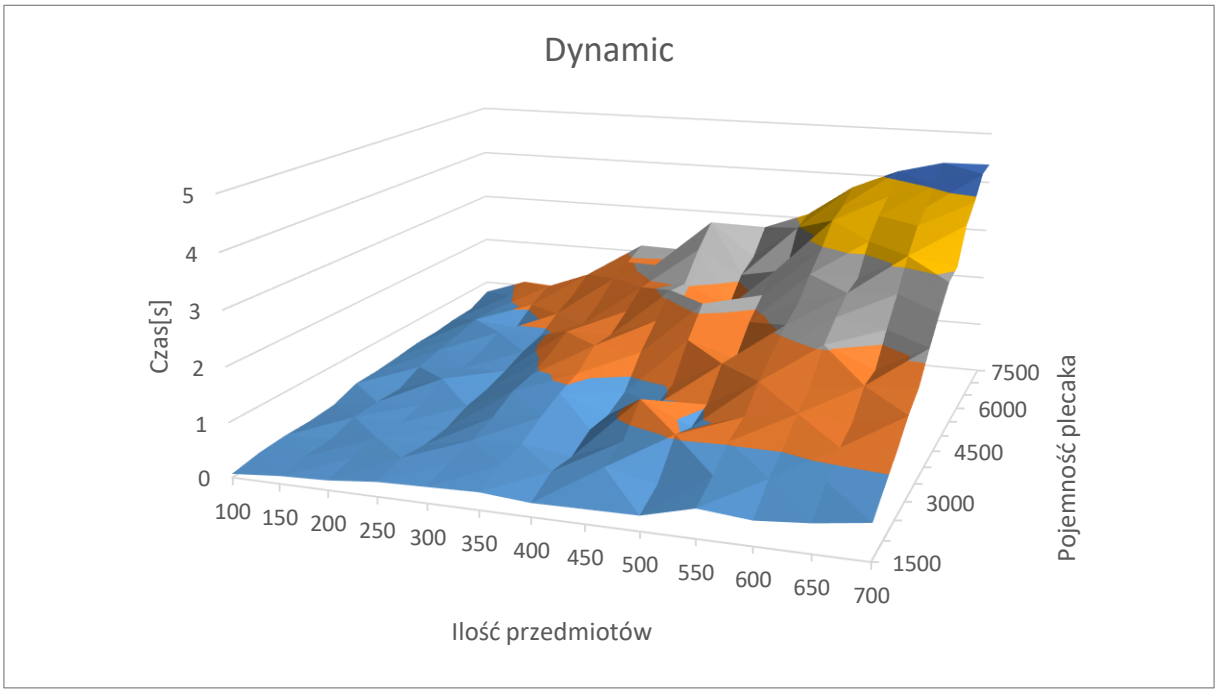
a)



b)


Greedy
7500
6000
4500
3000
1500
1
100 150 200 250 300 350 400 450 500 550 600 650 700
0,1
0,01
0,001
Czas[s]
Ilość przedmiotów
Pojemność przedmiotów

c)


Dynamic
5
4
3
2
1
0
Czas[s]
100 150 200 250 300 350 400 450 500 550 600 650 700
Ilość przedmiotów
7500
6000
4500
3000
1500
Pojemność plecaka

5. Punty 5 oraz 6

   Problem plecakowy jako problem decyzyjny należy do klasy problemów NP-trudnych. Natomiast jako problem decyzyjny należy do klasy problemów NP-zupełnych ze względu na złożoność pseudowielomianową algorytmów dynamicznych służących do jego rozwiązania.

   Algorytm Brutefoce jest najgorszym złożeniowo algorytmem. Polega on na wygenerowania każdego możliwego ułożenia przedmiotów w plecaku, a zatem każdy przedmiot może albo być albo nie być w plecaku. Takie podejście sprawia, że musimy sprawdzić aż 2^n różnych ułożeń w plecakach co już przy 28 przedmiotach trwa ~15 minut i każdy kolejny przedmiot wydłuży ten czas około dwukrotnie. Złożoność tego algorytmu wynosi O(2^n). Jedynym plusem tego algorytmu jest to, iż znajduje on rozwiązanie optymalne.

   Głównym czynnikiem wpływający na wydajność algorytmu zachłannego w tej implementacji jest szybkość sortowania użytego przy ustalaniu kolejności tablicy zabierającej stosunki wartości elementów do ich mas. W tej implementacji została użyta wbudowana w języku Python metoda .sort() o złożoności O(n*logn). Druga część algorytmu, czyli dodawanie kolejnych mieszczących się elementów do plecaka po kolei z listy wykonuje się w czasie liniowym, a zatem złożonością całego algorytmu będzie O(n*logn). Algorytm jest niezależny od całkowitej pojemności plecaka co doskonale widać na wykresach 3b oraz 4b - przy stałej ilości przedmiotów algorytm wykonuje się w praktycznie jednakowym czasie. Przez to, że algorytm zachłanny bardziej koncentruje się na szybkości niż dokładności rozwiązania, nie zawsze jest ono optymalne, przez co czasami pojawia się pewien błąd w wartości całkowitej plecaka w porównaniu do pozostałych algorytmów. To czy algorytm zachłanny znajdzie rozwiązanie optymalne zależy od kilku czynników, przede wszystkim nie ma gwarancji na rozwiązanie optymalne w przypadku gdy masa wszystkich przedmiotów jest większa od pojemności plecaka. Następnym czynnikiem jest chociażby sposób posortowania przedmiotów zanim zostaną umieszczone w plecaku (np. poprzez wagę lub wartość przedmiotu).

W przypadku podejścia dynamicznego w obu sytuacjach algorytm wykazuje bardzo podobną wydajność czasową. Dzieje się tak ponieważ działanie algorytmu polega na wygenerowaniu macierzy algorytmu dynamicznego informującej jaką wartość możemy osiągnąć dla danej pojemności plecaka z daną ilością dostępnych elementów, a czas wygenerowania tej macierzy będzie zależał właśnie od jej rozmiarów, czyli aktualnego n i b. Z tego powodu złożoność algorytmu będzie miała postać pseudowielomianową O(n*b). Doskonale widać to na wykresie 4c, gdzie czas wykonywania się algorytmu rośnie „po skosie”, a zatem wraz z pojemnością plecaka jak i ilością przedmiotów.