# Programowanie dynamiczne
## Problem 0-1 plecakowy




***Wykonanie***

***Adrian Madajewski*** 145406

***Michał Kwarta*** 145192

# 1. Informację wstępne

Problem 0-1 plecakowy w wersji decyzyjnej jest problemem NP-zupełnym, co oznacza że nie ma prawidłowego i szybkiego algorytmu rozwiązującego problem w wielomianowym czasie - istnieje za to algorytm programowania dynamicznego, który jest pseudowielomianowy. Wersja optymalizacyjna problemu jest natomiast NP-trudna.

Prezentowane dane są wynikiem serii 5 testów, w których to dane przedmiotów zostały wygenerowane w następujący sposób:

- wartość jednostki przedmiotu - liczba z przedziału $[1,\ 10]$.
- waga jednostki przedmiotu - liczba z przedziału $[1,\ b/2]$, gdzie $b$ określa pojemność plecaka, w celu zabezpieczenia przed niepotrzebnymi danymi.

W dalszych punktach sprawozdania prezentowane są wykresy, wraz z szczegółowym opisem wygenerowanych danych oraz poprzedzające je wnioski.

Sprawozdanie to ma na celu przedstawić i porównać algorytmy rozwiązujące 0-1 problem plecakowy następującymi implementacjami:

- algorytm zachłanny *AZ* (sortowanie według współczynnika opłacalności),
- algorytm wyczerpujący / siłowy *AW,*
- algorytm programowania dynamicznego *APD.*

# 2. Wykresy algorytmów

Poniżej prezentowane są wykresy wraz z zakresami danych poszczególnych algorytmów rozwiązujących 0-1 problem plecakowy.

Wprowadzone w niektórych przypadkach ograniczenia są skutkiem *bardzo wolnego* przetwarzania danych przez algorytm wyczerpujący *AW* - powód wprowadzenia ograniczeń znajduję się we wnioskach końcowych dotyczących powyższego algorytmu.

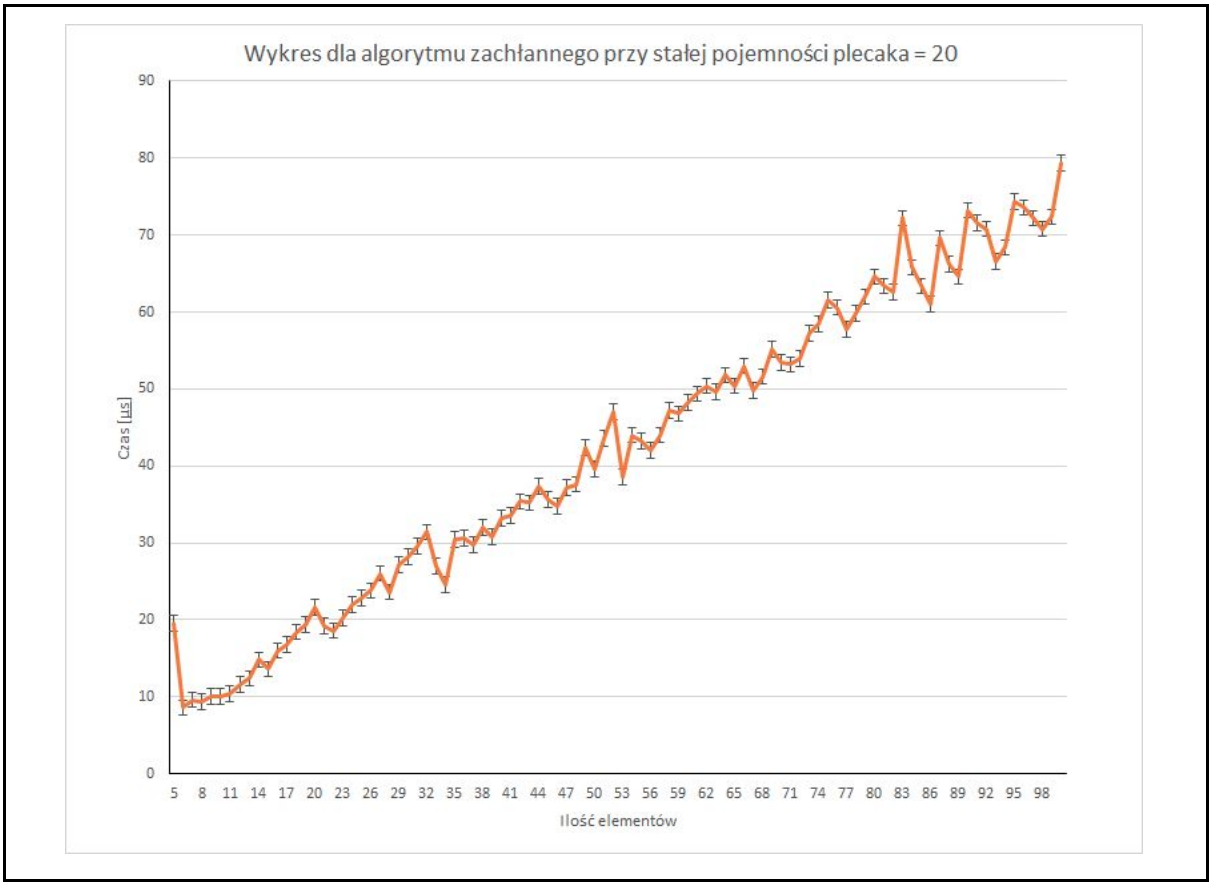
## 2.1. Według stałej pojemności plecaka

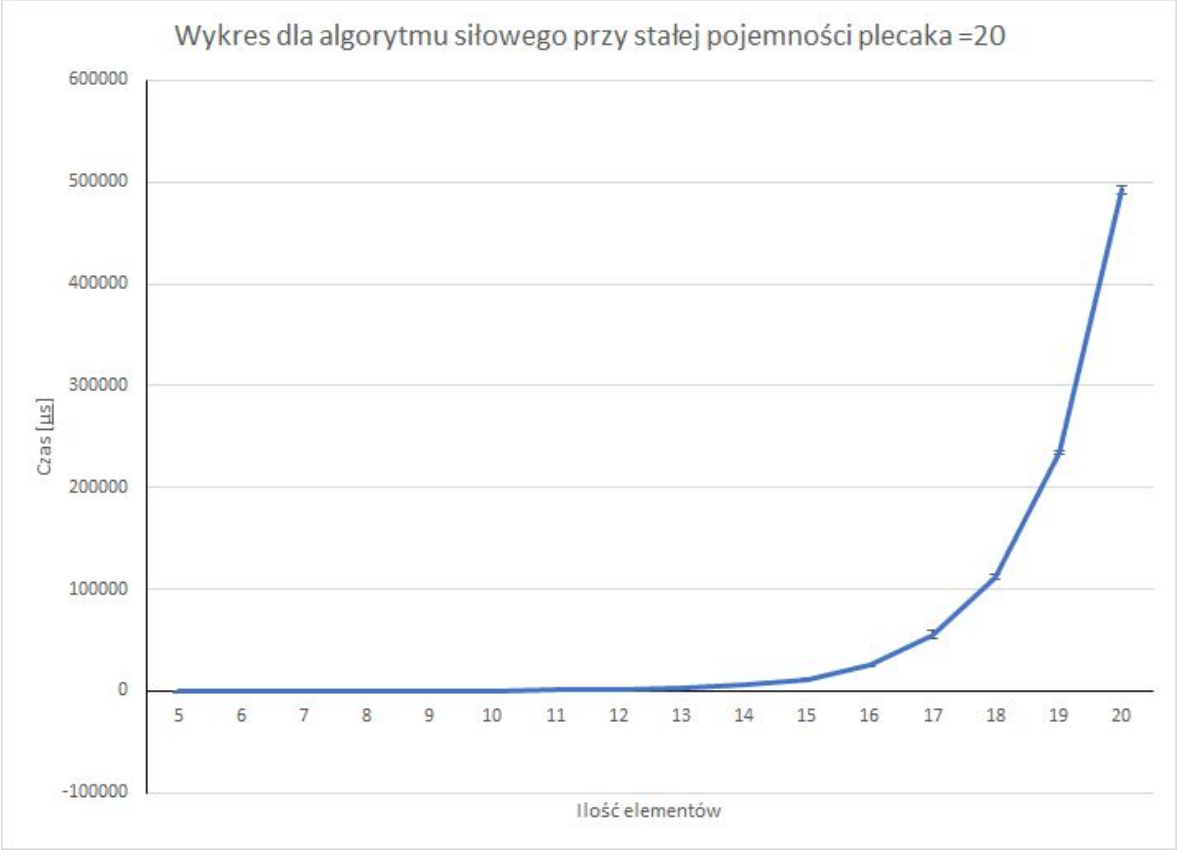
Generowane dane dla algorytmów rozwiązujących 0-1 problem plecakowy:

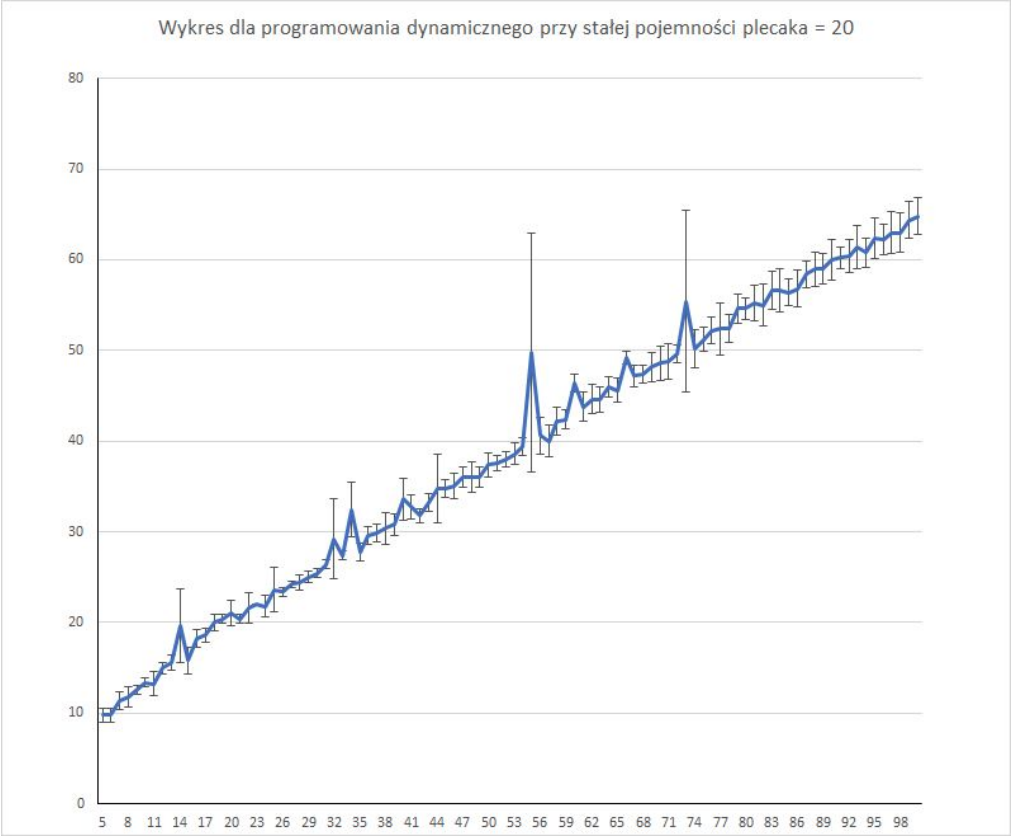
- pojemność plecaka $b = 20 - const$.
- liczba przedmiotów: zakres $[5,\ 100]$.

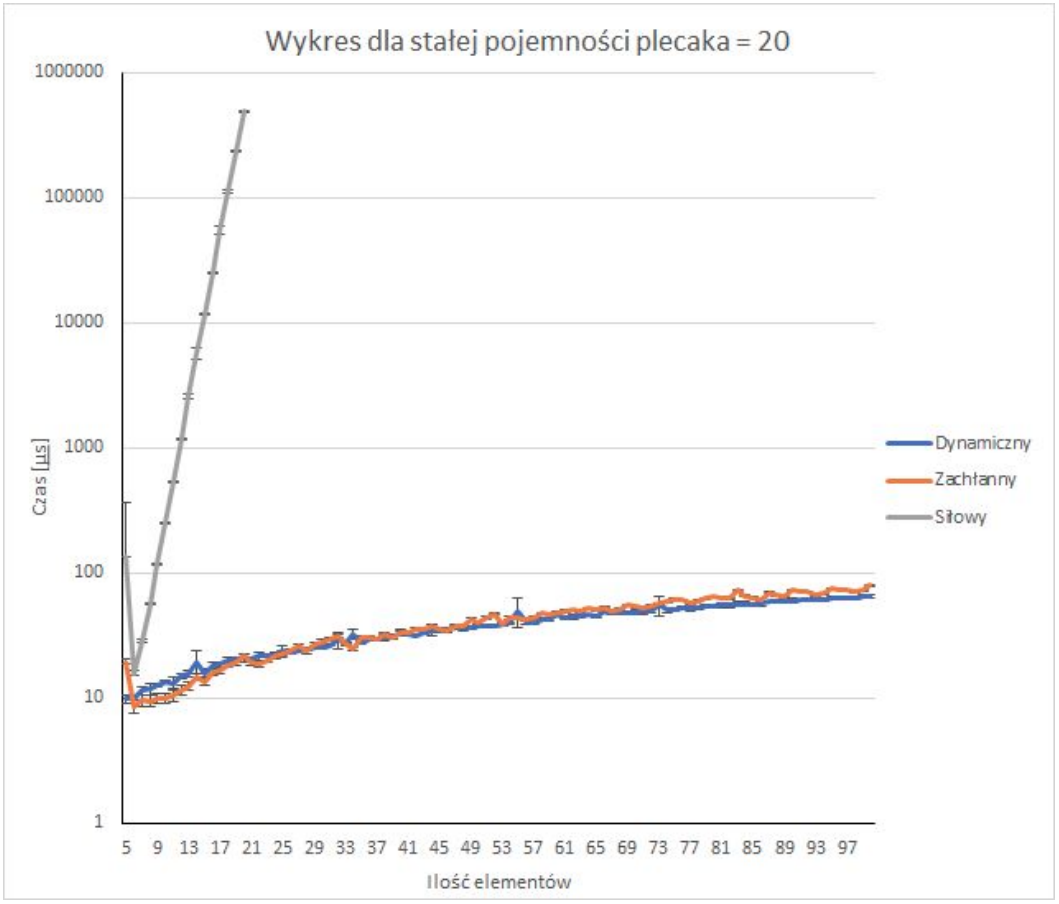
*UWAGA!* W przypadku algorytmu wyczerpującego / siłowego *AW* wprowadzono ograniczenia:

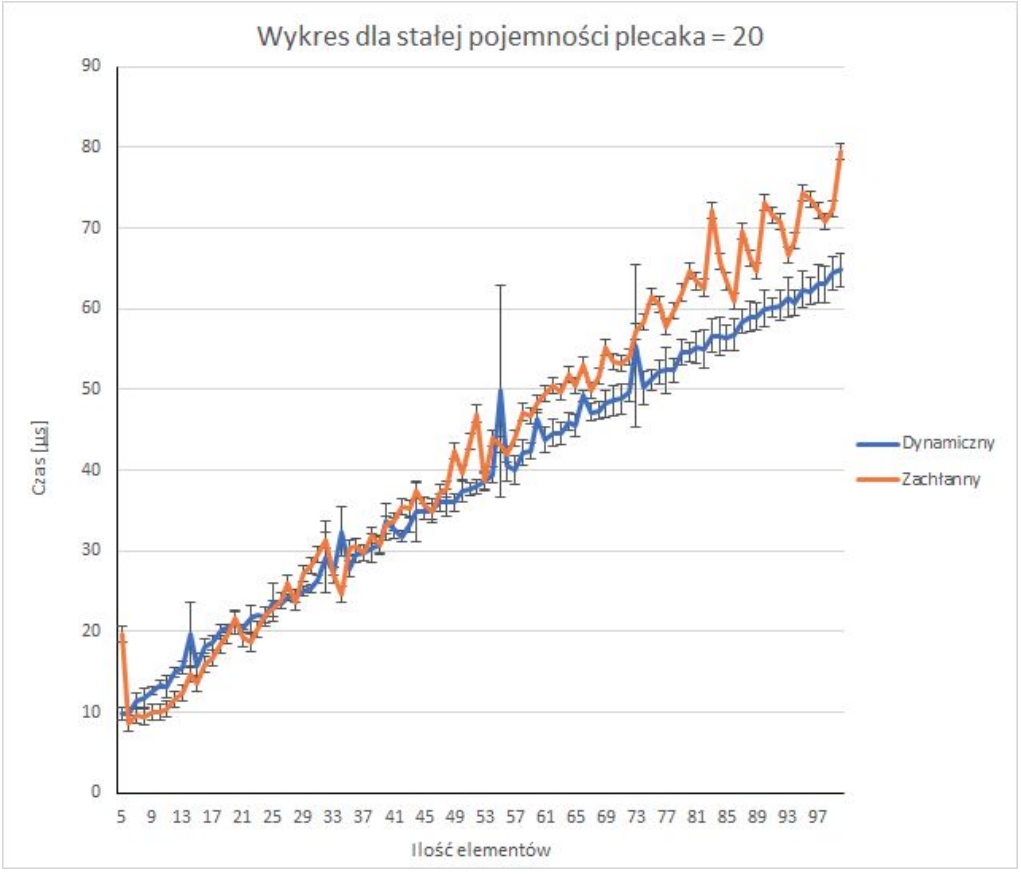
- liczba przedmiotów w plecaku: zakres $[5,\ 20]$.




Wykres dla algorytmu siłowego przy stałej pojemności plecaka =20
600000
500000
400000
300000
200000
100000
0
-100000
Czas [µs]
5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Ilość elementów


Wykres dla programowania dynamicznego przy stałej pojemności plecaka = 20
80
70
60
50
40
30
20
10
0
5 8 11 14 17 20 23 26 29 32 35 38 41 44 47 50 53 56 59 62 65 68 71 74 77 80 83 86 89 92 95 98


Wykres dla stałej pojemności plecaka = 20
1000000
100000
10000
1000
100
10
1
Czas [µs]
5 9 13 17 21 25 29 33 37 41 45 49 53 57 61 65 69 73 77 81 85 89 93 97
Ilość elementów
Dynamiczny
Zachłanny
Siłowy


Wykres dla stałej pojemności plecaka = 20
90
80
70
60
50
40
30
20
10
0
Czas [µs]
5 9 13 17 21 25 29 33 37 41 45 49 53 57 61 65 69 73 77 81 85 89 93 97
Ilość elementów
Dynamiczny
Zachłanny

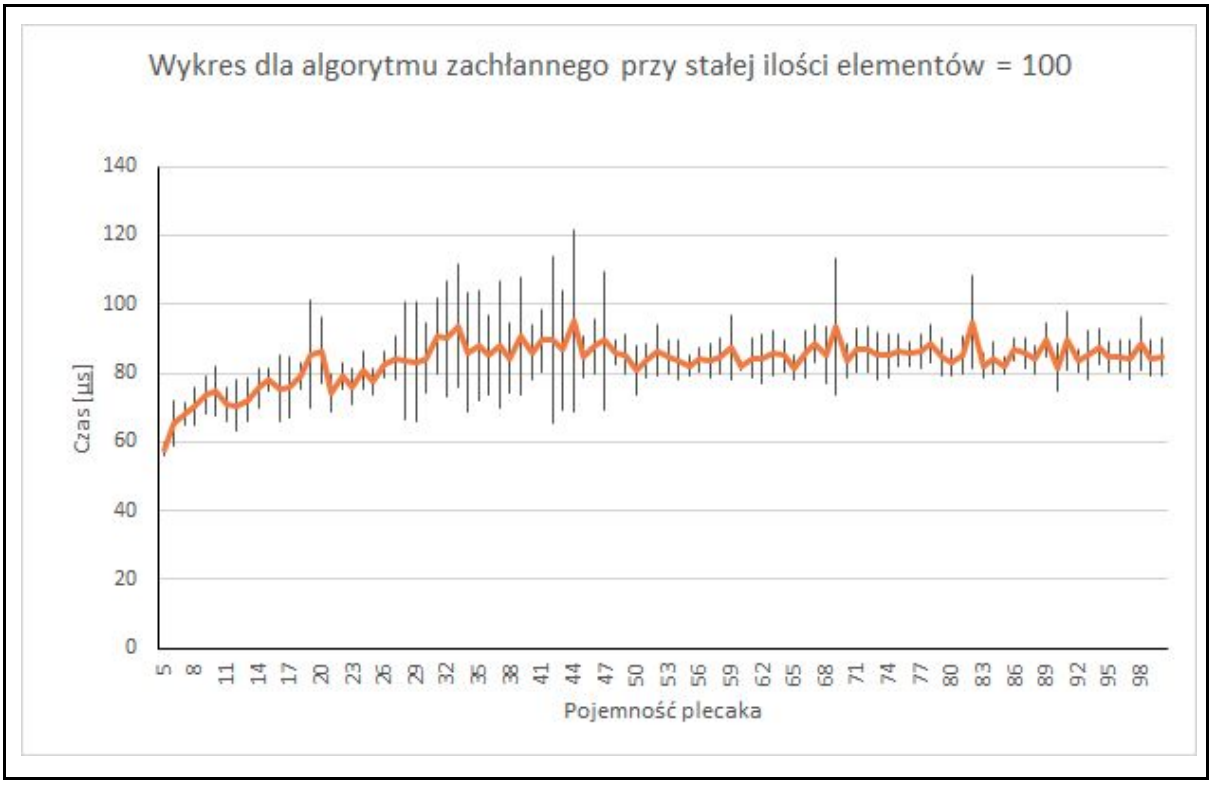
## 2.2. Według stałej liczby przedmiotów

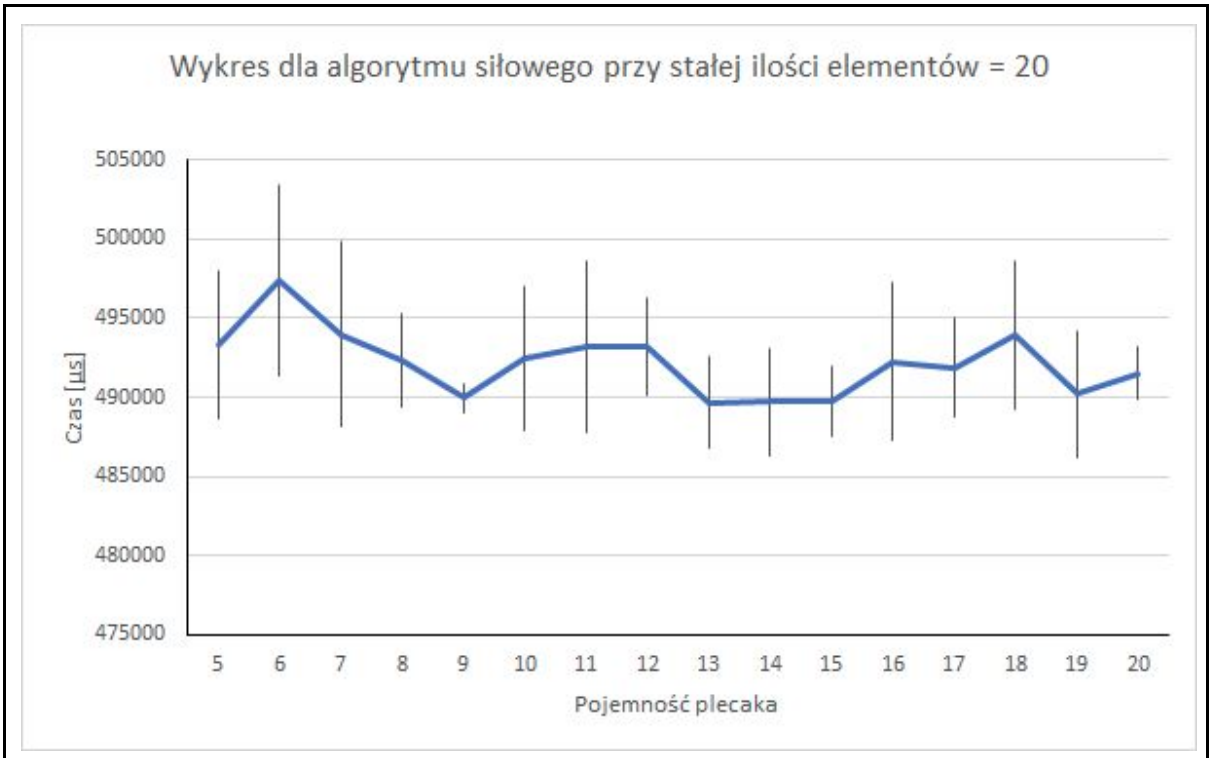
Generowane dane dla stałej liczby przedmiotów:

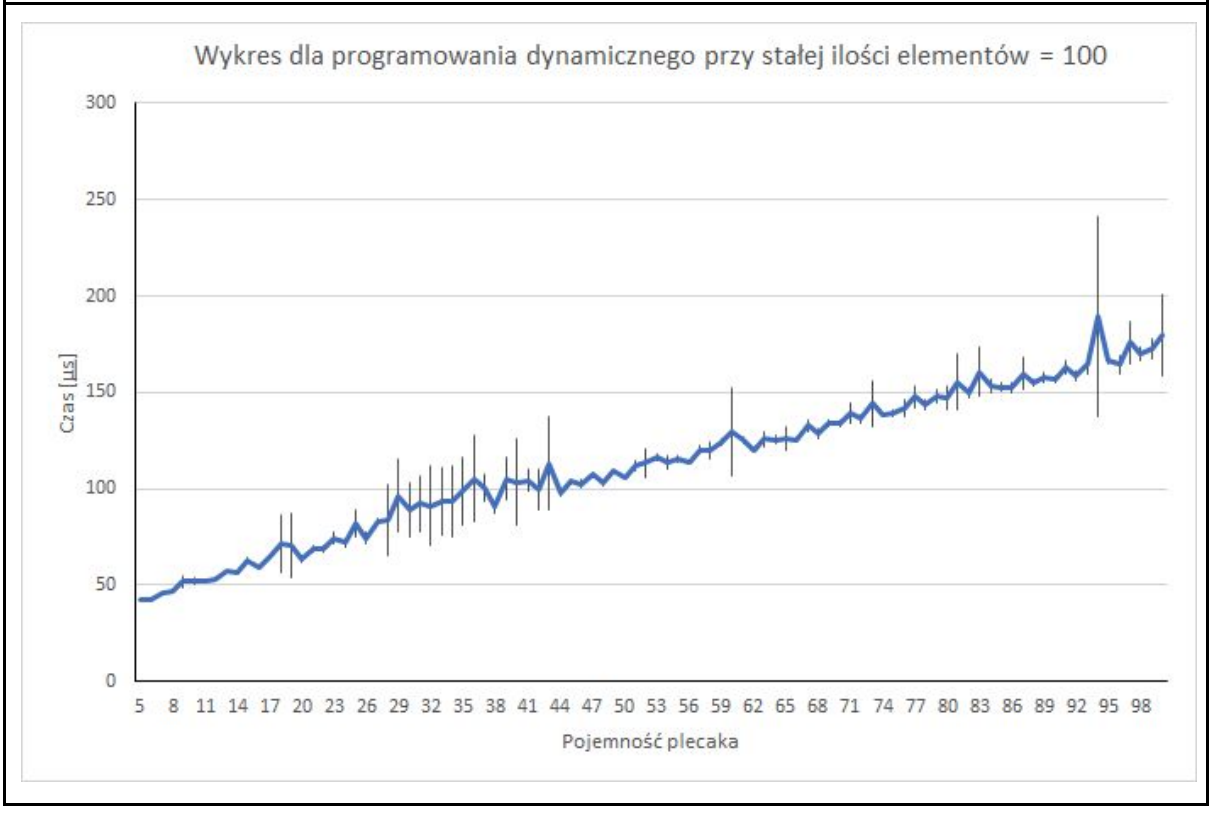
- pojemność plecaka: zakres $[5,\ 100]$.
- liczba przedmiotów: $n = 100 - const$.

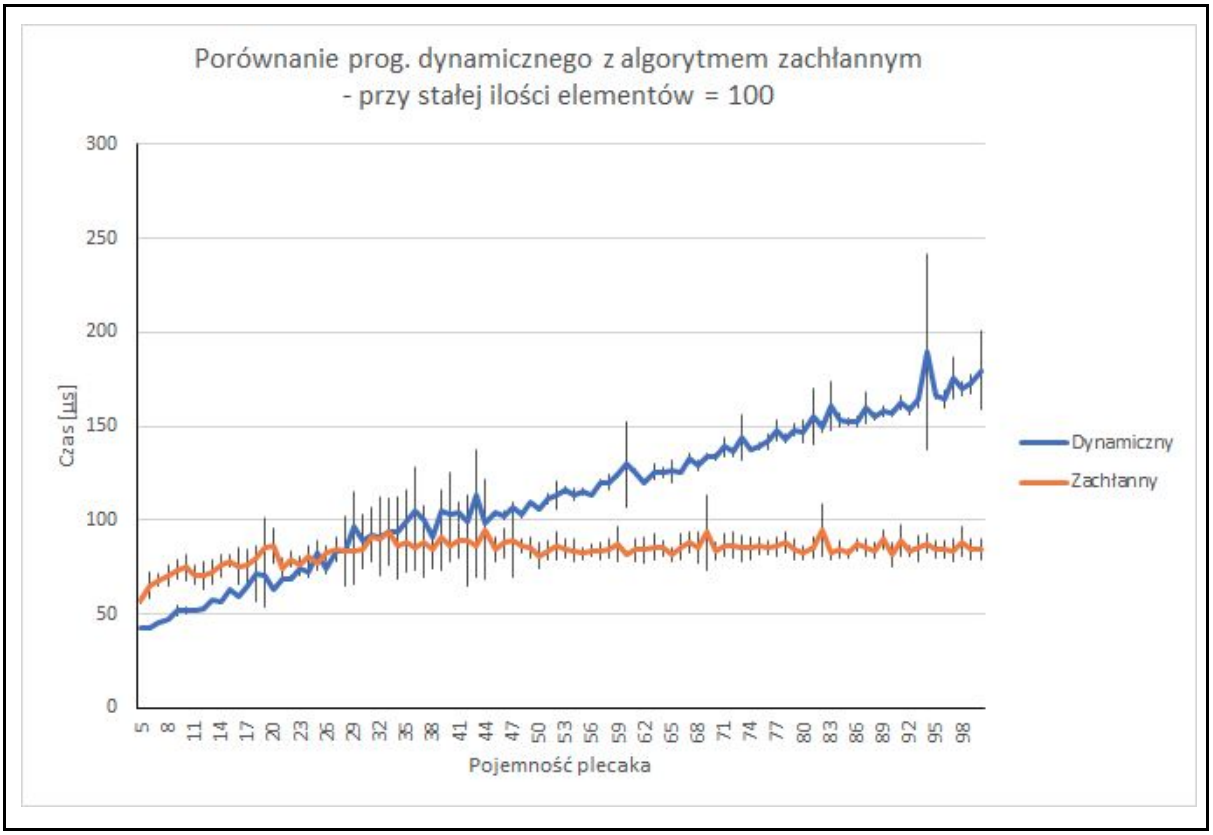
*UWAGA!* W przypadku algorytmu wyczerpującego / siłowego *AW* wprowadzono ograniczenia:

- pojemność plecaka: zakres $[5,\ 20]$.
- liczba przedmiotów: $n = 20 - const$.




Wykres dla algorytmu siłowego przy stałej ilości elementów = 20
505000
500000
495000
490000
485000
480000
475000
Czas [µs]
5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Pojemność plecaka


Wykres dla programowania dynamicznego przy stałej ilości elementów = 100
300
250
200
150
100
50
0
Czas [µs]
5 8 11 14 17 20 23 26 29 32 35 38 41 44 47 50 53 56 59 62 65 68 71 74 77 80 83 86 89 92 95 98
Pojemność plecaka




## 2.3. Według zmiennych 2 parametrów

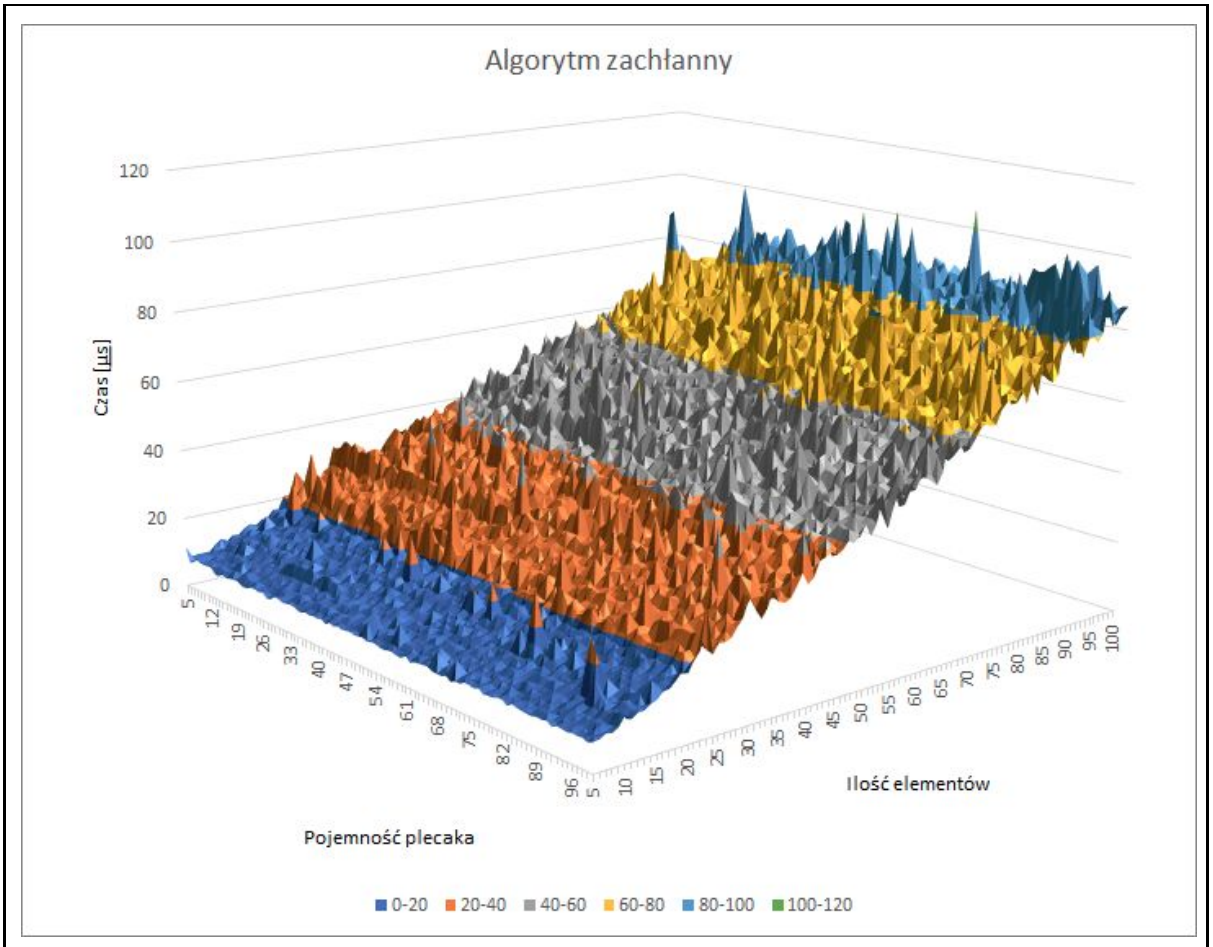
Generowane dane dla funkcji 2 zmiennych $t = f(n, b)$, gdzie $n$ to ilość elementów do wyboru, $b$ - pojemność plecaka - zakresy:

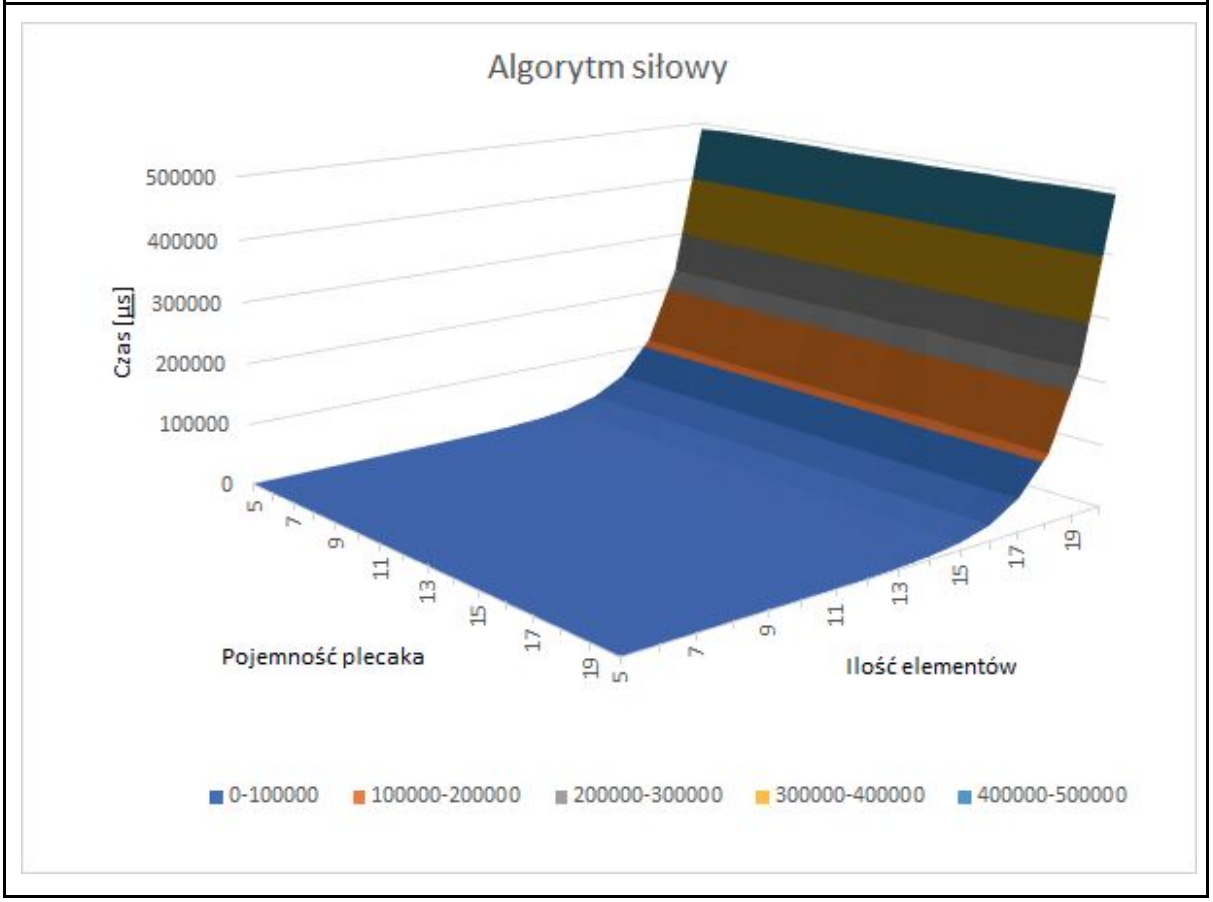
W przypadku algorytmu zachłannego oraz dynamicznego:

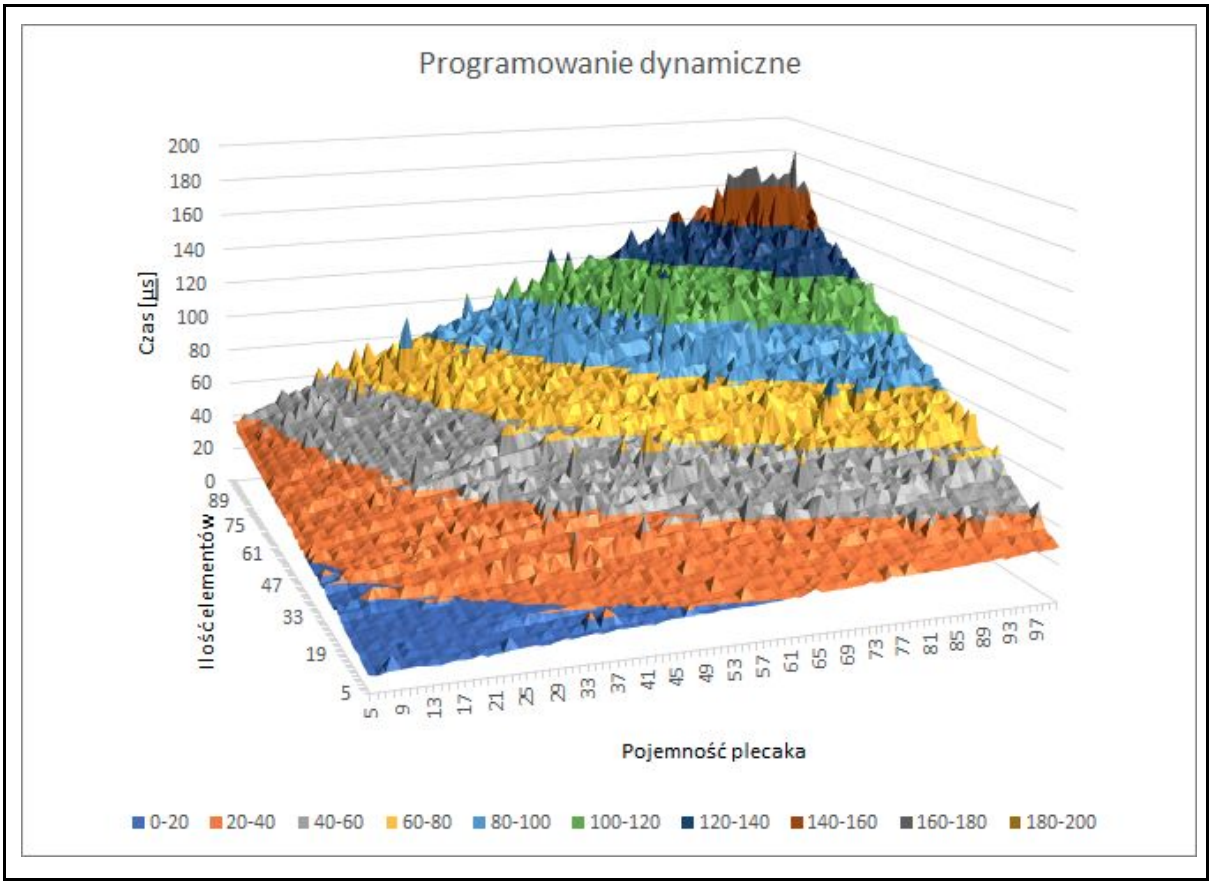
- pojemność plecaka: zakres $[5, 100]$.
- liczba przedmiotów: zakres $[5, 100]$.

*OGRANICZENIE:* W przypadku algorytmu wyczerpującego / siłowego *AW*:

- pojemność plecaka: zakres $[5, 20]$.
- liczba przedmiotów: zakres $[5, 20]$.


Algorytm zachłanny
Czas [µs]
120
100
80
60
40
20
0
Pojemność plecaka
Ilość elementów
0-20
20-40
40-60
60-80
80-100
100-120


Algorytm siłowy
Czas [µs]
500000
400000
300000
200000
100000
0
Pojemność plecaka
Ilość elementów
0-100000
100000-200000
200000-300000
300000-400000
400000-500000




# 3. Wnioski

Celem tego punktu jest przedstawienie odpowiednich wniosków podsumowujących rozwiązywanie 0-1 problemu plecakowego dla każdego z wymienionych algorytmów.

## 3.1. Algorytm zachłanny *AZ*

Informacja: W przypadku algorytmu implementacji algorytmu zachłannego posłużono się wbudowaną funkcją w bibliotekę standardową języka *C++*, *std::sort()*, której złożoność obliczeniowa jest rzędu $O(n \cdot log(n))$. Sortowanie odbywa się po *współczynniku opłacalności* - wartość przedmiotu na jednostkę masy.

Algorytm ten jest bardzo prostym w implementacji *rozwiązaniem* problemu plecakowego - jego sposób działania polega na posortowaniu przedmiotów w plecaku w kolejności nierosnącej, według wspomnianego wyżej współczynnika opłacalności, a następnie liniowego sprawdzenia, który element mieści się jeszcze w plecaku. Waga plecaka w każdej iteracji przechodzenia przez zbiór przedmiotów jest aktualizowana, a możliwość dodania przedmiotu sprawdzania jest poprzez porównanie wagi bieżącego

przedmiotu + wagi plecaka w obecnej iteracji, z całkowitą pojemnością plecaka.

Algorytm zachłanny jest głównie zależny od liczby przedmiotów - ponieważ to właśnie przedmioty mają największy wpływ na jego złożoność, ze względu na ich ówczesne sortowanie. Samo przejrzenia wszystkich przedmiotów jest liniowe i wykonuje się w czasie $O(n).$ W związku z czym całkowita złożoność algorytmu w wynosi $O(n \cdot log(n) + n)$, co w przypadku notacji $O$ upraszcza się do $O(n \cdot log(n))$.

Warty podkreślenia jest fakt, że algorytm zachłanny nie zapewnia znalezienia optymalnego rozwiązania - mimo sortowania po współczynniku opłacalności. Nie ma pewności czy w rozwiązaniu otrzymane zostało globalnie optymalne spakowanie plecaka - jednak zapewnia znalezienie wersji *suboptymalnej*. Dzieję się tak, ponieważ algorytm zachłanny wybiera najlepsze rozwiązanie dla bieżącej iteracji, jednak nie zapewnia to warunku, aby wybór takich *lokalnych najlepszych* rozwiązań był *globalnie* optymalny.

Z powyższych wniosków wynika, że algorytm zachłanny nie daje rozwiazania optymalnego, jeżeli następnikami danego ekstremum lokalnego - najlepszego biężącego rozwiązania - są wartości nieoptymalne, ponieważ wtedy - to bieżące ekstremum jest traktowane jako globalne i zwracane jako rozwiązanie problemu.

### 3.2. Algorytm programowania dynamicznego *APD*

*APD* w całym zestawieniu wydaję się najbardziej logiczną opcją, ponieważ nie tylko znajduję najlepsze rozwiązanie, ale też działa w relatywnie krótkim czasie. Jednak jego złożoność nie należy do prostych - algorytm programowania dynamicznego rośnie w pseudowielomianowym czasie - ze względu na $n$ rośnie wielomianowo, a ze względu na $b$ rośnie wykładniczo. Oznacza to, że dla małych danych algorytm ten jest bardzo szybki - ale, ponieważ jest zależny od 2 zmiennych, dla większej pojemności plecaka może wypaść dużo gorzej.

Ogólna złożoność algorytmu to $O(n \cdot b)$ - co wynika z powyższych wniosków.

Idea algorytmu opiera się o tzw. *własność optymalnej struktury,* która zapewnia, że wybór zawsze najlepszych rozwiązań w bieżącym - mniejszym - problemie doprowadzi nas do optymalnego rozwiązania globalnego. W taki sposób wypełniania jest macierz programowania - najpierw odwołujemy się do problemów mniejszych, dopiero następnie dodając nowy element - dodatkowo zapewniając, że każde z rozwiązań w matrycy jest najbardziej optymalną wersją podproblemu. Następnie mając całą macierz zapełnioną odczytywane jest rozwiązanie problemu - jakie elementy znajdują się w plecaku.

Zdecydowaną wadą algorytmu *PD* jest z pewnością jego złożoność pamięciowa, ponieważ dodatkowo potrzebujemy utworzyć matrycę *PD,* o rozmiarach $n+1$ na $b+1$.

### *3.3.* Algorytm wyczerpujący / siłowy *AW*

Algorytm siłowy jest zdecydowanie najwolniejszym algorytmem z całego zestawienia. Algorytmy siłowe z założenia sprawdzają wszystkie możliwe kombinacje, a więc w tym przypadku algorytm sprawdza wszystkie $2^n - 1$ możliwości i koduje je na bitach liczbowych zmiennej typu *int* w *C++* - co oznacza, że maksymalną liczbą przedmiotów jakie możemy obsłużyć jest liczba 63 elementów. Z czego dla liczb większych od 30 - nie jest to opłacalne czasowo - ponieważ złożoność czasowa algorytmu siłowego jest ***potężna***, bo wykładnicza - $O(2^n)$ .W związku z tym w prezentowanych algorytmach wprowadzono ograniczenie liczbowe 20 możliwych elementów.

Warto nadmienić, że algorytm ten zależny jest tylko od ilości elementów do wyboru, ponieważ rozmiar plecaka nie ma wpływu na złożoność algorytmu.

Zasada działania algorytmu jest bardzo prosta, ponieważ przeszukujemy przestrzeń wszystkich możliwych rozwiązań i sprawdzamy w zależności od ustawionych bitów, który przedmiot znajduję się w plecaku - następnie porównujemy otrzymane wyniki, aż w końcu otrzymamy rozwiązanie, które jest zawsze optymalne - jednak jest to czasowo mało opłacalne, głównie dla większych danych.

# 4. Informacje dodatkowe

*strona github z kodami źródłowymi* -
https://github.com/AdrianMadajewski/DS-Knapsack-problem

*link do arkusza z danymi*
https://1drv.ms/x/s!AmnqNWF6AFGwgZh7KUs3Gkd5VDbeMw?e=BiTmKl

**Platforma testowa:**

| | |
|---|---|
| procesor: | Intel i7 7700k 4.2 GHz |
| ram: | 8 GB RAM DDR4 CL16 |
| karta graficzna: | Geforce GTX 1060 6GB |
| system operacyjny: | Windows 7 64-bit |
| płyta główna: | MSI Z270-A PRO |
| dysk: | HDD 1000 GB |