Sortowanie przez wstawianie – jest to algorytm sortowania, który polega na wstawianiu kolejnych elementów tablicy w odpowiednie miejsce posortowanej już części tablicy. Iterując po sortowanej tablicy, patrzy się czy dany element jest mniejszy czy większy od wcześniejszych, posortowanych elementów tablicy i w zależności od potrzeb użytkownika (czy ma być to tablica posortowana rosnąco czy malejąco) wstawia się dany element w odpowiednie miejsce (czyli od miejsca, gdzie ma się wstawić dany element, przesuwa się posortowane elementy o jeden indeks w prawo, a w to miejsce wstawia się dany element) i przeskakuje się na kolejny element tablicy.

Złożoność obliczeniowa dla najgorszego i średniego przypadku jest identyczna i wynosi O(n2).

Dokładny opis działania programu, dla posortowania rosnącego:

1. Wywoływana jest pętla for, która rozpoczyna się od drugiego elementu tablicy.
2. Następnie do zmiennej pomocniczej zapisywana jest wartość aktualnego elementu.
3. W dalszej części aktualny element tablicy porównywany jest z elementem poprzednim i jeśli jest on od niego mniejszy, aktualny element przyjmuje wartość poprzedniego elementu. Operacja ta powtarzana jest w stronę początku tablicy, aż poprzedni element będzie mniejszy bądź równy aktualnemu elementowi. Na samym końcu na miejsce obecnego elementu przypisujemy wartość ze zmiennej pomocniczej.
4. Następnie przechodzi się do kolejnej iteracji pętli for i powtarza się krok 2 i 3. Powtarza się te czynności aż do ostatniego elementu tablicy.

Sortowanie Shella – jest to algorytm sortowania, który polega na początkowym stworzeniu ciągu elementów według konkretniej funkcji dla danej złożoności obliczeniowej. Elementy te są długościami odstępów między elementami sortowanej tablicy. Następnie sortowana tablica dzielona jest na podzbiór składający się z elementów tablicy odległych o największy odstęp (czyli największy element stworzonego początkowo ciągu). Stworzony podzbiór sortowany jest algorytmem przez wstawianie i dalej tworzone oraz sortowane są po kolei podzbiory, dla coraz to mniejszych odstępów (coraz mniejszych elementów stworzonego ciągu).

Dla funkcji na wyraz ogólny ciągu odstępów, którą stworzył Shell najgorsza złożoność obliczeniowa wynosi O(n2), natomiast dla funkcji na wyraz ogólny ciągu odstępów, którą stworzył Sedgewick złożoność obliczeniowa wynosi O().

Dokładny opis działania programu, dla posortowania rosnącego:

1. Na początku wyliczany jest według wzoru funkcji wyrazu ogólnego największy możliwy odstęp.
2. Następnie w pętli for rozpoczynającej się od wartości danego odstępu, do zmiennej pomocniczej zapisywana jest wartość aktualnego elementu i później porównywane są dwa elementy sortowanej tablicy. Element z indeksem równym odstępowi i elementem z indeksem mniejszym o dany odstęp. Jeśli ten drugi element jest większy, to do elementu pierwszego przypisywana jest wartość elementu drugiego. Sprawdzane jest także aż do końca czy nie znajduje się element tablic z indeksem mniejszym od indeksu drugiego elementu o odstęp. Jeśli istnieje to te kolejne dwa elementy są porównywane i w razie potrzeby zamieniane. Jeśli jednak nie istnieje to w miejsce elementu z najmniejszym istniejącym indeksem dla danego odstępu dla danej iteracji pętli for wpisywana jest wartość zmiennej pomocniczej.
3. W dalszej części przechodzi się do kolejnej iteracji pętli for i powtarza się czynności z kroku drugiego.
4. Jeśli pętla for dojdzie do wartości rozmiaru sortowanej tablicy, wyliczany jest kolejny (mniejszy) odstęp i powtarzane są kroki 2 i 3.
5. Ostatnią wartością odstępu, dla którego przeprowadzany jest proces sortowania jest wartość jeden.

Quicksort – jest to algorytm sortowania, który polega na wykorzystaniu techniki dziel i zwyciężaj. Z sortowanej tablicy na początku wybierany jest pivot, który jest ustalonym elementem tablicy (prawym, lewym, środkowym lub losowym). W dalszej kolejności tablica ustawiana jest w taki sposób, że po lewej stronie pivota znajdują się elementy nie większe od niego, a po jego prawej stronie znajdują się elementy nie mniejsze od niego. Następnie w powstałych dwóch podtablicach wybiera się nowe pivoty i dzieli się je na kolejne podtablice według tego samego schematu. Podziały podtablic następują do momentu, aż każda podtablica będzie miała dokładnie jeden element. Złożoność obliczeniowa dla najgorszego przypadku wynosi O(n2), natomiast dla średniego przypadku wynosi ona O().

Dokładny opis działania programu, dla posortowania rosnącego:

1. Wywoływana jest funkcja quicksort dla wszystkich elementów sortowanej tablicy.
2. W funkcji quicksort na początku wywoływana jest funkcja pivotIndex, w której tablica jest odpowiednio sortowana na elementy nie większe i nie mniejsze od pivota, a następnie zwracana jest wartość indeksu, na którym znajduje się pivot. Na samym początku funkcji pivotIndex wywoływana jest także funkcja pivot, która zwraca odpowiednie wartości (left, right, middle, random) w zależności od tego jaki rodzaj pivota został wybrany przez użytkownika na początku programu.
3. W dalszej części w funkcji quicksort, referencyjnie wywoływane są funkcje quicksort dla dwóch powstałych podtablic (podtablicy z elementami nie większymi od pivota i podtablicy z elementami nie mniejszymi od pivota).
4. Dla kolejnych wywołań funkcji quicksort powtarzane są kroki 2 i 3, aż końcowe podtablice będą zbudowane z dokładnie jednego elementu.

Sortowanie przez kopcowanie – jest to algorytm sortowania, który opiera się na tworzeniu specjalnych drzew binarnych zwanymi kopcami. Na samym początku z nieuporządkowanych elementów tablicy tworzy się kopiec, a następnie zamienia się ostatni liść z korzeniem. W dalszej części algorytmu odcina się ostatni liść i na nowo przywraca się własności kopca, odpowiednio układając tablicę. Proces ten powtarza się aż dojdzie się do ostatniego pozostałego elementu tablicy. Złożoność obliczeniowa dla najgorszego i średniego przypadku jest identyczna i wynosi O().

Dokładny opis działania programu, dla posortowania rosnącego:

1. Wywoływana jest funkcja sortowanie\_przez\_kopcowanie, w której na samym początku wywoływana jest funkcja tworzenie\_kopca. Funkcja ta, idąc od indeksu ostatniego rodzica aż do indeksu korzenia, wywołuje funkcję kopiec.
2. W funkcji kopiec porównywany jest rodzic wraz z jego potomkami. Z porównywanych wartości, największa z nich przypisywana jest do wartości rodzica poprzez zamianę z nim (o ile największej z nich nie ma rodzic).
3. Jeśli nastąpiła zamiana elementów, czyli jeśli któryś z potomków miał wartość większą od wartości rodzica, referencyjnie wywoływana jest funkcja kopiec dla indeksu elementu, który został zamieniony z rodzicem.
4. W dalszej części funkcji sortowanie\_przez\_kopcowanie wywoływana jest pętla for od ostatniego indeksu tablicy, w której na początku zamieniany jest korzeń z ostatnim liściem, a następnie wywoływana jest funkcja kopiec dla indeksu korzenia oraz dla ilości elementów sortowanej tablicy pomniejszonej z każdą iteracją o jeden. Czyli po każdej zamianie korzenia z ostatnim liściem, ostatni liść jest odcinany. Pętla for jest powtarzana aż dojdzie do zerowego indeksu sortowanej tablicy.

### **Podsumowanie:**

Jak widać na podstawie powyższych wykresów oraz tabeli wyników, dla tablic z elementami losowymi oraz posortowanymi w 33%, 66% i malejąco, na podwójny przyrost wielkości tablicy, czas sortowania zwiększa się o około cztery razy. Złożoność obliczeniowa dla algorytmu sortowania przez wstawianie w najgorszym oraz średnim przypadku wynosi O(n2), dlatego dla podwójnego przyrostu wielkości tablicy, maksymalny przyrost czasu powinien wynosić . Otrzymane wyniki są więc zgodne z teoretyczną złożonością obliczeniową badanego algorytmu.

W najlepszym przypadku, jeśli sortowana tablica jest już posortowana rosnąco, złożoność obliczeniowa algorytmu sortowania przez wstawianie wynosi O(n), co oznacza, że dla podwójnego przyrostu wielkości tablicy, przyrost czasu powinien również być podwójny. Na podstawie tabeli wyników oraz wykresu dla tablic posortowanych rosnąco, można zobaczyć, że w przybliżeniu przyrost czasu dla kolejnych wielkości tablic zgadza się z założeniami teoretycznymi.

### **Wnioski:**

Najlepszym przypadkiem algorytmu sortowania przez wstawianie, w którym algorytm jest najbardziej wydajny, jest przypadek, w którym tablice posortowane są rosnąco. Algorytm nie stosuje żadnych zamian i jedynie musi przejść po każdym elemencie sortowanej tablicy, dzięki czemu czas sortowania jest wyraźnie najniższy.

Średnim przypadkiem algorytmu sortowania przez wstawianie, jest przypadek, w którym są posortowane w 33% albo 66% lub są losowe.

Najgorszym przypadkiem algorytmu sortowania przez wstawianie, jest przypadek, w którym tablice posortowane są malejąco. Algorytm musi zamieniać i wstawiać elementy tablicy maksymalną ilość razy, przez co czas sortowania jest najdłuższy w porównaniu z innymi przypadkami sposobu generowania tablicy.

### **Podsumowanie:**

Jak widać na podstawie powyższych wykresów oraz tabeli wyników, dla każdego sposobu generowania tablicy, na podwójny przyrost wielkości tablicy, czas sortowania zwiększa się o około dwa razy. Złożoność obliczeniowa dla algorytmu sortowania przez kopcowanie wynosi O(), dlatego dla podwójnego przyrostu wielkości tablicy, przyrost czasu powinien wynosić minimalnie . Otrzymane wyniki są więc zgodne z teoretyczną złożonością obliczeniową badanego algorytmu.

### **Wnioski:**

Ciężko jednoznacznie ocenić, dla którego przypadku algorytm sortowania przez kopcowanie jest najlepszy, a dla którego najgorszy, ponieważ algorytm ten ma dużą odporność na dane wejściowe i średnie czasy dla wszystkich sposobów generowania tablic są do siebie bardzo zbliżone. Jednak na podstawie tabeli wyników oraz umieszczonych powyżej wykresów, algorytm sortowania przez kopcowanie jest najbardziej wydajny dla tablic posortowanych malejąco i tablic losowych, już trochę mniej wydajny jest dla tablic posortowanych rosnąco i w 66%, a najbardziej pesymistycznym przypadkiem dla tego algorytmu jest sortowanie tablic posortowanych w 33%.

### **Podsumowanie:**

Jak widać na podstawie powyższych wykresów oraz tabeli wyników, w najlepszym przypadku, gdy tablica posortowana jest rosnąco, na podwójny przyrost wielkości tablicy, czas sortowania zwiększa się o około dwa razy. Złożoność obliczeniowa algorytmu sortowania Shella dla najlepszego wypadku wynosi O(n), co oznacza, że dla podwójnego przyrostu wielkości tablicy, przyrost czasu powinien również być podwójny. Otrzymane wyniki są więc zgodne z teoretyczną złożonością obliczeniową badanego algorytmu.

Dla tablic losowych oraz posortowanych malejąco, w 33% i 66% średnie czasy sortowań są bardzo do siebie zbliżone. Ciężko jednak jednoznacznie określić jaki jest średni przyrost czasu sortowania na podwójny przyrost wielkości tablicy. Jednak analizując tabelę wyników oraz powyższe wykresy można zauważyć, że dla tablic losowych oraz posortowanych malejąco, w 33% i 66% dla tablic mających poniżej 20000 elementów czas sortowania zwiększa się o lekko ponad dwukrotnie, dla tablic mających między 20000 a 40000 elementów następuje wzrost około czterokrotny, natomiast dla tablic mających powyżej 40000 elementów czas sortowania także zwiększa się lekko ponad dwukrotnie. Złożoność obliczeniowa dla algorytmu Shella o złożoności O(n2) w najgorszym oraz średnim przypadku wynosi O(n2), dlatego dla podwójnego przyrostu wielkości tablicy, maksymalny przyrost czasu powinien wynosić . Przyrost czasu sortowania na podwójny przyrost wielkości tablicy dla otrzymanych wyników jest zawsze mniejszy od 4, a więc otrzymane wyniki są zgodne z teoretyczną złożonością obliczeniową badanego algorytmu.

### **Wnioski:**

Najlepszym przypadkiem algorytmu sortowania Shella o złożoności O(n2) jest przypadek, w którym tablice posortowane są rosnąco. Algorytm ten jest ulepszonym algorytmem przez wstawianie, dlatego tak samo jak on, dla tablic posortowanych rosnąco nie stosuje on żadnych zamian i jedynie musi po kolei porównywać elementy odległe o odpowiedni odstęp i na koniec musi przejść po każdym elemencie sortowanej tablicy, dzięki czemu czas sortowania jest wyraźnie najniższy.

Średnim przypadkiem sortowania Shella o złożoności O(n2), jest przypadek, w którym tablice są częściowo posortowane (w 33% albo w 66%) lub tablice są losowe.

Najgorszym przypadkiem Shella o złożoności O(n2), jest przypadek, w którym tablice posortowane są malejąco.

Algorytm ten ma dużą odporność na dane wejściowe, dlatego średnie czasy sortowań dla tablic losowych, posortowanych malejąco, w 33% i 66% są do siebie bardzo zbliżone.

### **Podsumowanie:**

Jak widać na podstawie powyższych wykresów oraz tabeli wyników, w najlepszym przypadku, gdy tablica posortowana jest rosnąco, na podwójny przyrost wielkości tablicy, czas sortowania zwiększa się o około dwa razy. Złożoność obliczeniowa algorytmu sortowania Shella dla najlepszego wypadku wynosi O(n), co oznacza, że dla podwójnego przyrostu wielkości tablicy, przyrost czasu powinien również być podwójny. Otrzymane wyniki są więc zgodne z teoretyczną złożonością obliczeniową badanego algorytmu.

Analizując tabelę wyników oraz powyższe wykresy można zauważyć dla tablic losowych oraz posortowanych malejąco, w 33% i 66% średnie czasy sortowań są bardzo do siebie zbliżone. Na podwójny przyrost wielkości tablicy, czas sortowania zwiększa się około lekko ponad dwukrotnie. Złożoność obliczeniowa dla algorytmu Shella o złożoności O(n4/3) w najgorszym oraz średnim przypadku wynosi O(n4/3), dlatego dla podwójnego przyrostu wielkości tablicy, maksymalny przyrost czasu powinien wynosić . Przyrost czasu sortowania na podwójny przyrost wielkości tablicy dla otrzymanych wyników jest zawsze mniejszy od 2.5, a więc otrzymane wyniki są zgodne z teoretyczną złożonością obliczeniową badanego algorytmu.

### **Wnioski:**

Najlepszym przypadkiem algorytmu sortowania Shella o złożoności O(n4/3) jest przypadek, w którym tablice posortowane są rosnąco. Algorytm ten jest ulepszonym algorytmem przez wstawianie, dlatego tak samo jak on, dla tablic posortowanych rosnąco nie stosuje on żadnych zamian i jedynie musi po kolei porównywać elementy odległe o odpowiedni odstęp i na koniec musi przejść po każdym elemencie sortowanej tablicy, dzięki czemu czas sortowania jest wyraźnie najniższy.

Średnim przypadkiem sortowania Shella o złożoności O(n4/3), jest przypadek, w którym tablice są częściowo posortowane (w 33% albo w 66%) lub tablice są losowe.

Najgorszym przypadkiem sortowania Shella o złożoności O(n4/3) jest przypadek, w którym tablice posortowane są malejąco.

Algorytm ten ma dużą odporność na dane wejściowe, dlatego średnie czasy sortowań dla tablic losowych, posortowanych malejąco, w 33% i 66% są do siebie bardzo zbliżone.

### **Podsumowanie:**

Jak widać na podstawie powyższych wykresów oraz tabeli wyników, dla tablic losowych oraz posortowanych w 33% i w 66%, dla tablic zawierających poniżej 100000 elementów średnie czasy sortowań są bardzo do siebie zbliżone. Dla tablic z większą ilością elementów, powyżej 100000 czasy zaczynają lekko się od siebie rozsuwać, jednak nadal są do siebie bardzo podobne. Na podwójny przyrost wielkości tablicy, średni czas sortowania wzrasta około dwukrotnie – dla mniejszych wielkości tablic lekko ponad 2 razy i idąc po kolei, dla coraz większych tablic przyrost czasu zbliża się coraz bardziej do 2. Zgadza się to z założeniami teoretycznymi, ponieważ algorytm quicksort dla średniego oraz najlepszego przypadku ma złożoność obliczeniową O(), czyli dla podwójnego przyrostu wielkości tablicy, dla coraz to większych tablic maksymalny przyrost czasu sortowania dąży do .

Jeśli pivot jest skrajną wartością, to tablice posortowane rosnąco i malejąco są pesymistycznymi przypadkami dla algorytmu sortowania quicksort ze skrajnie lewym pivotem. Przyrost średniego czasu sortowania zgadza się z teoretycznymi założeniami, ponieważ wacha się on pomiędzy 2 a 4, a dla najgorszego przypadku algorytm quicksort ma złożoność obliczeniową O(n2), co oznacza, że maksymalny przyrost średniego czasu sortowania na podwójny przyrost wielkości tablicy wynosi , co pokrywa się z otrzymanymi wynikami.

Jak widać na podstawie otrzymanych wyników, dla tablic posortowanych rosnąco z większą ilością elementów, program automatycznie się wyłączał. Działo się tak dlatego, że występowało zjawisko przepełnienia stosu, ponieważ rekurencyjne wywołanie funkcji quicksort było powtarzane maksymalną ilość razy i stos, który przechowywał informacje o bieżących wywołaniach funkcji przepełniał się powodując automatyczne zatrzymanie działania programu.

### **Wnioski:**

Ciężko jednoznacznie ocenić, który przypadek jest najlepszym przypadkiem dla algorytmu sortowania quicksort ze skrajnie lewym pivotem. Jednak analizując otrzymane wyniki, widać, że tablice losowe są optymalnym przypadkiem dla badanego algorytmu, natomiast tablice częściowo posortowane (w 33% i w 66%) są przypadkiem średnim.

Pesymistycznym przypadkiem sortowania quicksort ze skrajnie lewym pivotem jest przypadek, w którym tablice posortowane są rosnąco lub malejąco. Dodatkowo dla tablic posortowanych rosnąco, występuje przepełnienie stosu, w wyniku czego dla większych wielkości tablic działanie algorytmu automatycznie się przerywa.