Projekt
Algorytmy i struktury danych

Autor: Hubert Stasiowski

Inżynieria i analiza danych, 1 FS0-DI, P01

Numer albumu:166703

Spis treści

1.Temat

Zaimplementuj sortowanie przez gnoma oraz sortowanie kopcowe.

Sortowanie polega na uporządkowaniu zbioru danych według pewnych kryteriów. Pojęcie to stosuje się w celu czytelniejszej prezentacji danych poprzez ich uporządkowanie bądź zmodyfikowania algorytmów na bardziej wydajne. Niekiedy z pozoru wolniejsze algorytmy okazują się lepszym rozwiązaniem z powodu np. uporządkowanej już tablicy, bądź też jej długości.

2.Teoria

2.1 Czasowa złożoność obliczeniowa

Przyjętą miarą złożoności czasowej jest liczba operacji podstawowych w zależności od rozmiaru wejścia. Pomiar rzeczywistego czasu zegarowego jest mało użyteczny ze względu na silną zależność od sposobu realizacji algorytmu, użytego kompilatora oraz maszyny, na której algorytm wykonujemy. Dlatego w charakterze czasu wykonania rozpatruje się zwykle liczbę operacji podstawowych (dominujących). Operacjami podstawowymi mogą być na przykład: podstawienie, porównanie lub prosta operacja arytmetyczna.

2.2 Sortowanie gnoma

Sortowanie Gnoma to metoda sortowania danych, która wymaga jedynie jednej pętli. Implementacja częściowo opiera się na pomyśle z sortowania bąbelkowego. Złożoność tego algorytmu to $\Theta(n_2)$. W najlepszym przypadku złożoność może być liniowa. Algorytm wymaga tylko zmiennej do zapamiętania aktualnej pozycji sortowania, więc złożoność pamięciowa to $\Theta(1)$.

Sortowanie rozpoczyna się od pierwszego elementu na liście. Jeśli aktualnie rozpatrywany element jest pierwszym elementem na liście, albo spełnia warunki posortowania to należy zwiększyć numer indeksu. W przeciwnym wypadku należy zamienić aktualny element z poprzednim i zmniejszyć indeks o 1.

2.3 Przykład:

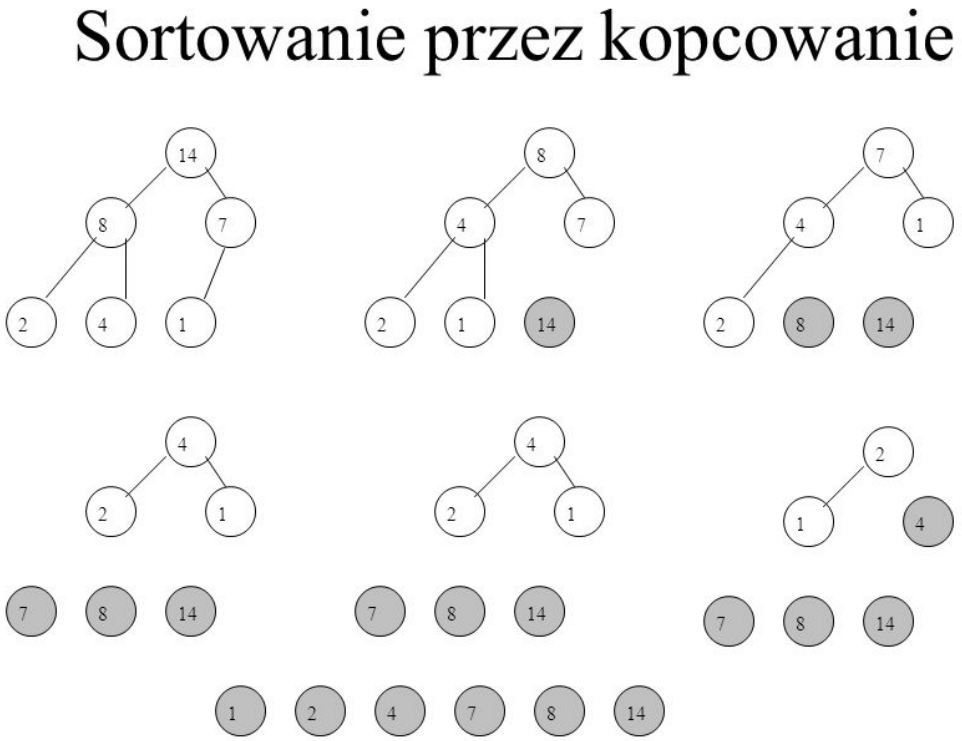
Weźmy przykładowo listę L:=[1, 5, 2, 4, 3]. Początkowy indeks *pos* = *0* (wskazuje na pierwszy element na liście). Lista będzie sortowana rosnąco:

pos	**Lista**	**Warunki**	**Akcja**
0	[**1**, 5, 2, 4, 3]	indeks wynosi 0	zwiększenie *pos* o 1
1	[1, **5**, 2, 4, 3]	element 5 ≥ 1	zwiększenie *pos* o 1
2	[1, 5, **2**, 4, 3]	element 2 < 5	zamiana 2 z 5, zmniejszenie *pos* o 1
1	[1, **2**, 5, 4, 3]	element 2 ≥ 1	zwiększenie *pos* o 1
2	[1, 2, **5**, 4, 3]	element 5 ≥ 2	zwiększenie *pos* o 1
3	[1, 2, 5, **4**, 3]	element 4 < 5	zamiana 4 z 5, zmniejszenie *pos* o 1
2	[1, 2, **4**, 5, 3]	element 4 ≥ 2	zwiększenie *pos* o 1
3	[1, 2, 4, **5**, 3]	element 5 ≥ 4	zwiększenie *pos* o 1
4	[1, 2, 4, 5, **3**]	element 3 < 5	zamiana 3 z 5, zmniejszenie *pos* o 1
3	[1, 2, 4, **3**, 5]	element 3 < 4	zamiana 3 z 4, zmniejszenie *pos* o 1
2	[1, 2, **3**, 4, 5]	element 3 ≥ 2	zwiększenie *pos* o 1
3	[1, 2, 3, **4**, 5]	element 4 ≥ 3	zwiększenie *pos* o 1
4	[1, 2, 3, 4, **5**]	element 5 ≥ 4	zwiększenie *pos* o 1
5	[1, 2, 3, 4, 5]	*pos* = długość listy	Lista została posortowana

2.4 Sortowanie kopcowe

Sortowanie przez kopcowanie: jeden z algorytmów sortowania, choć niestabilny, to jednak szybki i niepochłaniający wiele pamięci (złożoność czasowa wynosi **O(n log n)** a pamięciowa – przy czym jest to rozmiar sortowanych danych, złożoność pamięciowa **O(n)** dodatkowych struktur wynosi **O(1)** jest to zatem algorytm sortowania *w miejscu*). Jest on w praktyce z reguły nieco wolniejszy od sortowania szybkiego, lecz ma lepszą pesymistyczną złożoność czasową (przez co jest odporny np. na atak za pomocą celowo spreparowanych danych, które spowodowałyby jego znacznie wolniejsze działanie)

2.5 Przykład:



3 Pseudokod

3.1 Sortowanie gnoma

```
SortowanieGnoma (int *lista, int n):
 start = 0
 koniec = 0
   while (start < n):

      if (start == 0 or lista[start] >= lista[start - 1]):

         start = koniec + 1
         koniec = start
      else:
         swap(lista[start], lista[start - 1])
         start = start - 1
```

3.2 Kopcowanie

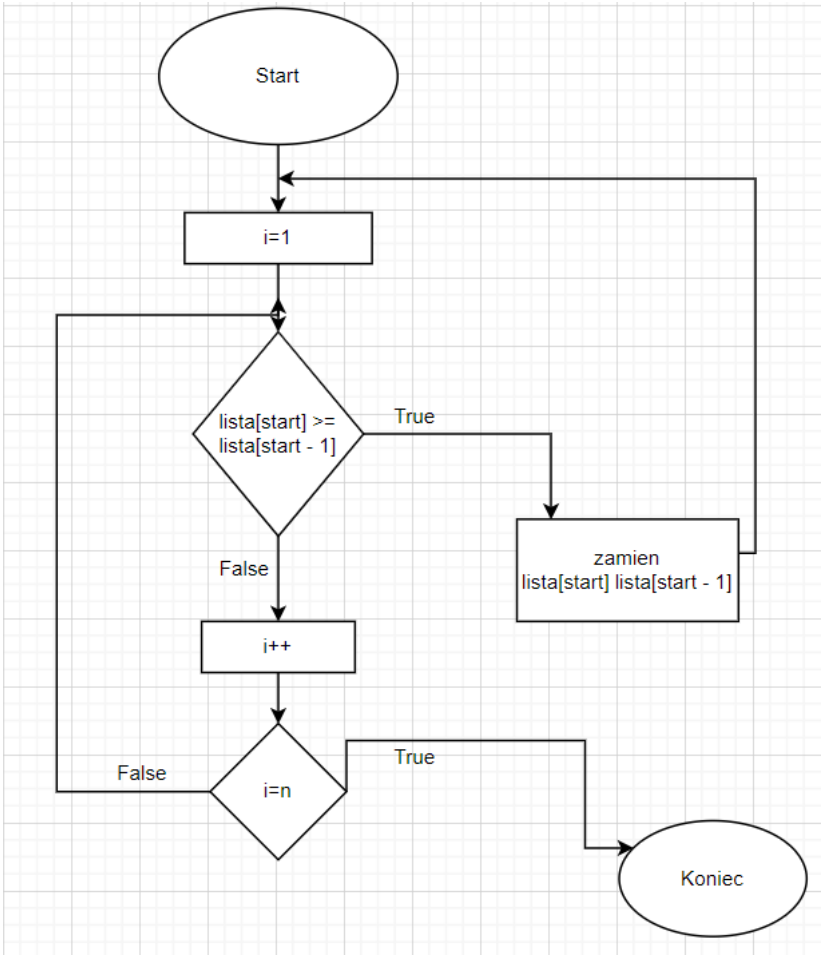
```
kopcowanie(int *tab, int n, int i)

najwiekszy = i
   l = 2 * i + 1
    p = 2 * i + 2

   if (l < n  i tab[l] > tab[najwiekszy]):
      najwiekszy = l

   if (p < n i tab[p] > tab[najwiekszy]):
      najwiekszy = p

   if (najwiekszy != i):

      swap(tab[i], tab[najwiekszy])

      kopcowanie(tab, n, najwiekszy)
```

3.3 Sortowanie kopcowe

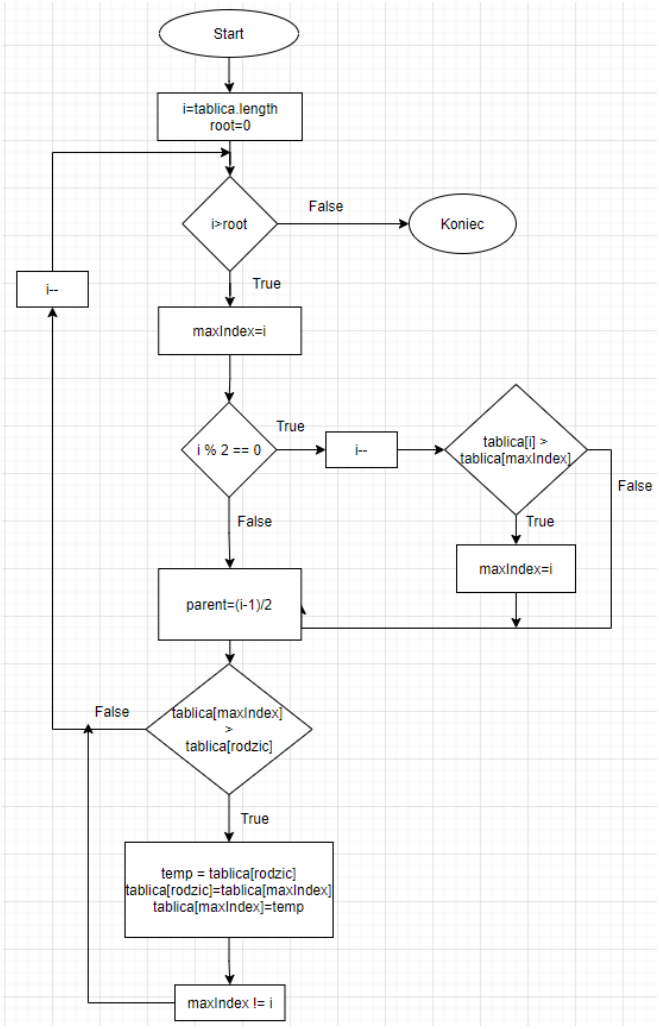
```
sortowanieKopcowe(int* tab, int n)

   for (licznik = n / 2 – 1 to i >= 0 do licznik--):
      kopcowanie(tab, n, i)
   for (licznik = n – 1 to i > 0 do licznik--):
      swap(tab[0], tab[i])
      kopcowanie(tab, i, 0)
```

4. Schemat blokowy

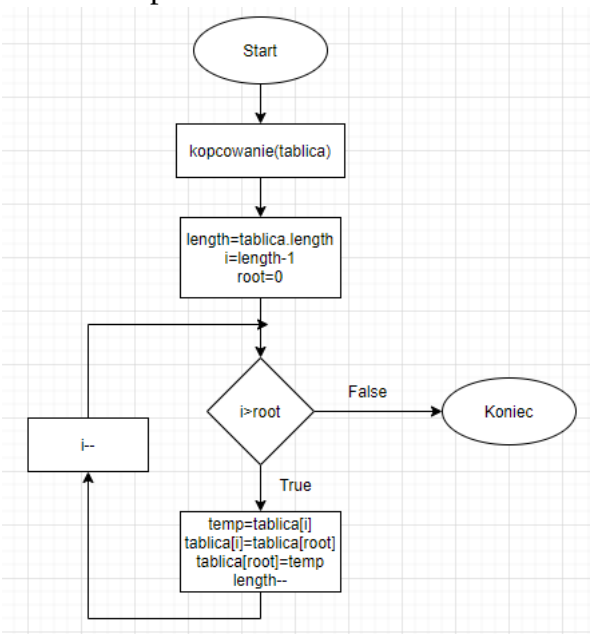
4.1 Sortowanie gnoma



4.2 Kopcowanie



4.3 Sortowanie kopcowe



5.Działanie programu

Po wpisaniu 1 do konsoli posortuje się losowa tablica o wielkości podanej przez użytkownika.
Ponad to program poda czas obu sortowań i zapisze tablice oraz wyniki to pliku tekstowego.

```
Wybierz opcje od 1-5.
Opcja 1, to sortowanie oczekiwane.
Opcja 2, to sortowanie z pliku gnoma.
Opcja 3, to sortowanie z pliku kopcowe.
Opcja 4, to sortowanie pesymistyczne.
Opcja 5, to sortowanie optymistyczne.
1
wynik sortowania znajduje sie w pliku wyniki_testow_oczekiwane.txt

Press any key to continue . . .
```

Po wpisaniu 2 będziemy mieć możliwość posortowania tablicy za pomocą sortowania gnoma z wybranego pliku tekstowego posiadającego liczby wpisane po enterze.
Tablica zostanie pokazana w konsoli i zapisana w pliku.

```
Wybierz opcje od 1-5.
Opcja 1, to sortowanie oczekiwane.
Opcja 2, to sortowanie z pliku gnoma.
Opcja 3, to sortowanie z pliku kopcowe.
Opcja 4, to sortowanie pesymistyczne.
Opcja 5, to sortowanie optymistyczne.
2
Plik powinien zawierac liczby oddzielone enterem!
Podaj nazwe pliku lub jego sciezke: dane.txt


Posortowana tablica:
0 2 2 2 2 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 7 7 7 7 7 7 7 7 7
7 7 8 8 8 8 8 9 9 12 23 23 23 23 23 32 32 32 32 32 32 32 32 32 32 32 32 34 34 34 34 34 34 34 34 34 34 34 34 35 36 37 4
3 43 45 45 45 45 45 45 45 45 45 46 46 46 47 48 52 53 54 54 54 54 54 56 56 56 57 57 59 63 64 64 65 65 65 65 67 67 72 7
4 74 74 76 83 85 86 124 125 235 237 237 237 237 237 237 237 252 264 325 325 325 325 325 326 326 327 327 327 327 346 346
347 348 358 358 387 426 426 435 436 436 437 437 437 456 457 458 458 458 459 462 475 532 532 532 546 546 547 623 624 632
634 634 634 634 643 645 657 658 732 735 745 745 763 783 795 834 834 845 845 845 845 865 865 865 875 945 965 965 967 3267
 3458 4426 5327 5476 6326 6326 6345 6346 6357 6732 7346 7348 8457 32426 34632 45856 54965 64334 345274 8745845
Sortowanie zakonczone. Wyniki zostaly zapisane w pliku: wyniki_z_pliku_gnoma.txt

Process returned 0 (0x0)   execution time : 2.751 s
Press any key to continue.
```

Po wpisaniu 3 będziemy mieć możliwość posortowania tablicy za pomocą sortowania kopcowego z wybranego pliku tekstowego posiadającego liczby wpisane po enterze.
Tablica zostanie pokazana w konsoli i zapisana w pliku.

```
Wybierz opcje od 1-5.
Opcja 1, to sortowanie oczekiwane.
Opcja 2, to sortowanie z pliku gnoma.
Opcja 3, to sortowanie z pliku kopcowe.
Opcja 4, to sortowanie pesymistyczne.
Opcja 5, to sortowanie optymistyczne.
3
Plik powinien zawierac liczby oddzielone enterem!
Podaj nazwe pliku lub jego sciezke: dane.txt


Posortowana tablica sortowaniem kopcowym:
0 2 2 2 2 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
6 6 6 6 6 7 7 7 7 7 7 7 7 7 7 7 7 7 7 8 8 8 8 8 8 8 8 8 8 9 9 9 9 9 9 12 12 12 21 23 23 23 23 23 23 23 23 32 32 32 32
 32 32 32 32 32 32 32 32 32 32 32 32 34 34 34 34 34 34 34 34 34 34 34 34 34 34 35 35 35 36 36 36 37 37 43 43 43 45 45 45
 45 45 45 45 45 45 45 45 45 45 45 45 46 46 46 47 47 48 49 49 52 52 53 53 54 54 54 54 54 54 54 54 56 56 56 56 57 57 59 62
 63 64 64 65 65 65 65 65 67 67 68 68 72 73 73 74 74 74 75 76 76 83 85 86 94 124 125 216 235 237 237 237 237 237 237 237
252 264 325 325 325 325 325 325 325 326 326 326 327 327 327 327 327 327 346 346 347 348 358 358 387 412 426 426 435 436
436 436 437 437 437 456 457 458 458 458 458 458 459 462 475 523 532 532 532 538 546 546 546 547 548 549 623 624 632 632
634 634 634 634 634 643 643 645 657 658 679 732 735 745 745 763 783 795 834 834 845 845 845 845 845 865 865 865 875 945
945 965 965 965 967 3267 3458 3532 4426 5125 5327 5476 6326 6326 6345 6346 6357 6732 6732 7346 7348 8457 23532 32426 346
32 45856 54965 64334 75473 94598 345274 8745845
Sortowanie zakonczone. Wyniki zostaly zapisane w pliku: wyniki_z_pliku_kopcowe.txt

Process returned 0 (0x0)   execution time : 4.942 s
Press any key to continue.
```

Po wpisaniu 4 mamy możliwość wybrania ilości elementów w tablicy, która będzie ułożona malejąco (pesymistycznie).
Ponad to program poda czas obu sortowań i zapisze tablice oraz wyniki to pliku tekstowego.

```
Wybierz opcje od 1-5.
Opcja 1, to sortowanie oczekiwane.
Opcja 2, to sortowanie z pliku gnoma.
Opcja 3, to sortowanie z pliku kopcowe.
Opcja 4, to sortowanie pesymistyczne.
Opcja 5, to sortowanie optymistyczne.
4
Podaj rozmiar tablicy: 200000
Wyniki zostaly zapisane w pliku: wyniki_testow_pesymistyczny.txt

Process returned 0 (0x0)   execution time : 6.522 s
Press any key to continue.
```

Po wpisaniu 5 mamy możliwość wybrania ilości elementów w tablicy, która będzie ułożona rosnąco (optymistycznie).
Ponad to program poda czas obu sortowań i zapisze tablice oraz wyniki to pliku tekstowego.

```
Wybierz opcje od 1-5.
Opcja 1, to sortowanie oczekiwane.
Opcja 2, to sortowanie z pliku gnoma.
Opcja 3, to sortowanie z pliku kopcowe.
Opcja 4, to sortowanie pesymistyczne.
Opcja 5, to sortowanie optymistyczne.
5
Podaj rozmiar tablicy: 20000
Sortowanie zakonczone. Wyniki zostaly zapisane w pliku: wyniki_testow_optymistyczny.txt

Process returned 0 (0x0)   execution time : 2.188 s
Press any key to continue.
```

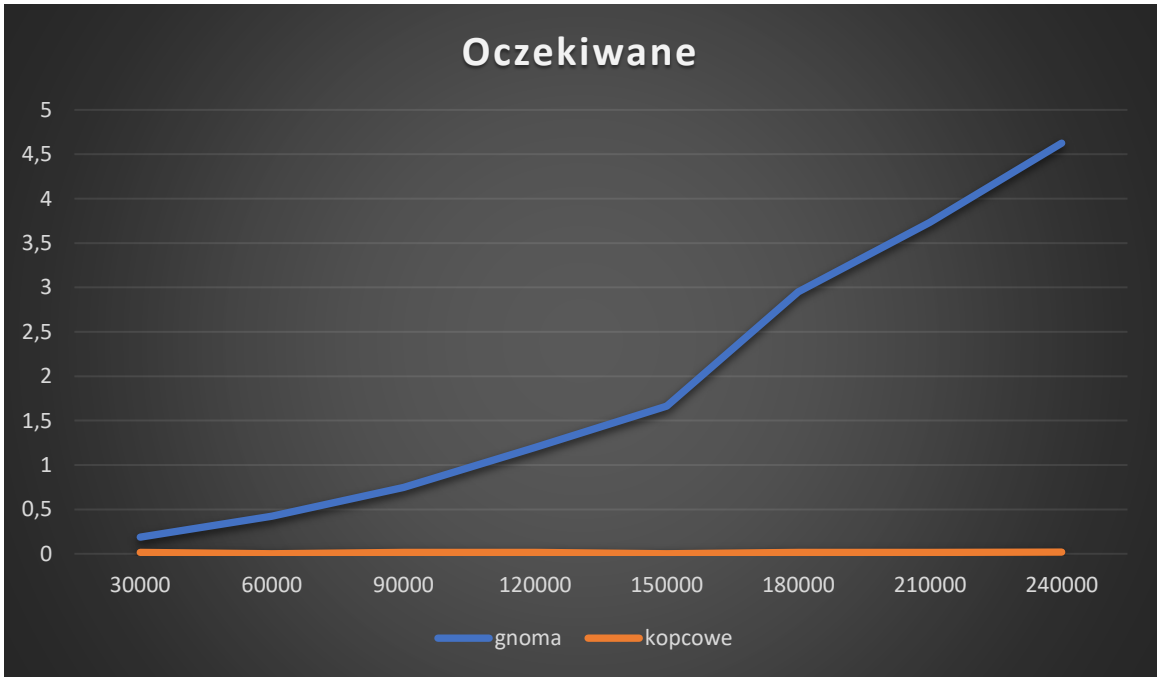
Po wpisaniu jakiejkolwiek innej liczby niż podane w legendzie, bądź też litery pojawi się komunikat:

```
Wybierz opcje od 1-5.
Opcja 1, to sortowanie oczekiwane.
Opcja 2, to sortowanie z pliku gnoma.
Opcja 3, to sortowanie z pliku kopcowe.
Opcja 4, to sortowanie pesymistyczne.
Opcja 5, to sortowanie optymistyczne.
gwkejgkwegwoegj
Nie ma takiej opcji
Process returned 0 (0x0)   execution time : 2.765 s
Press any key to continue.
```

6.Testy

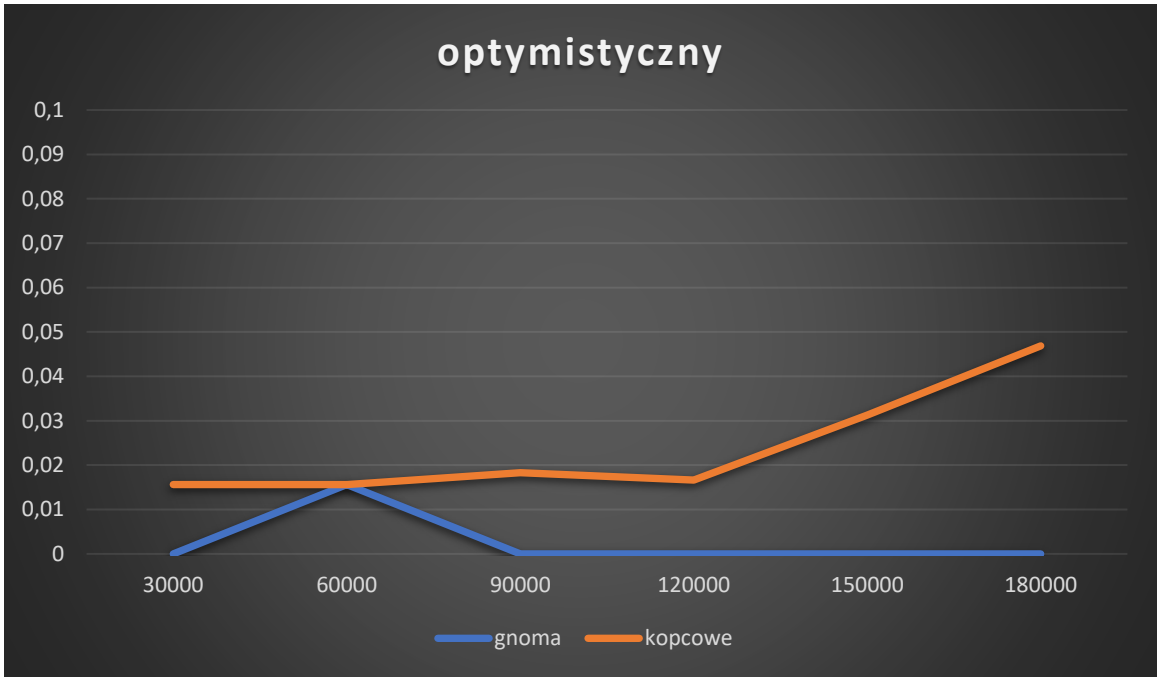
a) Oczekiwane: zestaw licz losowych.
Wynik: sortowanie kopcowe radzi sobie dużo lepiej od gnoma, zwłaszcza przy większej ilości liczb.

```
7707 7707 7707 7707 7707 7708 7710 7712 7712 7714 7714 7715 7716 7718 7719 7723 7723 7724 7724
7932 7932 7933 7933 7934 7934 7936 7937 7939 7939 7939 7939 7941 7942 7942 7943 7944 7948 7948
8163 8163 8165 8166 8167 8168 8169 8169 8171 8171 8172 8173 8174 8175 8175 8176 8176 8177 8177
8379 8380 8384 8385 8385 8387 8387 8390 8390 8390 8392 8395 8397 8397 8397 8402 8404 8406 8406
8611 8611 8611 8612 8612 8614 8615 8615 8617 8617 8618 8618 8618 8618 8618 8619 8621 8622 8626
8825 8826 8826 8828 8829 8829 8830 8832 8832 8834 8834 8835 8835 8835 8836 8844 8845 8845 8847
9034 9034 9035 9035 9038 9040 9041 9044 9044 9044 9047 9047 9048 9050 9051 9051 9052 9052 9055
9232 9234 9235 9235 9236 9238 9241 9243 9244 9248 9249 9250 9251 9251 9253 9254 9254 9256 9257
9450 9451 9452 9452 9452 9453 9455 9455 9456 9458 9458 9460 9461 9462 9462 9465 9465 9466 9467
9678 9682 9682 9686 9687 9688 9688 9689 9690 9690 9691 9692 9692 9692 9696 9697 9697 9701 9703
9896 9898 9899 9900 9900 9901 9901 9902 9903 9905 9905 9909 9909 9910 9912 9913 9913 9913 9914
Posortowanie gnoma zajelo: 0.187217s
Posortowanie kopcowe zajelo: 0.01561s
```



b) Złożoność optymistyczna - najkorzystniejszy zestaw danych

Najkorzystniejszą sytuacją w sytuacji sortowania danych rosnąco jest posortowanie ciągu rosnącego.

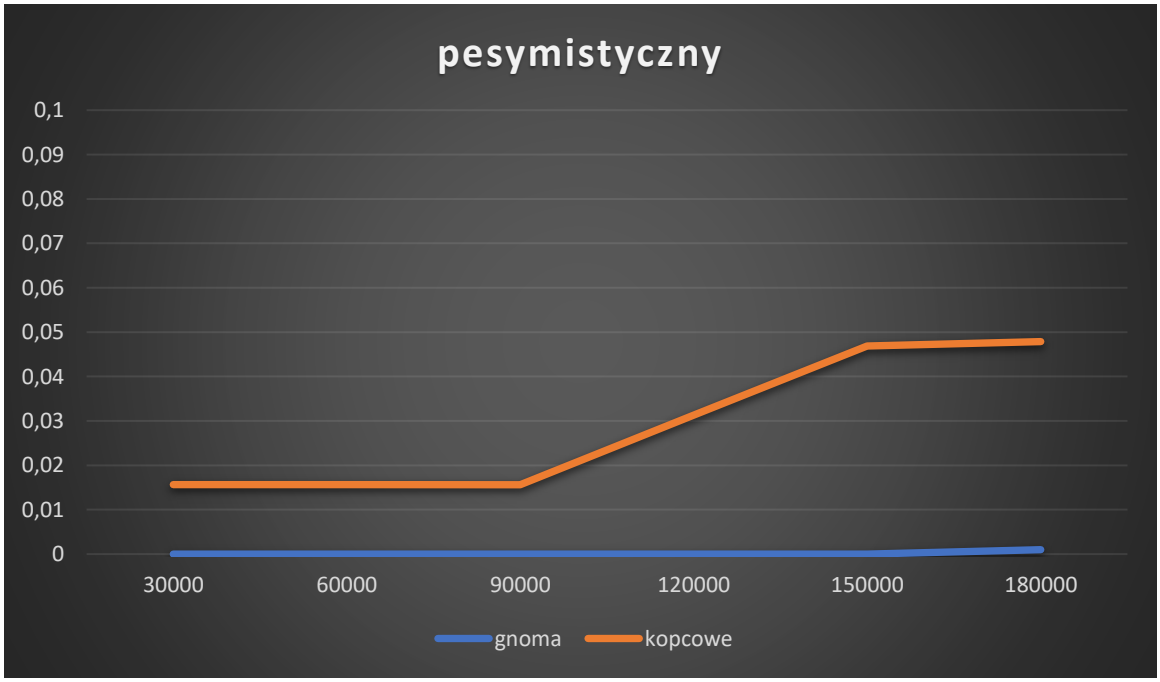



```
117992 117993 117994 117995 117996 117997 117998 117999 118000 118001 118002 118003 118004 118005 118006 11800
8138 118139 118140 118141 118142 118143 118144 118145 118146 118147 118148 118149 118150 118151 118152 118153
84 118285 118286 118287 118288 118289 118290 118291 118292 118293 118294 118295 118296 118297 118298 118299 11
118431 118432 118433 118434 118435 118436 118437 118438 118439 118440 118441 118442 118443 118444 118445 11844
8577 118578 118579 118580 118581 118582 118583 118584 118585 118586 118587 118588 118589 118590 118591 118592
23 118724 118725 118726 118727 118728 118729 118730 118731 118732 118733 118734 118735 118736 118737 118738 11
118870 118871 118872 118873 118874 118875 118876 118877 118878 118879 118880 118881 118882 118883 118884 11888
9016 119017 119018 119019 119020 119021 119022 119023 119024 119025 119026 119027 119028 119029 119030 119031
62 119163 119164 119165 119166 119167 119168 119169 119170 119171 119172 119173 119174 119175 119176 119177 11
119309 119310 119311 119312 119313 119314 119315 119316 119317 119318 119319 119320 119321 119322 119323 11932
9455 119456 119457 119458 119459 119460 119461 119462 119463 119464 119465 119466 119467 119468 119469 119470
01 119602 119603 119604 119605 119606 119607 119608 119609 119610 119611 119612 119613 119614 119615 119616 11
119748 119749 119750 119751 119752 119753 119754 119755 119756 119757 119758 119759 119760 119761 119762 11976
9894 119895 119896 119897 119898 119899 119900 119901 119902 119903 119904 119905 119906 119907 119908 119909
Posortowanie gnoma zajelo: 0.000998s
Posortowanie kopcowe zajelo: 0.030904s
```

c) Złożoność pesymistyczna - najbardziej niekorzystny zestaw danych

Najbardziej niekorzystną sytuacją w przypadku sortowania danych rosnąco jest posortowanie ciągu malejącego.

```
175940 175941 175942 175943 175944 175945 175946 175947 175948 175949 175950 175951 175952 175953 175954 175955 175956 175957
6086 176087 176088 176089 176090 176091 176092 176093 176094 176095 176096 176097 176098 176099 176100 176101 176102 176103 176
32 176233 176234 176235 176236 176237 176238 176239 176240 176241 176242 176243 176244 176245 176246 176247 176248 176249 1762
176379 176380 176381 176382 176383 176384 176385 176386 176387 176388 176389 176390 176391 176392 176393 176394 176395 176396
6525 176526 176527 176528 176529 176530 176531 176532 176533 176534 176535 176536 176537 176538 176539 176540 176541 176542 176
71 176672 176673 176674 176675 176676 176677 176678 176679 176680 176681 176682 176683 176684 176685 176686 176687 176688 1766
176818 176819 176820 176821 176822 176823 176824 176825 176826 176827 176828 176829 176830 176831 176832 176833 176834 176835
6964 176965 176966 176967 176968 176969 176970 176971 176972 176973 176974 176975 176976 176977 176978 176979 176980 176981 176
10 177111 177112 177113 177114 177115 177116 177117 177118 177119 177120 177121 177122 177123 177124 177125 177126 177127 1771
177257 177258 177259 177260 177261 177262 177263 177264 177265 177266 177267 177268 177269 177270 177271 177272 177273 177274
7403 177404 177405 177406 177407 177408 177409 177410 177411 177412 177413 177414 177415 177416 177417 177418 177419 177420 177
49 177550 177551 177552 177553 177554 177555 177556 177557 177558 177559 177560 177561 177562 177563 177564 177565 177566 1775
177696 177697 177698 177699 177700 177701 177702 177703 177704 177705 177706 177707 177708 177709 177710 177711 177712 177713
7842 177843 177844 177845 177846 177847 177848 177849 177850 177851 177852 177853 177854 177855 177856 177857 177858 177859 177
88 177989 177990 177991 177992 177993 177994 177995 177996 177997 177998 177999 178000 178001 178002 178003 178004 178005 1780
178135 178136 178137 178138 178139 178140 178141 178142 178143 178144 178145 178146 178147 178148 178149 178150 178151 178152
8281 178282 178283 178284 178285 178286 178287 178288 178289 178290 178291 178292 178293 178294 178295 178296 178297 178298 178
27 178428 178429 178430 178431 178432 178433 178434 178435 178436 178437 178438 178439 178440 178441 178442 178443 178444 1784
178574 178575 178576 178577 178578 178579 178580 178581 178582 178583 178584 178585 178586 178587 178588 178589 178590 178591
8720 178721 178722 178723 178724 178725 178726 178727 178728 178729 178730 178731 178732 178733 178734 178735 178736 178737 178
66 178867 178868 178869 178870 178871 178872 178873 178874 178875 178876 178877 178878 178879 178880 178881 178882 178883 1788
179013 179014 179015 179016 179017 179018 179019 179020 179021 179022 179023 179024 179025 179026 179027 179028 179029 179030
9159 179160 179161 179162 179163 179164 179165 179166 179167 179168 179169 179170 179171 179172 179173 179174 179175 179176 179
05 179306 179307 179308 179309 179310 179311 179312 179313 179314 179315 179316 179317 179318 179319 179320 179321 179322 1793
179452 179453 179454 179455 179456 179457 179458 179459 179460 179461 179462 179463 179464 179465 179466 179467 179468 179469
9598 179599 179600 179601 179602 179603 179604 179605 179606 179607 179608 179609 179610 179611 179612 179613 179614 179615 179
44 179745 179746 179747 179748 179749 179750 179751 179752 179753 179754 179755 179756 179757 179758 179759 179760 179761 1797
179891 179892 179893 179894 179895 179896 179897 179898 179899 179900 179901 179902 179903 179904 179905 179906 179907 179908

Posortowanie gnoma zajelo: 0s

Posortowanie kopcowe zajelo: 0.046858s
```



7.Bibliografia

Patrząc na omówione powyżej algorytmy, można zauważyć, że algorytm kopcowania działa lepiej, gdy liczby są losowo porozrzucane po tablicy i im jest ich więcej tym większa jest różnica co do algorytmu gnoma.

Porównując oba algorytmy sortowania można stwierdzić, że nie znajdziemy jednego idealnego rozwiązania. Algorytm należy dobrać do postawionego problemu.

Znając algorytmy sortowania można w szybki sposób poukładać dane w zamierzony przez nas sposób. Ważne jest jednak, aby odpowiednio dobrać metodę jak i również zwrócić uwagę na zapotrzebowanie danego sortowania na pamięć, gdyż za szybkość algorytmu sortowania najczęściej płacimy pamięcią, więc trzeba na to uważać.

8.Bibliografia

[1] ***„Wikipedia,” [Online]. Available: https://pl.wikipedia.org/wiki/Sortowanie_przez_kopcowanie.***

[2] ***„Wikipedia,” [Online]. Available: https://pl.wikipedia.org/wiki/Sortowanie_gnoma.***