

Sprawozdanie z projektu

Klaudia Zawiaślak

Algorytmy i struktury danych

nr indeksu 166719

Inżynieria i Analiza Danych

Semestr 1, Grupa nr 8

1.Wstęp.

Niniejsze sprawozdanie przedstawia wykonanie projektu z przedmiotu „Algorytmy i struktury danych”. Zawiera w sobie opis, dokładną treść zadania oraz sposób w jaki je wykonałam. Posiada także pseudokod oraz schemat blokowy. Zamieściłam przykładowe testy programu i moje wnioski, opisałam także kilka funkcji z których tworzeniem zмагаłam się nieco dłużej. Celem projektu było dokonanie implementacji kodu, który rozwiązuje zadanie. Implementacja jest to proces pisania programu (kodu źródłowego), czyli programowanie, lub efekt takiego procesu, czyli program. Kod został napisany w programie Codeblocks, jest to wieloplatformowe, zintegrowane środowisko programistyczne na licencji GNU, oparte na projekcie Scintilla. Program jest napisany w języku C++.

2.Treść zadania do wykonania.

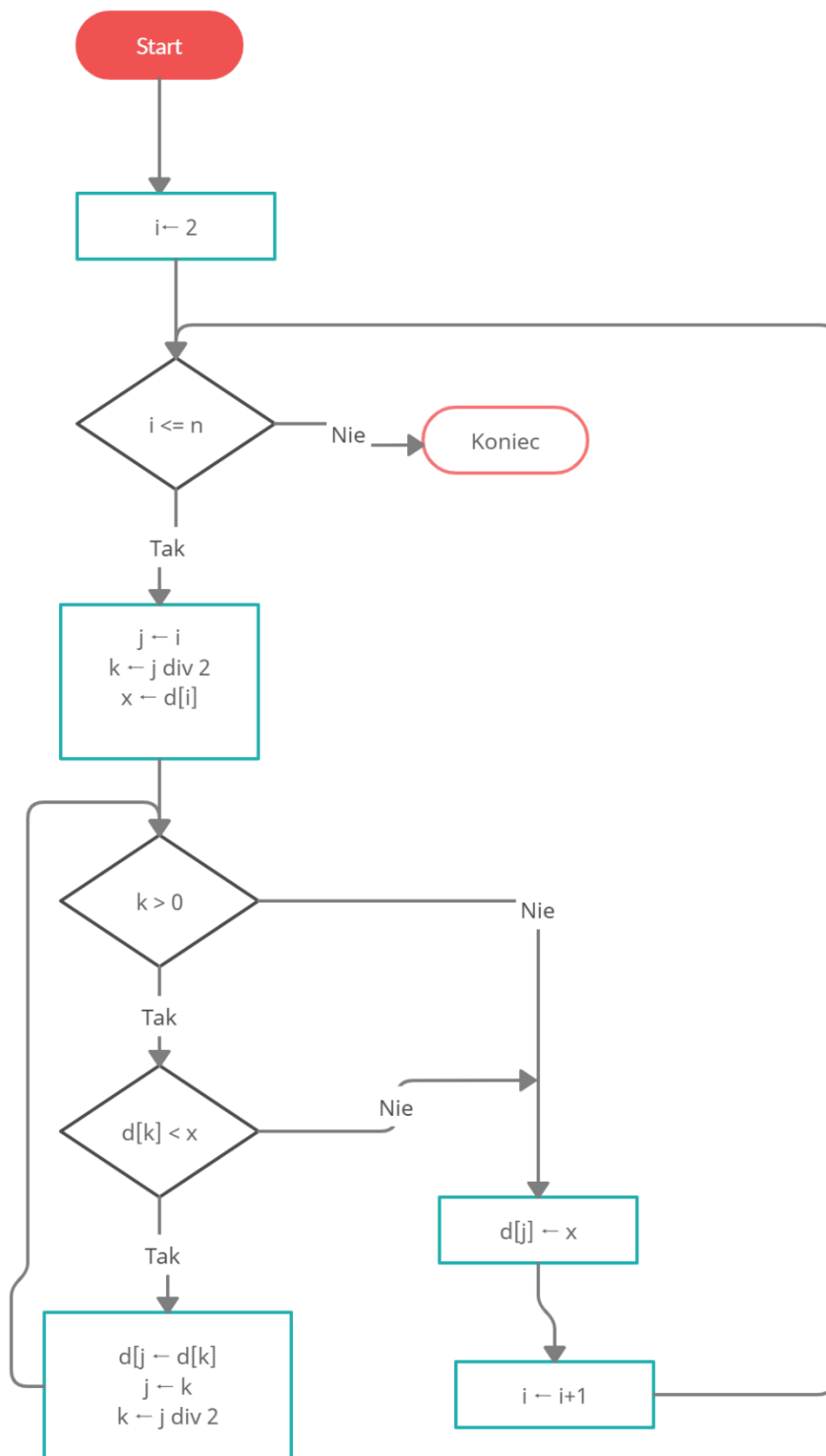
Zaimplementuj sortowanie przez scalanie oraz sortowanie kopcowe.

- 1) przedstaw schematy blokowe algorytmów oraz pseudokod odpowiadający obu schematom
- 2) przedstaw teoretyczne podstawy obu metod
- 3) wykonaj testy porównujące działanie obu metod na różnych próbkach danych i przedstaw ich wyniki w sprawozdaniu
- 4) omów złożoność obliczeniową obu algorytmów
- 5) przedstaw w postaci wykresów $t(N)$ złożoność czasową obu algorytmów dla przypadków oczekiwanego/ optymistycznego/ pesymistycznego (”odpowiednio preparując” dane do posortowania dla każdego z algorytmów) otrzymaną eksperymentalnie w wyniku serii testów dla rosnących próbek danych N

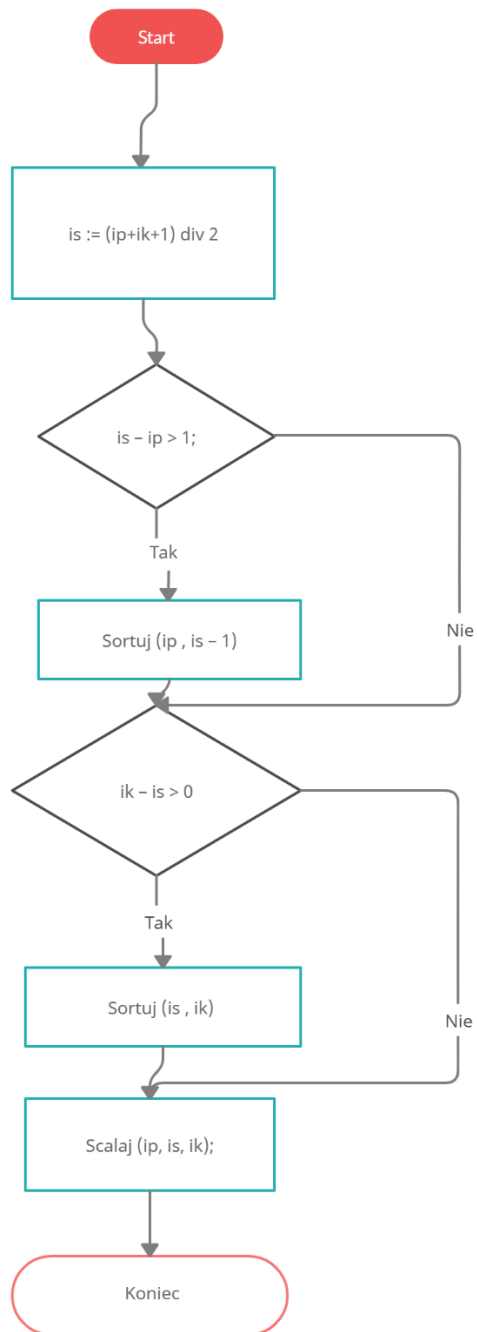
3. Schemat blokowy algorytmu.

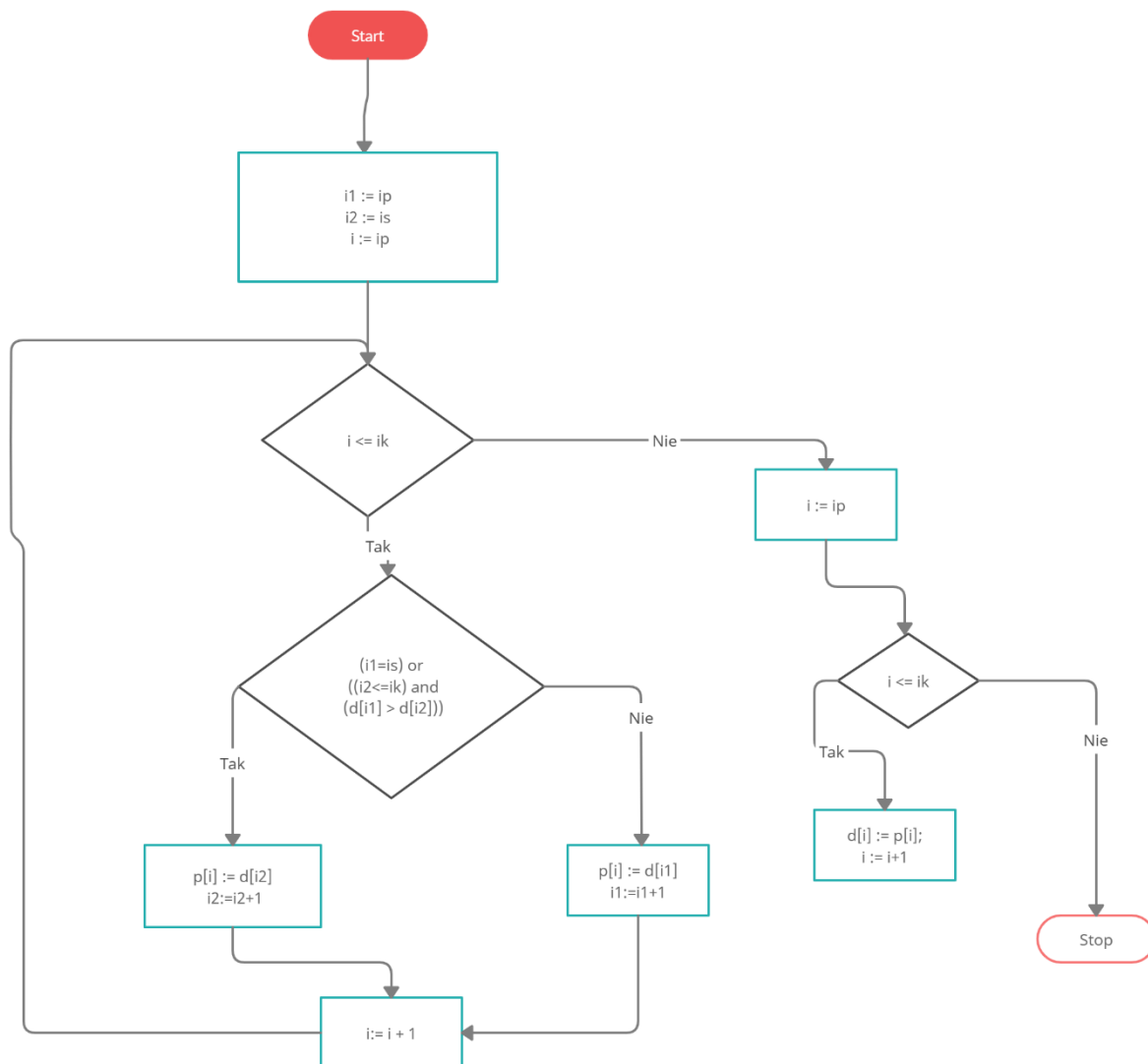
Schemat blokowy to narzędzie służące do przedstawienia kolejnych czynności w projektowanym algorytmie. Jest to diagram, na którym procedura, system lub program komputerowy są reprezentowane przez opisane figury geometryczne połączone strzałkami zgodnie z kolejnością wykonywania czynności wynikających z przyjętego algorytmu rozwiązania zadania.

Kopcowe



Scalanie





4.Pseudokod.

Pseudokodem nazywany jest taki sposób zapisu algorytmu, który zachowując strukturę charakterystyczną dla kodu zapisanego w języku programowania, rezygnuje ze ścisłych reguł składniowych na rzecz prostoty i czytelności.

Pseudokod nie zawiera szczegółów implementacyjnych często też opuszcza się w nim opis działania podprocedur (jeśli powinien być on oczywisty dla czytelnika), zaś nietrywialne kroki algorytmu opisywane są z pomocą formuł matematycznych lub zdań w języku naturalnym.

Dla sortowania przez scalanie:

Funkcja `scal(int* tab, int from, int sr, int to) {`

```

Dla (int i = from; i <= to; i++)
    tab_pom[i] = tab[i];
int i = from, j = sr + 1;
dla (int k = from; k <= to; k++) {
    jeśli (i <= sr) {
        jeśli (j <= to) {
            tab[k] = (tab_pom[j] < tab_pom[i]) ? tab_pom[j++] : tab_pom[i++];
        } w innym wypadku {
            tab[k] = tab_pom[i++];
        }
    } w innym wypadku {
        tab[k] = tab_pom[j++];
    }
}
funkcja sortowaniescalanie(int* tab, int from, int to) {
    jeśli(to <= from)
        return;
    int sr = (to + from) / 2;
    sortowaniescalanie(tab, from, sr);
    sortowaniescalanie(tab, sr + 1, to);
    scal(tab, from, sr, to);
}

```

Dla sortowania kopcowego:

Funkcja wstaw(int tab[],int n) //funkcja zapewnij¹ca tablicê

```

{

```

```
dla(int i=0; i<n; i++)
    tab[i]=rand()%1000;
}
```

```
funkcja drukuj(int tab[],int n)
{
    dla(int i =0; i<n; i++)
        wyswietl<<tab[i]<<" ";
    cout<<endl;
} //funkcja do wyœwietlenia tablicy
```

```
funkcja zanurzanie(int i, int kolejka[], int rozmiarKolejki)
{
    int l=2*i+1;
    int r=2*i+2;
    int wiekszy = i;
    jeœli(l<rozmiarKolejki&&kolejka[l]>kolejka[wiekszy])//l<N-1
        wiekszy=l;
    jeœli(r<rozmiarKolejki&&kolejka[r]>kolejka[wiekszy])//N-1
        wiekszy=r;
    jeœli(wiekszy!=i)
    {
        swap(kolejka[wiekszy],kolejka[i]);
        zanurzanie(wiekszy,kolejka,rozmiarKolejki);
    }
}
```



```
funkcja utworzKopiec(int tab[],int n)
```

```
{  
    dla(int i =n/2; i>=0; i--)  
        zanurzenie(i,tab,n);  
}
```

```
Funkcja sortuj(int tab[], int n)
```

```
{  
    utworzKopiec(tab,n);  
    dla (int i = n; i>=1; i--)  
    {  
        swap(tab[0],tab[n-1]);  
        --n;  
        zanurzenie(0,tab,n);  
    }  
}
```

5. Kod programu.

```
#include <iostream>
```

```
#include <fstream>
```

```
#include <string>
```

```
#include <time.h>
```

```
#include <cstdlib>
```

```
#include <stdio.h>
```

```
#include <iomanip>
```

```
#include <cmath>
```

```
#include <conio.h>
```

```
#include<windows.h>
```

```
#include <cstdio>
```

```
#include <ctime>
```

```
using namespace std;
```

```
const int N = 100000;
```

```
int ile,i ;
```

```
clock_t start,stop;
```

```
double czas;
```

```
int*tab_pom;
```

```
int liczby[100];
```

```
void scal(int* tab, int from, int sr, int to) {
```

```
    for (int i = from; i <= to; i++)
```

```
        tab_pom[i] = tab[i];
```

```
    int i = from, j = sr + 1;
```

```
    for (int k = from; k <= to; k++) {
```

```
        if (i <= sr) {
```

```
            if (j <= to) {
```

```
                tab[k] = (tab_pom[j] < tab_pom[i]) ? tab_pom[j++] : tab_pom[i++];
```

```
            } else {
```

```
                tab[k] = tab_pom[i++];
```

```
            }
```

```
        } else {
```

```
            tab[k] = tab_pom[j++];
```

```

    }
}
}

void sortowaniescalanie(int* tab, int from, int to) {
    if (to <= from)
        return;
    int sr = (to + from) / 2;
    sortowaniescalanie(tab, from, sr);
    sortowaniescalanie(tab, sr + 1, to);
    scal(tab, from, sr, to);
}

void wstaw(int tab[],int n) //funkcja zape³niaj¹ca tablicê
{
    for(int i=0; i<n; i++)
        tab[i]=rand()%1000;
}

void drukuj(int tab[],int n)
{
    for(int i =0; i<n; i++)
        cout<<tab[i]<<" ";
    cout<<endl;
} //funkcja do wyœwietlenia tablicy

void zanurzanie(int i, int kolejka[], int rozmiarKolejki)

```

```

{
    int l=2*i+1;
    int r=2*i+2;
    int wiekszy = i;
    if(l<rozmiarKolejki&&kolejka[l]>kolejka[wiekszy])//l<N-1
        wiekszy=l;
    if(r<rozmiarKolejki&&kolejka[r]>kolejka[wiekszy])//N-1
        wiekszy=r;
    if(wiekszy!=i)
    {
        swap(kolejka[wiekszy],kolejka[i]);
        zanurzenie(wiekszy,kolejka,rozmiarKolejki);
    }
}

```

```

void utworzKopiec(int tab[],int n)

```

```

{
    for(int i =n/2; i>=0; i--)
        zanurzenie(i,tab,n);
}

```

```

void sortuj(int tab[], int n)

```

```

{
    utworzKopiec(tab,n);
    for(int i = n; i>=1; i--)
    {
        swap(tab[0],tab[n-1]);
        --n;
    }
}

```

```

        zanurzenie(0,tab,n);
    }

}

```

```

int main()
{
    fstream plik, wyniki;

    plik.open("tablica.txt",ios::in); // Otwarcie/utworzenie pliku, z którego
    pobierana będzie tablica

    wyniki.open("wyniki.txt",ios::out); // Otwarcie/utworzenie pliku, do którego
    zapisywany będzie wynik

    cout<<"Program sortujący przez scalanie oraz kopcowo, porównujący działanie
    obu metod"<<endl;

    int x;

    cout<<"Jeśli chcesz porównać czas działania obu metod wpisz 9 "<<endl;
    cout<<"Jeśli chcesz posortować dowolną tablicę obiema metodami wpisz
    6"<<endl;

    cout<<"Jeśli chcesz odczytać tablicę z pliku wpisz 3"<<endl;

    cin >> x;

    while((x!=9)&&(x!=6)&&(x!=3))//jeżeli x nie jest równy p,d lub o postępuj
    zgodnie z pętlą
    {
        cout<<"Wprowadź literę : 9, 6 lub 3! " <<endl;

        cin>>x;
    }
}

```

```
}
```

```
if(x==9)
```

```
{
```

```
cout << "Porownanie czasow sortowania v.1" << endl;
```

```
cout<<"Ile losowych liczb w tablicy: ";
```

```
cin>>ile;
```

```
//dynamiczna alokacja tablicy
```

```
int *tablica;
```

```
tablica=new int [ile];
```

```
    tab_pom = new int[ile];
```

```
int* tab_pom = new int[ile];
```

```
int *tablica2;
```

```
tablica2=new int [ile];
```

```
//inicjowanie generatora
```

```
srand(time(NULL));
```

```
//wczytywanie losowych liczb do tablicy
```

```
for(int i=0; i<ile; i++)
```

```
{
```

```

        tablica[i] = rand()%1000+1;
    }
//przepisanie tablicy do tablicy2
    for(int i=0; i<ile; i++)
    {
        tablica2[i]=tablica[i];
    }
    for(int i=0; i<ile; i++)
    {
        tab_pom[i]=tablica[i];
    }


    cout<<"Przed posortowaniem: "<<endl;
    for(int i=0; i<ile; i++)
    {
        cout<<tablica[i]<<" ";
    }


    cout<<endl<<"Sortuje teraz przez scalanie. Prosze czekac!"<<endl;
    start = clock();
    sortowaniescalanie(tablica, 0, ile - 1);
    stop = clock();
    czas = (double)(stop-start) / CLOCKS_PER_SEC;
    cout<<endl<<"Czas sortowania przez scalanie: "<<czas<<" s"<<endl;

```

```
cout<<"Po posortowaniu: "<<endl;
```

```
    for(int i=0; i<ile; i++)
```

```
    {
```

```
        cout<<tablica[i]<<" ";
```

```
    }
```

```
cout<<endl<<"Sortuje teraz kopcowo. Prosze czekac!"<<endl;
```

```
start = clock();
```

```
drukuj(tablica, 0);
```

```
stop = clock();
```

```
czas = (double)(stop-start) / CLOCKS_PER_SEC;
```

```
cout<<endl<<"Czas sortowania kopcowego: "<<czas<<" s"<<endl;
```

```
cout<<"Po posortowaniu: "<<endl;
```

```
    for(int i=0; i<ile; i++)
```

```
    {
```

```
        cout<<tablica[i]<<" ";
```

```
    }
```

```
delete [] tablica;
```

```
delete [] tablica2;
```

```
delete[] tab_pom;
```

```
}
```



```

if(x==6)
{
    cout << "Program sortujacy przez scalanie i kopcowo" << endl<<endl;

    cout<<"Ile losowych liczb w tablicy: ";
    cin>>ile;
    int *tablica; //dynamiczna alokacja tablicy
    tablica=new int [ile];

    int *tablica2;
    tablica2=new int [ile];
    tab_pom = new int[ile];
    int* tab_pom = new int[ile];
    srand(time(NULL)); //inicjowanie generatora

    for(int i=0; i<ile; i++) //wczytywanie losowych liczb do tablicy
    {
        tablica[i] = rand()% 100000+1;
    }
    for(int i=0; i<ile; i++) //przepisanie tablicy do tablicy2
    {
        tablica2[i]=tablica[i];
    }
    for(int i=0; i<ile; i++)
    {

```

```
    tab_pom[i]=tablica[i];  
}
```

```
cout<<"Przed posortowaniem: "<<endl;  
wyniki<<"Przed posortowaniem: "<<endl;  
for(int i=0; i<ile; i++)  
{  
    cout<<tablica[i]<<" ";  
    wyniki<<tablica[i]<<" ";  
}
```

```
cout<<endl<<endl<<"Sortuje przez scalanie."<<endl;
```

```
sortowaniescalanie(tablica, 0, ile - 1);
```

```
cout<<"Po posortowaniu przez scalanie: "<<endl;  
wyniki <<"\n"<< "Po posortowaniu przez scalanie: "<<endl; //wczytywanie  
tablicy do pliku  
for(int i=0; i<ile; i++)  
{  
    cout<<tablica[i]<<" ";  
    wyniki<< tablica[i] <<" "; //wczytywanie tablicy do pliku  
}
```

```
cout<<endl<<endl<<"Sortuje kopcowe."<<endl;  
sortuj(tablica2,ile);
```

```
cout<<"Po posortowaniu kopcowym: "<<endl;
```

```

wyniki <<"\n"<< "Po posortowaniu kopcowym: "<<endl;
for(int i=0; i<ile; i++)
{
    cout<<tablica[i]<<" ";
    wyniki<< tablica[i] <<" ";
}

cout<<endl;
delete [] tablica;
delete [] tablica2;
delete [] tab_pom;
}

if(x==3)
{
    int ile=10000; //liczba elementow w pliku //funkcja do wczytywania tablic
do testow

    int tablica[ile]; //zadeklarowanie dwoch tablic
    int tablica2[ile];

    for(int i=0; i<ile; i++) //petla wpisujaca elementy z pliku do tablicy
    {
        plik>>tablica[i];
        cout<<"Przed posortowaniem: "<<endl;
        wyniki<<"Przed posortowaniem: "<<endl;
        for(int i=0; i<ile; i++)
        {
            cout<<tablica[i]<<" ";

```

```

        wyniki<<tablica[i]<<" ";
    }

    cout<<endl<<endl<<"Sortuje przez scalanie."<<endl;

    sortowaniescalanie(tablica, 0, ile - 1);

    cout<<"Po posortowaniu przez scalanie: "<<endl;
    wyniki <<"\n"<< "Po posortowaniu przez scalanie: "<<endl; //wczytywanie
    tablicy do pliku
    for(int i=0; i<ile; i++)
    {
        cout<<tablica[i]<<" ";
        wyniki<< tablica[i] <<" "; //wczytywanie tablicy do pliku
    }

    cout<<endl<<endl<<"Sortuje kopcowe."<<endl;
    sortuj(tablica2,ile);

    cout<<"Po posortowaniu kopcowym: "<<endl;
    wyniki <<"\n"<< "Po posortowaniu kopcowym: "<<endl;
    for(int i=0; i<ile; i++)

    {
        cout<<tablica[i]<<" ";
        wyniki<< tablica[i] << " ";
    }

```

```

    cout<<endl;

    plik.close();

    wyniki.close();
}

return 0;
}
}

```

6. Działanie programu.

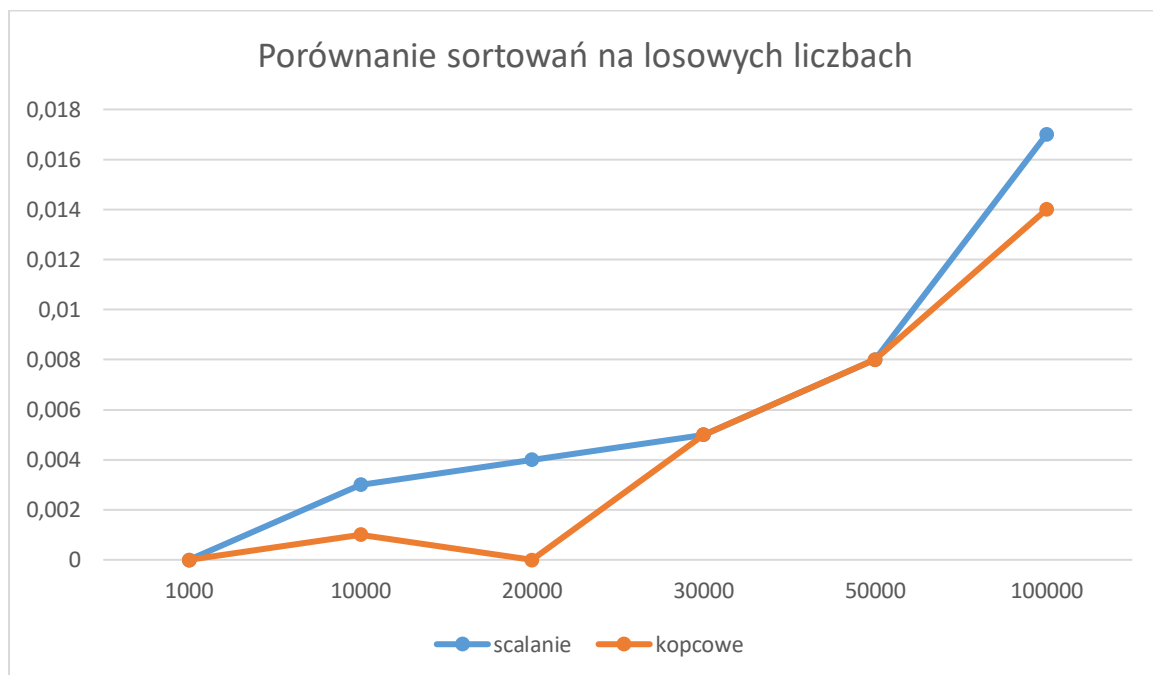
Program miał na celu pobranie liczb z pliku lub wczytanie losowych liczb i porównanie ich w konkretny sposób, a mianowicie sortowaniem przez scalanie oraz sortowaniem kopcowym.

Sortowanie przez scalanie to obecnie najszybsza znana metoda sortowania danych. Działanie algorytmu opiera się na scalaniu posortowanych już list. Posortowane listy uzyskujemy dzieląc listę elementów na pary elementów (możliwe jest wystąpienie elementu bez pary), które sortujemy w obrębie ich pod listy. Algorytm na początek musi podzielić listę na mniejsze podlisty o maksymalnie dwóch elementach. Kolejny krok polega na posortowaniu każdej z powstałych podlist. A następnie czas na kolejną część działania algorytmu – scalanie. Scalane listy są już posortowane, więc można porównać pierwsze elementy scalanych list. Dopisać na listę wynikową mniejszą liczbę i usunąć ze starej listy. Należy powtarzać krok porównywania do momentu, gdy nie będziemy mieć elementów do porównania - wtedy przepisujemy wszystkie pozostałe elementy do listy wynikowej z niepustej listy. W ten sposób uzyskujemy posortowaną listę.

Natomiast sortowanie kopcowe składa się z kopca. Kopiec jest drzewem binarnym, w którym wszystkie węzły spełniają następujący warunek (zwany warunkiem kopca): Węzeł nadrzędny jest większy lub równy węzłom potomnym (w porządku malejącym relacja jest odwrotna - mniejszy lub równy). Konstrukcja kopca jest nieco bardziej skomplikowana od konstrukcji drzewa binarnego, ponieważ musimy dodatkowo troszczyć się o zachowanie warunku kopca. Zatem po każdym dołączeniu do kopca nowego węzła, sprawdzamy odpowiednie warunki i ewentualnie dokonujemy wymian węzłów na ścieżce

wiodącej od dodanego wężła do korzenia. Charakterystyczną cechą kopca jest to, iż korzeń zawsze jest największym (w porządku malejącym) elementem z całego drzewa.

Program posiada możliwość wczytania danych z pliku oraz posortowania tych danych. Działając na sortowaniach program ma możliwość porównania obu metod oraz pokazania różnicy czasu. Wyniki testów, które wykonałam pokazane są na wykresie poniżej.



Poniżej zamieszczone są zrzuty ekranu z konsoli:

[illegible]

```

Zasada sortowania kopcowego: 0 s
Po posortowaniu:
2 4 5 6 6 7 8 9 10 11 11 14 17 17 18 18 22 22 23 23 25 25 31 33 35 35 38 38 39 39 40 40 44 44 47 47 52 52 53 53 58 58 59 59 65 65 67 67 68 68 69 70 70 71 71 73 73 74 75 75 86 86 82 82 83 83
3 5 136 136 137 138 138 140 141 142 142 143 143 144 144 146 147 148 148 149 150 150 151 151 152 152 154 155 156 159 160 162 163 164 164 165 165 167 168 171 172 172 178 178 178 178 182 183 185 186 186 187 189 189 191 191 192 193 195 195 197 197 197 197 198 199 205 205 205 206 206 207 207 208 208 209 209 211 216 217 218 218 219 220 220 220 221 221 222 224 225 226 226
229 230 230 231 232 233 238 243 244 244 248 248 249 250 250 252 254 256 258 259 262 263 265 267 267 268 268 270 270 271 272 274 275 276 276 278 284 284 285 285 286 286 286 287 289 289 290 292
300 300 301 303 306 306 307 308 309 310 311 312 314 315 316 320 320 321 322 322 323 324 324 324 327 327 327 327 330 330 330 331 334 334 336 337 338 340 340 341
42 343 343 344 344 344 345 346 347 349 353 354 355 355 357 359 361 362 364 367 367 372 372 373 376 376 378 379 381 381 382 383 384 388 390 391 391 395 396 396 397 398 401 402 403 404 405 406 406 407
408 410 410 410 411 412 414 414 416 419 419 419 423 424 424 425 426 427 430 432 433 434 434 434 436 437 437 438 438 439 441 441 444 445 446 448 448 449 451 452 452 455 455 455 455 455 455
456 456 458 459 459 459 461 461 462 468 469 470 471 474 475 476 478 478 488 481 482 482 483 483 483 485 485 487 489 489 489 492 493 494 494 494 495 496 496 497 497 497 498 502 502 503 503 505 507 507
510 511 512 512 514 515 515 516 516 516 516 516 516 516 516 516 516 516 516 516 516 516 516 516 516 516 516 516 516 516 516 516 516 516 516 516 516 516 516 516 516 516 516 516 516 516 516
557 557 558 558 561 561 561 565 565 565 567 568 571 572 572 572 572 572 572 572 572 572 572 572 572 572 572 572 572 572 572 572 572 572 572 572 572 572 572 572 572 572 572 572 572 572 572
610 610 610 611 612 612 612 615 616 616 617 623 624 625 626 629 631 632 633 634 635 636 636 637 638 639 639 640 640 641 641 641 645 645 645 645 646 646 648 651 652 658 658 659 660 661 663 663
665 666 666 666 667 669 671 671 672 674 675 675 676 677 678 679 679 682 682 682 683 683 683 685 686 686 687 687 689 691 692 693 693 693 695 696 696 697 698 698 701 702 708 708 708 708 708 708 708
709 710 711 713 715 716 718 720 720 720 721 724 726 726 727 727 727 728 728 730 732 733 734 735 736 738 738 738 738 740 740 740 742 743 743 744 745 746 750 751 754 754 756 757 760 766 767 768 77
814 816 817 818 820 821 821 823 823 827 828 829 830 830 832 833 834 836 839 840 842 842 843 845 845 845 846 848 849 851 852 852 855 855 856 856 860 861 862 863 865 866 866 867 867 868 869 869
873 875 876 876 877 878 878 881 881 882 882 886 886 886 886 887 890 891 893 894 894 897 898 898 899 901 902 903 903 904 904 904 906 906 906 907 908 909 909 910 910 914 914 915 918 920 922 927 928 930
912 934 937 938 940 944 945 946 946 947 948 948 950 950 951 952 953 953 954 954 954 955 956 960 960 961 964 964 966 966 967 967 970 971 972 972 972 973 973 974 975 975 976 977 977 977 978
978 979 979 979 982 982 982 982 982 982 982 982 982 982 982 982 982 982 982 982 982 982 982 982 982 982 982 982 982 982 982 982 982 982 982 982 982 982 982 982 982 982 982 982 982
Process returned 0 (0x0)   execution time : 5,868 s
Press any key to continue.

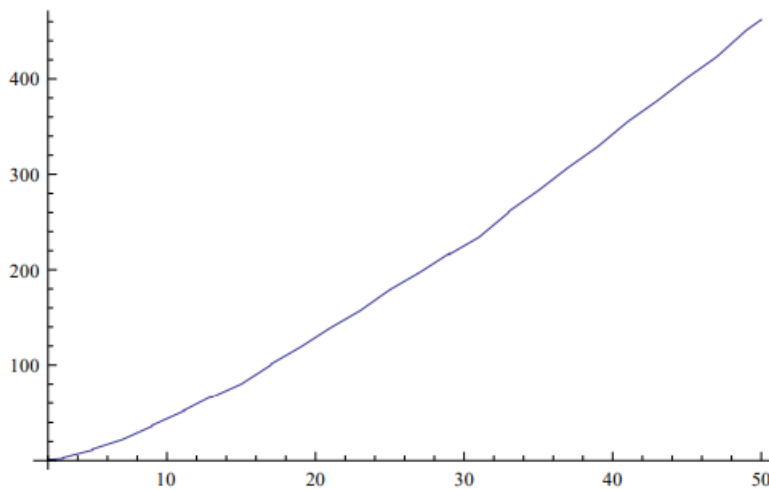
```

```

C:\Users\zawik\OneDrive\Pulpit\programy\sortowania\cudofastmain.exe
Programy sortowania: sortowanie oraz kopcowanie, porównywanie działania obu metod
Jeśli chcesz porównać czas działania obu metod wpisz 1
Jeśli chcesz posortować dowolną tablicę oblicz metodą wpisz 2
Jeśli chcesz odczytać tablicę z pliku wpisz 3
Programy sortowania: sortowanie oraz kopcowanie, porównywanie działania obu metod
Porównanie czasów sortowania v1
Ple losowych liczb w tablicy: 10000
Sortowanie posortowaniem:
1 113 427 427 1887 843 526 111 900 953 341 565 351 134 266 402 337 532 5 154 87 363 121 933 127 857 401 502 38 893 599 180 865 231 979 52 111 102 593 283 295 446 200 555 959 859 488 232 104 566 807
483 819 377 780 483 641 48 196 902 672 504 609 396 127 848 177 844 385 564 690 126 332 61 309 171 174 570 527 372 357 215 698 120 953 478 250 634 374 102 462 85 864 348 351 657 300 106 162 789 70
483 839 597 227 520 436 13 496 239 175 20 777 522 374 284 233 644 308 93 268 273 984 50 782 358 12 256 138 718 321 901 3 687 867 109 989 165 225 197 513 843 225 510 500 699 711 901 761 1017 10 32 969 466 705
5 140 682 467 68 433 114 545 483 920 396 586 327 120 827 967 743 344 829 505 789 35 318 24 367 375 381 11 342 478 416 713 792 372 866 833 120 780 829 505 708 192 77 268 993 194 819 720 291 627 864 623 39
5 433 981 31 785 265 262 17 20 493 634 670 911 64 144 27 726 776 363 997 955 435 422 961 381 233 966 780 345 409 75 420 474 784 923 644 271 455 21 652 168 276 745 770 892 296 395 927 708 444 116 304 239 572 82
5 670 933 253 924 830 598 832 48 1000 73 509 510 49 615 863 728 812 4 342 852 37 699 52 420 465 422 515 724 657 255 732 395 258 1042 226 309 164 997 194 423 426 466 942 457 925 213 148 312 982 552 959 998 121 117
746 828 761 958 718 237 127 671 327 678 847 415 911 65 17 317 318 731 265 600 862 97 450 265 765 538 121 186 262 106 198 348 624 100 798 997 77 647 67 684 783 61 163 511 45 430 143 852 532 59 684 280 7
746 794 959 59 621 121 705 897 623 280 699 636 427 15 817 75 338 438 767 551 292 69 49 516 515 383 1 3 232 269 530 446 626 835 827 113 444 378 327 879 848 992 980 34 700 957 102 34
5 31 98 24 905 929 627 672 774 978 334 611 311 588 620 335 888 751 380 984 489 486 61 659 337 942 316 1 312 507 395 21 372 278 280 337 11 965 617 227 517 311 554 327 168 898 613 273 765 59 464 808
269 503 295 577 131 630 936 502 549 99 494 951 801 368 959 333 575 403 711 968 915 437 463 563 6 515 129 211 92 344 984 124 952 833 134 493 317 40 600 412 991 109 27 756 307 148 181 437 405 56 186 457 10
72 872 68 101 922 661 83 526 72 627 680 670 750 887 380 383 652 382 648 500 396 103 856 16 862 924 217 578 671 262 106 798 155 178 972 590 610 280 226 580 180 826 251 502 55 506 758 561 445 70 189 123 161
72 693 784 964 437 983 49 680 546 805 3 815 236 638 569 339 409 177 656 184 973 727 372 462 344 970 1700 530 190 514 834 734 396 623 544 668 953 606 203 561 602 696 842 659 808 325 323 25 130 930 664 941 904
5 693 558 537 784 216 171 418 294 885 574 659 678 279 979 386 211 311 609 287 487 148 833 881 390 103 420 567 521 661 278 954 112 92 294 363 508 5 107 589 310 763 924 339 523 385 8732 321 681 080 177
617 955 864 772 910 455 562 510 296 236 343 764 241 783 71 615 596 588 213 400 122 286 836 41 555 311 108 577 17 301 748 558 191 125 742 669 732 476 816 730 846 899 760 797 732 7376 294 79 295 533 753 553 912
72 693 784 964 437 983 49 680 546 805 3 815 236 638 569 339 409 177 656 184 973 727 372 462 344 970 1700 530 190 514 834 734 396 623 544 668 953 606 203 561 602 696 842 659 808 325 323 25 130 930 664 941 904
72 693 784 964 437 983 49 680 546 805 3 815 236 638 569 339 409 177 656 184 973 727 372 462 344 970 1700 530 190 514 834 734 396 623 544 668 953 606 203 561 602 696 842 659 808 325 323 25 130 930 664 941 904
5 693 558 537 784 216 171 418 294 885 574 659 678 279 979 386 211 311 609 287 487 148 833 881 390 103 420 567 521 661 278 954 112 92 294 363 508 5 107 589 310 763 924 339 523 385 8732 321 681 080 177
617 955 864 772 910 455 562 510 296 236 343 764 241 783 71 615 596 588 213 400 122 286 836 41 555 311 108 577 17 301 748 558 191 125 742 669 732 476 816 730 846 899 760 797 732 7376 294 79 295 533 753 553 912
72 693 784 964 437 983 49 680 546 805 3 815 236 638 569 339 409 177 656 184 973 727 372 462 344 970 1700 530 190 514 834 734 396 623 544 668 953 606 203 561 602 696 842 659 808 325 323 25 130 930 664 941 904
72 693 784 964 437 983 49 680 546 805 3 815 236 638 569 339 409 177 656
```

Złożoność obliczeniowa:

Dokładna analiza przypadków w sortowaniu kopcowym



Sortowanie kopcowe:

Najlepszy przypadek- $n \log n$

Najgorszy przypadek- $n \log n$

Liczba porównań dokonanych faktycznie może zależeć od kolejności, w jakiej wartości są podane. Fakt, że najlepszym i najgorszym przypadkiem są każdy $\Theta(n \log n)$ - zakładając, że wszystkie elementy są różne - oznacza tylko, że *asymptotycznie* nie ma różnicy między nimi, choć mogą się różnić stałym czynnikiem. Nie mam żadnych prostych przykładów tego z góry mojej głowy, ale wierzę, że można tworzyć dane wejściowe, gdzie liczba porównań różni się stałym czynnikiem między tymi dwoma podejściami. Ponieważ notacja big-O ignoruje stałe, nie znajduje to odzwierciedlenia w analizie najlepszych i najgorszych przypadków.

Sortowanie przez scalanie:

Najgorszym przypadkiem scalenia będzie ten, w którym sortowanie scalania będzie musiało wykonać **maksymalną liczbę porównań**.

Przypadek pesymistyczny:

1. Załóżmy, że tablica w ostatnim kroku po sortowaniu jest $\{0,1,2,3,4,5,6,7\}$
2. W najgorszym przypadku tablica przed tym krokiem musi być, ponieważ tutaj lewa $tab_pom =$ i prawa $tab_pom =$ spowoduje maksymalne

porównania. (*Przechowywanie alternatywnych elementów w lewej i prawej podawanie*{0,2,4,6,1,3,5,7}{0,2,4,6}{1,3,5,7})

Powód: Każdy element tablicy będzie porównywany conajmniej raz.

3. Zastosowanie tej samej powyższej logiki dla lewej i prawej podarze dla poprzednich kroków: Dla tablicy najgorszym przypadkiem będzie, jeśli poprzednia tablica jest i i dla tablicy najgorszym przypadkiem będzie dla i .{0,2,4,6}{0,4}{2,6}{1,3,5,7}{1,5}{3,7}
4. Teraz zastosowanie tego samego dla poprzednich tablic kroków: W *najgorszych przypadkach*: musi być , musi być , musi być . Cóż, jeśli spojrzeć wyraźnie ten krok nie jest **konieczne**, ponieważ jeśli rozmiar tab_pom jest 2 następnie każdy element będzie porównywany conajmniej raz, nawet jeśli tablica rozmiar 2 jest sortowane.{0,4}{4,0}{2,6}{6,2}{1,5}{5,1}{3,7}{7,3}

Przypadek optymistyczny:

Co jest najlepszym przypadkiem operacji scalania, musimy tylko wziąć pod uwagę część algorytmu, gdzie "splata" dwa tab_pom razem. Podprogram ten działa w $O(n+m)O(n+m)$ Gdzie NN I MM są długościami dwóch podabli. Dzieje się tak, ponieważ poszczególne podaje są już sortowane z poprzedniej operacji scalania. Podsumowując, operacja scalania przebiega w czasie liniowym. Między każdą parą poziomów tablicy jest poziom scalania. Nawet jeśli poprzedni poziom tablicy nie jest scalanie w jednej tablicy na następnym poziomie, nadal jest scalany do innej tablicy na tym samym poziomie.

7. Wnioski.

Zadanie projektowe rozszerzyło moją wiedzę z zakresu programowania, a w szczególności z zakresu sortowań. Program działa prawidłowo. Posiada możliwość odczytu danych z pliku tekstowego, a także zapisania wyników do pliku tekstowego. W programie umieściłam także odpowiednie komentarze, które pomagają w dokładnym zrozumieniu jego działania. Zadanie projektowe pozwoliło mi na lepsze zrozumienie pojęcia funkcji w programowaniu oraz poćwiczenie podstawowych pętli jak i instrukcji warunkowych.