Sprawozdanie 1

27.03.2020r.

prowadzący: mgr inż. Marta Emirsajłow

termin: PT 9.15

Kinga Długosz: 249002

1. Wprowadzenie

Krótki opis algorytmów:

-**sortowanie przez scalanie (merge sort)** polega na zastosowaniu metody Dziel i Zwyciężaj. Polega ona na dzieleniu tablicy na dwie podtablice, rekursywnie, tak długo, aż każda z podtablic będzie miała długość 1. Następnie scala się podtablice w jedną tablicę, na bieżąco porównując je ze sobą, mniejsze wartości ustawiając po lewej stronie (w przypadku sortowania rosnącego). W przypadku tego sortowania musimy korzystać z tablicy pomocniczej do układania wartości w odpowiedniej kolejności.

-**sortowanie szybkie (quicksort)** podobnie jak sortowanie przez scalanie korzysta z metody Dziel i Zwyciężaj. Po pierwszej czynności – wybraniu elementu rozdzielającego, tzw. piwota, dokonywana jest seria porównań. Elementy mniejsze lub równe od elementu rozdzielającego są przenoszone do lewej części tablicy, natomiast większe do prawej. Następnie wywołuje się sortowanie dla lewej i prawej części z osobna (rekurencja). Ciąg wywołań zakończy się, gdy wszystkie podtablice będą miały długość 1. W przeciwieństwie do sortowania przez scalanie, nie ma już potrzeby składania wszystkich elementów w całość – są już posortowane w tablicy źródłowej

-sortowanie introspektywne (introsort)
Podczas tego sortowania są wykorzystywane algorytmy sortowania szybkiego, przez wstawianie oraz przez kopcowanie. Badana jest głębokość wywołań rekurencyjnych z poziomu głównej funkcji sortowania introspektywnego, pozwalająca zachować w najgorszym przypadku złożoność obliczeniową O(n*logn).
W przypadku małych tablic czynności organizacyjne, jak obliczenie miejsca podziału tablicy i utworzenie wskaźników na nie, zajmują więcej czasu niż samo sortowanie. Dla takich tablic są stosowane podstawowe algorytmy sortujące, czyli w moim przypadku sortowanie przez wstawianie. Natomiast gdy rozmiar podtablicy będzie większy niż 16, zostanie uruchomiona procedura Partition, opierająca się na zasadzie działania sortowania szybkiego, a w przypadku

najgorszym, gdy przekroczymy głębokość wywołań rekurencyjnych zostanie wywołane sortowanie przez kopcowanie.

2. Złożoność obliczeniowa

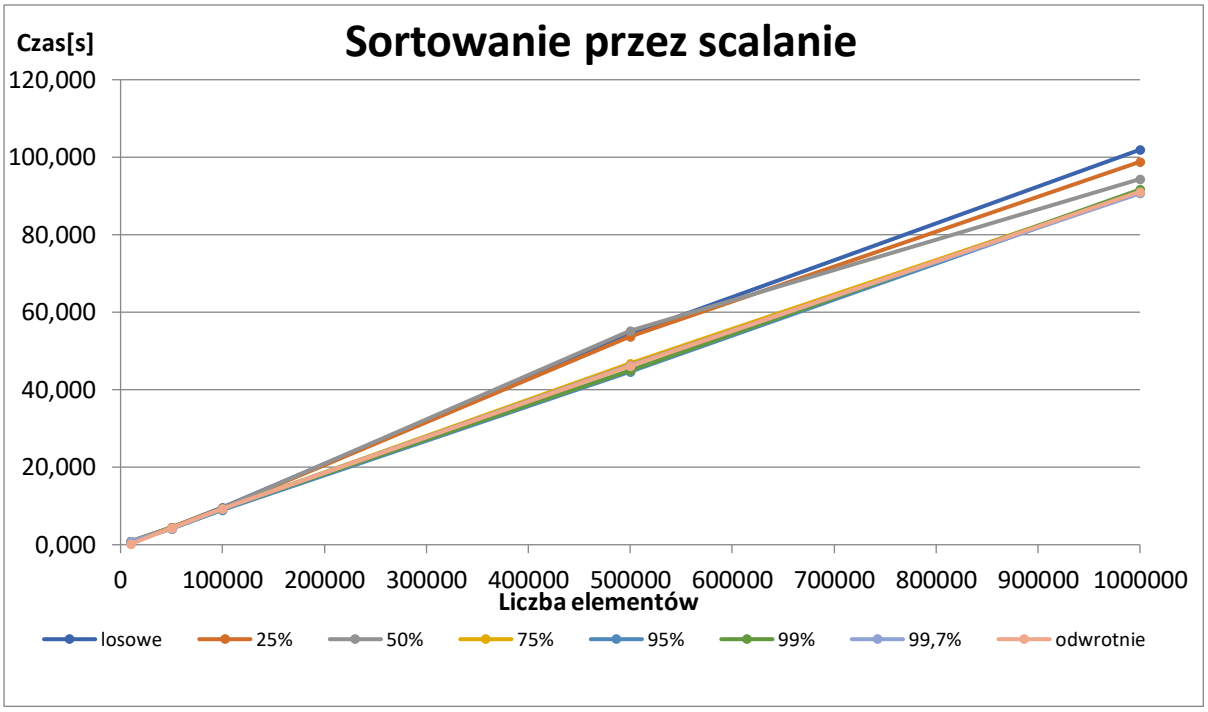
Dla każdego z badanych algorytmów uzyskano złożoność obliczeniową równą O(n*logn). Jednak w przypadku sortowania przez scalanie czas ten jest ok. 2 razy dłuższy niż w pozostałych algorytmach, jednak to nadal nie wpływa na złożoność i algorytm działa ze złożonością O(n*logn). Natomiast w sortowaniu szybkim w najgorszym przypadku możemy nawet dojść do złożoności O(n^2). Dzieje się tak wówczas, gdy wybrany piwot będzie największym lub najmniejszym elementem tablicy. W programie jako piwot wybierano zawsze element środkowy, więc w przypadku posortowania tablicy w 50% możemy zauważyć, że program pracuje ze złożonością O(n^2). Dla każdej innej wartości posortowania różnej od około 50% problem ten nie występuję. Zastosowanie Sortowania introspektywnego pozwala uniknąć tego przypadku, ponieważ program nie będzie zagłębiał się w „nieskończoność” w rekurencję, lecz po przekroczeniu odpowiedniej głębokości wywołań uruchomi sortowanie przez kopcowanie dla pozostałej części tablicy.

3. Prezentacja wyników

Dla każdej wskazanej kombinacji wykonano po pięć testów a następnie zebrano odpowiednie dane w tabeli policzono ich średnie oraz sporządzono odpowiednie wykresy.

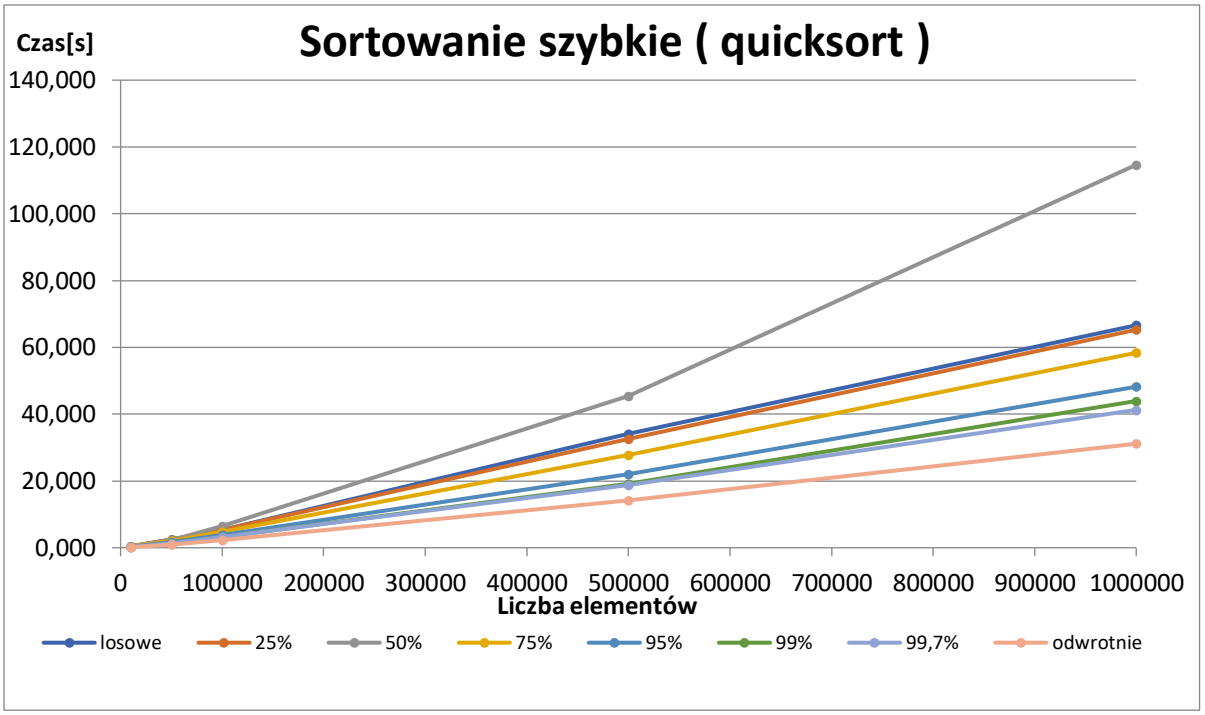
Sortowanie przez scalanie								
test numer:	wszystkie elementy losowe[s]	25%[s]	50%[s]	75%[s]	95%[s]	99%[s]	99,7%[s]	Elementy posortowane odwrotnie[s]
100 tablic po 10 000 elementów								
1	0,903	0,872	0,839	0,826	0,812	0,809	0,819	0,849
2	0,861	0,856	0,828	0,841	0,923	0,815	0,817	0,840
3	0,854	0,945	0,847	0,809	0,859	0,828	0,810	0,810
4	0,855	0,870	0,894	0,813	0,848	0,870	0,813	0,811
5	0,862	0,850	0,907	0,823	0,835	0,815	0,813	0,802
srednia	0,867	0,879	0,863	0,822	0,855	0,827	0,814	0,822
100 tablic po 50 000 elementów								

1	4,559	4,494	4,317	4,233	4,168	4,102	4,096	4,180
2	4,530	4,505	4,328	4,241	4,137	4,091	4,176	4,346
3	4,492	4,425	4,298	4,252	4,141	4,118	4,139	4,227
4	4,507	4,467	4,302	4,241	4,169	4,094	4,135	4,679
5	4,789	4,724	4,286	4,280	4,140	4,091	4,134	4,609
srednia	4,575	4,523	4,306	4,249	4,151	4,099	4,136	4,408
100 tablic po 100 000 elementów								
1	9,585	9,362	9,186	9,321	9,041	9,219	9,151	9,233
2	9,570	9,356	9,167	9,313	9,017	9,109	9,138	9,206
3	9,590	9,418	9,181	9,350	9,041	9,124	9,214	9,226
4	9,587	9,859	9,608	9,276	9,040	9,170	9,826	9,227
5	10,011	9,616	10,140	9,340	9,001	9,573	9,284	9,262
srednia	9,669	9,522	9,456	9,320	9,028	9,239	9,323	9,231
100 tablic po 500 000 elementów								
1	49,115	51,840	54,999	46,182	44,604	45,057	46,194	46,604
2	58,832	55,816	55,608	47,145	44,480	44,691	45,388	46,223
3	52,156	52,684	55,481	45,321	45,317	44,079	46,072	45,415
4	59,503	53,134	55,461	48,611	44,987	44,971	45,716	45,842
5	52,103	54,831	54,239	46,388	43,842	45,874	47,941	46,961
srednia	54,342	53,661	55,158	46,730	44,646	44,934	46,262	46,209
100 tablic po 1 000 000 elementów								
1	102,660	99,393	94,616	92,803	91,396	92,064	90,911	91,868
2	100,464	98,413	93,416	90,416	90,160	91,041	90,111	90,616
3	102,452	99,674	94,461	91,496	91,710	91,984	90,846	89,951
4	101,150	97,164	94,365	90,741	92,410	92,140	89,471	90,161
5	102,942	99,471	95,001	90,641	90,041	91,041	91,975	92,961
srednia	101,934	98,823	94,372	91,219	91,143	91,654	90,663	91,111



Wykres 1 Sortowanie przez scalanie

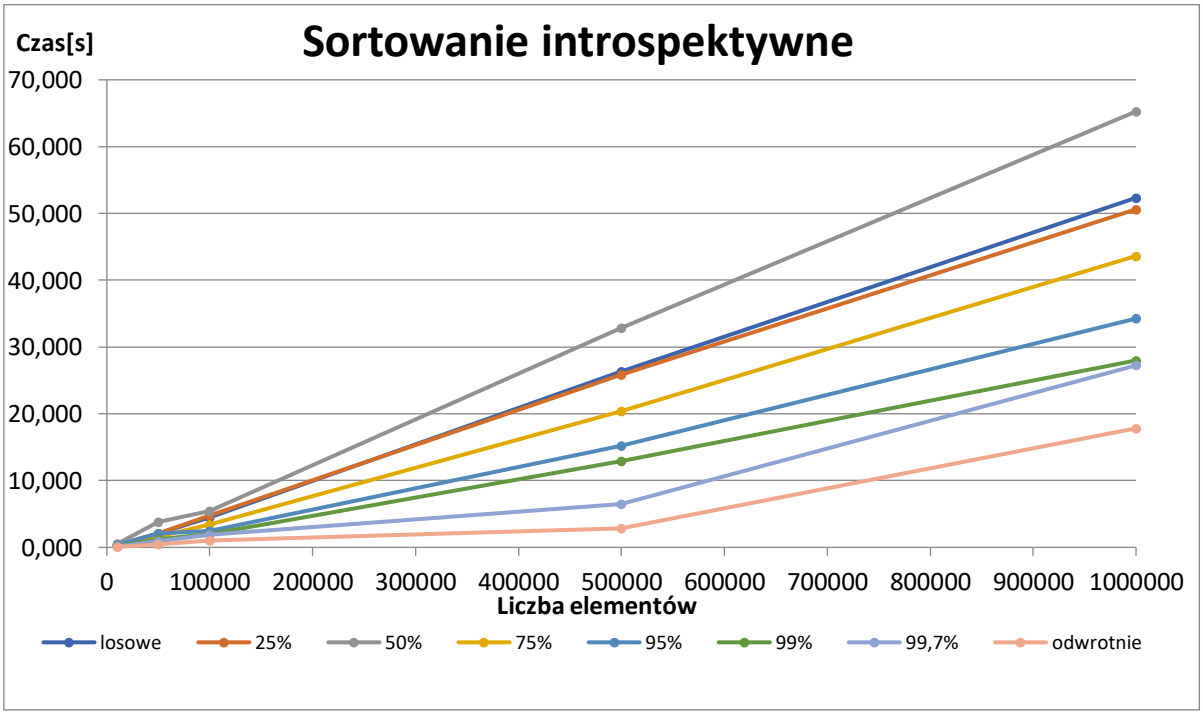
Sortowanie szybkie (quicksort)								
test numer:	wszystkie elementy losowe[s]	25%[s]	50%[s]	75%[s]	95%[s]	99%[s]	99,7%[s]	Elementy posortowane odwrotnie[s]
100 tablic po 10 000 elementów								
1	0,430	0,413	0,416	0,348	0,278	0,237	0,225	0,166
2	0,420	0,417	0,421	0,354	0,281	0,237	0,224	0,158
3	0,417	0,423	0,420	0,357	0,287	0,233	0,220	0,142
4	0,413	0,410	0,415	0,355	0,285	0,297	0,227	0,142
5	0,412	0,423	0,410	0,354	0,286	0,233	0,227	0,138
srednia	0,418	0,417	0,416	0,354	0,283	0,247	0,225	0,149
100 tablic po 50 000 elementów								
1	2,449	2,410	2,328	2,088	1,630	1,411	1,308	0,875
2	2,466	2,393	2,334	2,107	1,660	1,383	1,320	0,873
3	2,461	2,370	2,322	2,082	1,617	1,403	1,311	0,879
4	2,452	2,389	2,343	2,083	1,631	1,393	1,309	1,077
5	2,657	2,402	2,333	2,096	1,648	1,383	1,311	1,037
srednia	2,497	2,393	2,332	2,091	1,637	1,395	1,312	0,948
100 tablic po 100 000 elementów								
1	5,400	5,309	5,848	4,820	3,819	3,317	3,138	2,240
2	5,365	5,320	5,390	4,839	3,807	3,298	3,134	2,245
3	5,367	5,355	7,550	4,825	3,806	3,325	3,121	2,255
4	5,372	5,367	6,761	4,778	3,824	3,303	3,159	2,252
5	5,326	5,485	6,732	4,802	3,829	3,362	3,528	2,258
srednia	5,366	5,367	6,456	4,813	3,817	3,321	3,216	2,250
100 tablic po 500 000 elementów								
1	35,820	32,745	39,706	27,625	22,150	19,415	18,348	14,448
2	37,174	32,988	48,753	28,152	22,077	19,468	18,739	14,064
3	31,500	33,384	46,183	27,384	21,987	18,954	19,415	14,947
4	33,165	31,431	46,816	27,417	22,437	18,452	18,010	14,251
5	32,412	32,412	45,743	28,481	21,412	19,684	19,415	13,125
srednia	34,014	32,592	45,440	27,812	22,013	19,195	18,785	14,167
100 tablic po 1 000 000 elementów								
1	66,426	65,254	120,842	58,407	48,664	43,412	41,161	31,151
2	66,974	64,412	110,811	58,416	46,140	44,148	40,941	29,951
3	67,321	66,485	118,135	60,014	48,552	44,949	40,317	32,916
4	66,416	65,041	110,741	58,414	49,140	43,941	42,134	31,061
5	65,921	65,140	112,741	56,514	48,475	43,147	41,842	30,616
srednia	66,612	65,266	114,654	58,353	48,194	43,919	41,279	31,139



Wykres 2 Sortowanie szybkie

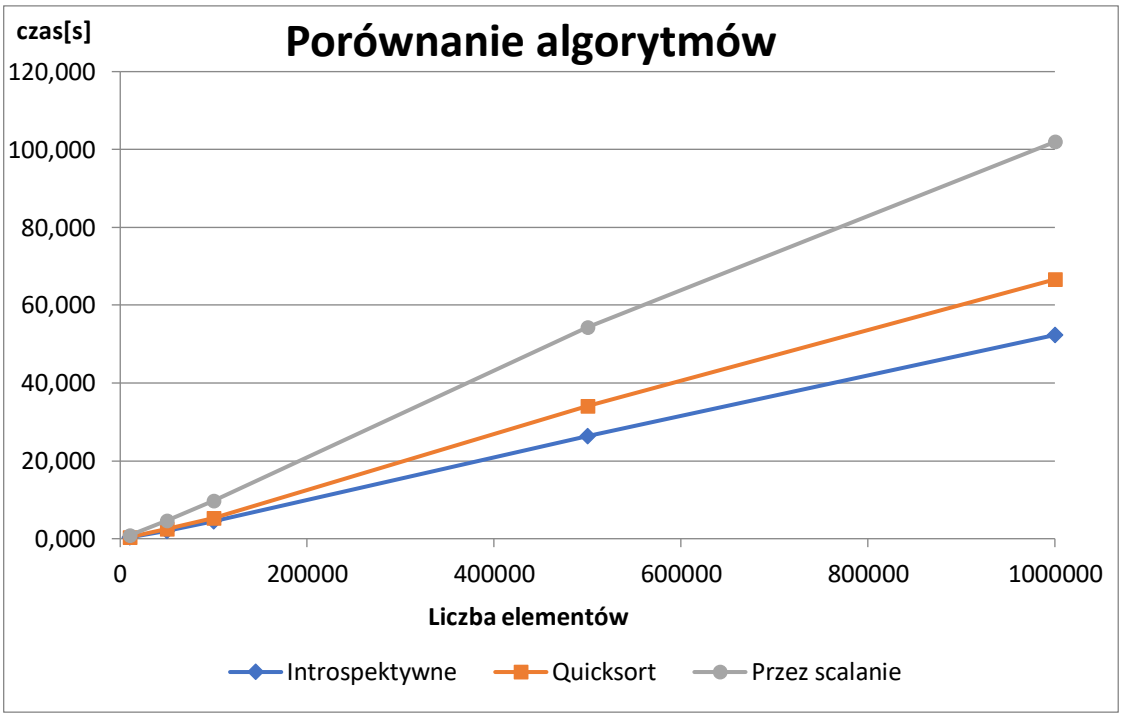
Sortowanie introspektywne								
test numer:	wszystkie elementy losowe[s]	25%[s]	50%[s]	75%[s]	95%[s]	99%[s]	99,7%[s]	Elementy posortowane odwrotnie[s]
100 tablic po 10 000 elementów								
1	0,365	0,307	0,595	0,248	0,331	0,183	0,125	0,082
2	0,355	0,396	0,515	0,246	0,386	0,176	0,126	0,076
3	0,359	0,395	0,510	0,251	0,342	0,206	0,141	0,074
4	0,330	0,400	0,520	0,258	0,386	0,172	0,142	0,075
5	0,328	0,397	0,511	0,254	0,348	0,165	0,145	0,076
srednia	0,347	0,379	0,530	0,251	0,359	0,180	0,136	0,077
100 tablic po 50 000 elementów								
1	1,985	2,091	2,396	1,733	2,092	1,200	0,868	0,412
2	2,005	2,081	2,677	1,550	1,983	1,157	0,907	0,485
3	2,115	2,000	3,064	1,527	1,968	1,221	0,860	0,448
4	2,102	1,930	5,127	1,641	2,130	1,132	0,887	0,467
5	2,077	2,130	5,787	1,612	2,080	1,197	0,902	0,452
srednia	2,057	2,046	3,810	1,613	2,051	1,181	0,885	0,453

100 tablic po 100 000 elementów								
1	4,605	4,953	5,292	3,421	2,458	2,013	1,859	0,992
2	4,386	4,724	7,405	3,387	2,484	2,018	1,815	0,983
3	4,451	4,590	4,385	3,405	2,514	2,007	1,847	0,979
4	4,412	5,039	4,864	3,452	2,473	2,021	2,104	1,034
5	4,522	4,704	5,362	3,387	2,462	2,042	1,894	1,008
srednia	4,475	4,802	5,462	3,410	2,478	2,020	1,904	0,999
100 tablic po 500 000 elementów								
1	27,385	25,239	30,146	20,270	15,065	12,069	6,973	2,797
2	27,633	26,895	32,132	19,921	15,084	12,552	6,929	2,797
3	25,063	25,161	34,612	20,971	14,964	13,984	6,145	2,922
4	25,444	25,671	33,512	21,124	15,071	13,137	5,974	2,828
5	26,157	26,134	33,652	19,841	15,714	12,974	6,415	2,880
srednia	26,336	25,820	32,811	20,425	15,180	12,943	6,487	2,845
100 tablic po 1 000 000 elementów								
1	52,806	50,575	66,014	43,347	34,064	27,852	26,625	18,033
2	53,024	51,180	65,314	43,138	33,415	27,455	26,741	16,911
3	51,714	51,031	65,145	43,764	33,149	28,142	28,151	17,981
4	51,381	50,145	64,641	43,891	34,748	28,511	26,796	18,049
5	52,741	50,142	65,162	44,000	35,915	27,981	27,916	18,141
srednia	52,333	50,615	65,255	43,628	34,258	27,988	27,246	17,823



Wykres 3 Sortowanie introspektywne

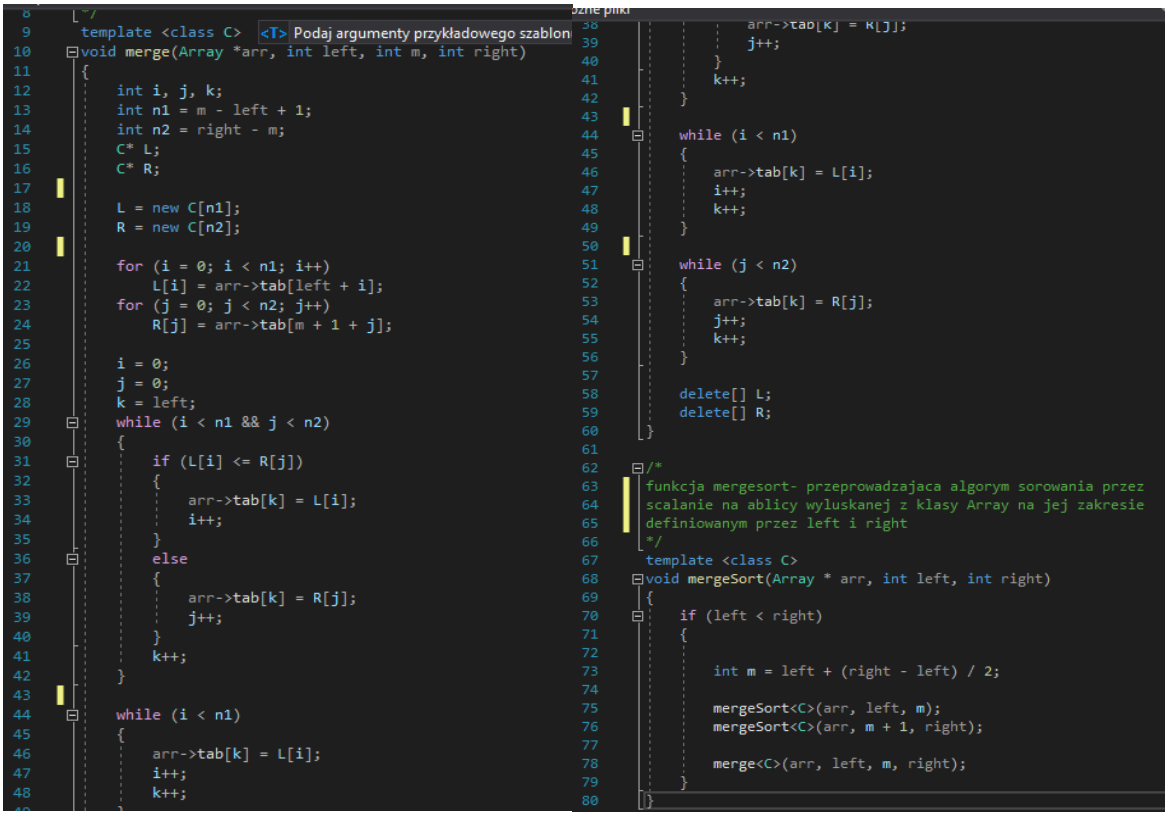
Tabela zbiorcza usrednionych wynikow								
Liczba elementów w tablicach	wszystkie elementy losowe[s]	25%[s]	50%[s]	75%[s]	95%[s]	99%[s]	99,7%[s]	Elementy posortowane odwrotnie[s]
Sortowanie przez scalanie								
10000	0,867	0,879	0,863	0,822	0,855	0,827	0,814	0,193
50000	4,575	4,523	4,306	4,249	4,151	4,099	4,136	4,408
100000	9,669	9,522	9,456	9,320	9,028	9,239	9,323	9,231
500000	54,342	53,661	55,158	46,730	44,646	44,934	46,262	46,209
1000000	101,934	98,823	94,372	91,219	91,143	91,654	90,663	91,111
Sortowanie szybkie (quicksort)								
10000	0,418	0,417	0,416	0,354	0,283	0,247	0,225	0,149
50000	2,497	2,393	2,332	2,091	1,637	1,395	1,312	0,948
100000	5,366	5,367	6,456	4,813	3,817	3,321	3,216	2,250
500000	34,014	32,592	45,440	27,812	22,013	19,195	18,785	14,167
1000000	66,612	65,266	114,654	58,353	48,194	43,919	41,279	31,139
Sortowanie introspektywne								
10000	0,347	0,379	0,530	0,251	0,359	0,180	0,136	0,077
50000	2,057	2,046	3,810	1,613	2,051	1,181	0,885	0,453
100000	4,475	4,802	5,462	3,410	2,478	2,020	1,904	0,999
500000	26,336	25,820	32,811	20,425	15,180	12,943	6,487	2,845
1000000	52,333	50,615	65,255	43,628	34,258	27,988	27,246	17,823



Wykres 4 Porównanie algorytmów

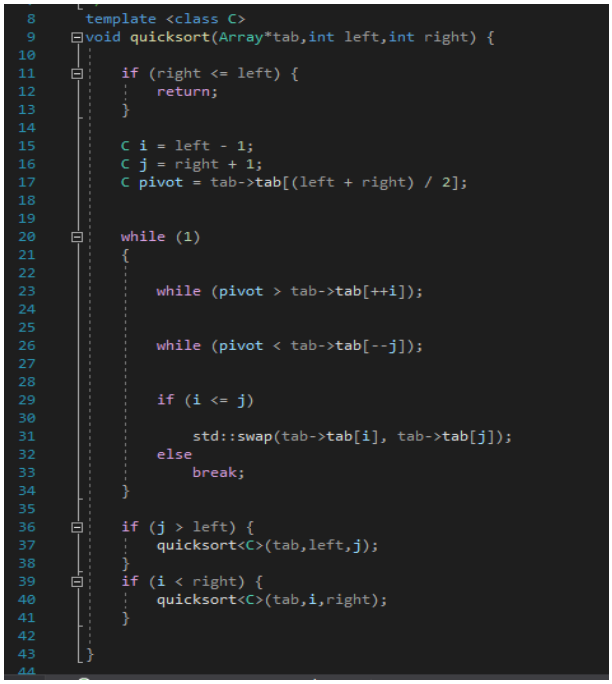
4. Funkcje

sortowanie przez scalanie (merge sort):

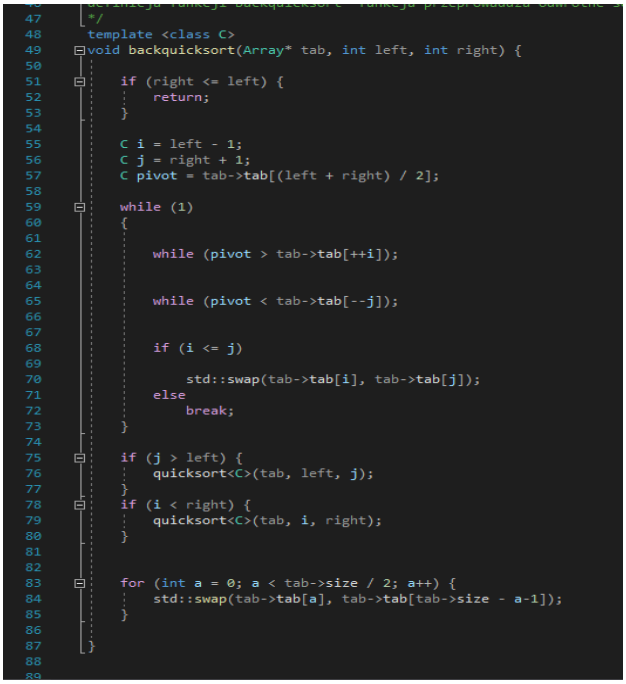


```
template <class C>
void merge(Array *arr, int left, int m, int right)
{
    int i, j, k;
    int n1 = m - left + 1;
    int n2 = right - m;
    C* L;
    C* R;

    L = new C[n1];
    R = new C[n2];

    for (i = 0; i < n1; i++)
        L[i] = arr->tab[left + i];
    for (j = 0; j < n2; j++)
        R[j] = arr->tab[m + 1 + j];

    i = 0;
    j = 0;
    k = left;
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            arr->tab[k] = L[i];
            i++;
        }
        else
        {
            arr->tab[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1)
    {
        arr->tab[k] = L[i];
        i++;
        k++;
    }

    while (j < n2)
    {
        arr->tab[k] = R[j];
        j++;
        k++;
    }

    delete[] L;
    delete[] R;
}

/*
funkcja mergesort- przeprowadzajaca algorym sorowania przez
scalanie na ablicy wyluskanej z klasy Array na jej zakresie
definiowanym przez left i right
*/
template <class C>
void mergeSort(Array * arr, int left, int right)
{
    if (left < right)
    {

        int m = left + (right - left) / 2;

        mergeSort<C>(arr, left, m);
        mergeSort<C>(arr, m + 1, right);

        merge<C>(arr, left, m, right);
    }
}
```

sortowanie szybkie (quicksort):



```
template <class C>
void quicksort(Array*tab,int left,int right) {

    if (right <= left) {
        return;
    }

    C i = left - 1;
    C j = right + 1;
    C pivot = tab->tab[(left + right) / 2];


    while (1)
    {

        while (pivot > tab->tab[++i]);


        while (pivot < tab->tab[--j]);


        if (i <= j)

            std::swap(tab->tab[i], tab->tab[j]);
        else
            break;
    }

    if (j > left) {
        quicksort<C>(tab,left,j);
    }
    if (i < right) {
        quicksort<C>(tab,i,right);
    }

}
```



```
*/
template <class C>
void backquicksort(Array* tab, int left, int right) {

    if (right <= left) {
        return;
    }

    C i = left - 1;
    C j = right + 1;
    C pivot = tab->tab[(left + right) / 2];

    while (1)
    {

        while (pivot > tab->tab[++i]);


        while (pivot < tab->tab[--j]);


        if (i <= j)

            std::swap(tab->tab[i], tab->tab[j]);
        else
            break;
    }

    if (j > left) {
        quicksort<C>(tab, left, j);
    }
    if (i < right) {
        quicksort<C>(tab, i, right);
    }


    for (int a = 0; a < tab->size / 2; a++) {
        std::swap(tab->tab[a], tab->tab[tab->size - a-1]);
    }

}
```

sortowanie introspektywne (introsort):

```
/*
funkcja introsortUtil-funckja wykorzystywana do realizacji algorytmu sortowania
introspektywnego. Podejmuje decyzje i wykonuje rodzaj sortowania najbardziej
wydajny na danym zakresie tablicy
*/
 template<class C>
void introsortUtil(Array* tab, int left, int right, int deepthlimit) {
    int deepthlimit1 = deepthlimit;
    if (right - left > 16) {
        if (deepthlimit <= 0) {
            heapsort<C>(tab, left, right);
        }
        int partitionPoint = partition<C>(tab, left, right);
        introsortUtil<C>(tab, partitionPoint + 1, right, deepthlimit1 - 1);
        introsortUtil<C>(tab, left, partitionPoint, deepthlimit1 - 1);
    }
    else {
        insertionsort<C>(tab, left, right);
    }
};

/*
funckja introsort- realizuja algorytm sortowania introspektywnego przy uzyciu
funkcji introsortUtil na zadanej tablicy wyluskane z klasy Array na jej zakresie
definiowanym przez left i right
*/
 template<class C>
void introsort(Array* tab, int left, int right) {
    int deepthlimit = 2 * log(right - left);
    introsortUtil<C>(tab, left, right, deepthlimit);
};
```

5. Wnioski

Po przeprowadzeniu testów można zauważyć, że każdy z badanych algorytmów uzyskał złożoność obliczeniową zgodną z oczekiwaną. W prawie każdym przypadku jest spełniona zasada, że im więcej elementów początkowych jest posortowanych tym czas sortowania jest szybszy. Jednak jest jeden wyjątek, w sortowaniu szybkim. W przypadku posortowania tablicy w 50% możemy zauważyć, że program pracuje ze złożonością O(n^2). Znacznie lepiej z tym przypadkiem radzi sobie sortowanie introspektywne, które zapobiega takiej złożoności używając w tym przypadku sortowania przez kopcowanie. Na wykresie można zauważyć, że algorytm sortowania introspektywnego nigdy nie osiąga złożoności obliczeniowej O(n^2). Algorytm sortowania przez scalanie, ma cały czas taką samą złożoność, wynika to z jego budowy, lecz praktycznie w każdym przypadku jego czas pracy jest ok. dwa razy dłuższy jak w pozostałych metodach sortowania. Wykonanie testów pięciokrotnie pozwoliło na wykluczenie ewentualnych wyników losowych spowodowanych przez inne czynniki zewnętrzne.

6. Literatura

- https://pl.wikipedia.org/wiki/Sortowanie_przez_scalanie

- https://pl.wikipedia.org/wiki/Sortowanie_szybkie
- https://pl.wikipedia.org/wiki/Sortowanie_introspektywne
- http://www.algorytm.edu.pl/algorytmy-maturalne.html
- https://www.programmingalgorithms.com
- http://eduinf.waw.pl/inf/alg/003_sort/index.php
- http://www.algorytm.org/algorytmy-sortowania.html