Projekt1 – algorytmy i struktury danych.

Zadanie: *Zweryfikować przedstawioną na wykładzie ocenę średniej i pesymistycznej złożoności wyszukiwania liniowego i binarnego. Przeprowadzić analizę za pomocą instrumentacji i pomiarów czasu. W porównaniu wykorzystać tablice liczb całkowitych o rozmiarze rzędu 2^30 bajtów (2^28 elementów typu uint/int).*

Słowem wstępu.

W powyższym zadaniu, w związku z problemami sprzętowymi, zostały wykorzystane tablice dążące do rozmiaru 2^28 elementów, jednak ich maksymalny przedział mieści się w granicach 2^24-2^25.

Informacje sprzętowe dla komputera na których dokonywano pomiarów:

Procesor:	Intel(R) Core(TM) i5 CPU M 520 @ 2.40GHz 2.40 GHz
Zainstalowana pamięć	4,00 GB (dostępne: 3,87 GB)

Do pisania kodu, wykorzystano program Visual Studio 2017.

Implementacja metody wyszukiwania liniowego.

Metoda wyszukiwania liniowego bez instrumentacji:

```
public static int simpleSearch(int[] tab, int a)
{
    for (int i = 0; i < tab.Length; i++)
    {
        if (tab[i] == a) return i;
    }
    return -1;
}
```

Metoda ta jako dominantę wykorzystuje liczbę porównań elementów w tablicy, do wyszukiwanej przez nas liczby.

Metoda wykorzystująca instrumentację:

```
private static int counter;
public static int simpleSearch(int[] tab, int a)...
public static int SimpleSearchOperations(int [] tab, int a)
{
    counter = 0;
    for (int i = 0; i < tab.Length; i++)
    {
        counter++;
        if (tab[i] == a) return i;
    }
    return -1;
}
```

Kod wykorzystany w eksperymencie:

```
static void Main(string[] args)
{
    Random rand = new Random();
    int lookUpValue = 1001;
    int result;
    Console.WriteLine("rozmiar;szukana;wynik;czas;ile operacji");
    for (int k = 5368709; k < (int) Math.Pow(2,28); k+=5368709)
    {
        int[] tab = new int[k];
        for (int i = 0; i < tab.Length; i++)
        {
            tab[i] = rand.Next(1, 1000);
        }
        Array.Sort(tab);
        long start = Stopwatch.GetTimestamp();
        result = simpleSearch(tab, lookUpValue);//kod przed instrumentalizacją
        long stop = Stopwatch.GetTimestamp();
        result = SimpleSearchOperations(tab, lookUpValue);//kod po instrumentalizacji
        Console.WriteLine(k+";"+lookUpValue+";"+result+";"+(stop-start)+";"+counter);
    }
}
```

W przypadku złożoności pesymistycznej wyszukiwania liniowego, mamy do czynienia z sytuacją, w którym element poszukiwany znajduje się na końcu tablicy lub nie ma go w niej wcale.
W tym zdarzeniu, przyjmujemy, że wartość wyszukiwana znajduje się poza zakresem tablicy.
W eksperymencie wykorzystano rozmiary tablic w przedziale od 5368709 do 220117069, przy czym każda kolejna tablica jest iteracją pierwszej liczby przedziału.

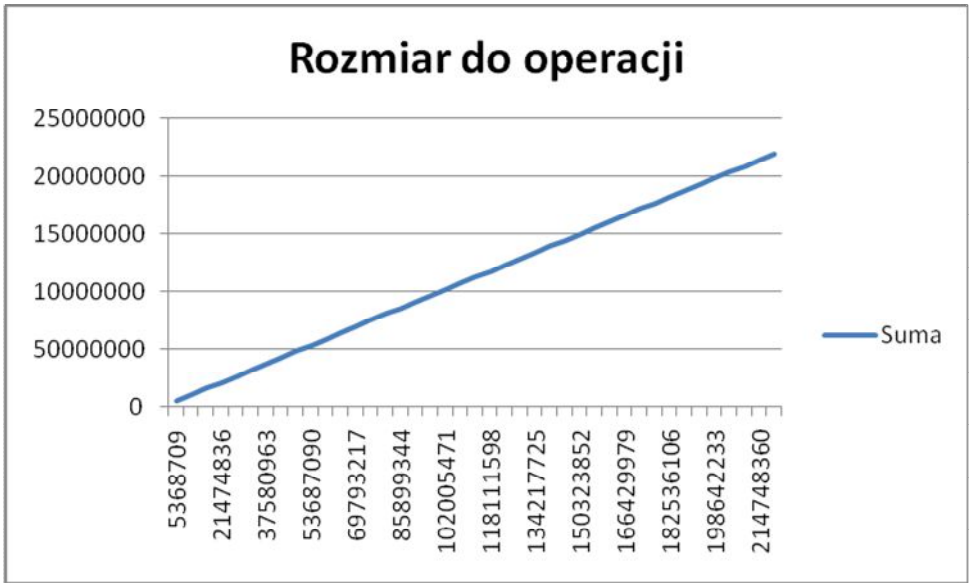
Uwaga – tablice są posortowane.

Zebrane dane:

Iteracja	rozmiar	ile operacji
0	5368709	5368709
1	10737418	10737418
2	16106127	16106127
3	21474836	21474836
4	26843545	26843545
5	32212254	32212254
6	37580963	37580963
7	42949672	42949672
8	48318381	48318381
9	53687090	53687090
10	59055799	59055799
11	64424508	64424508
12	69793217	69793217
13	75161926	75161926
14	80530635	80530635
15	85899344	85899344
16	91268053	91268053
17	96636762	96636762
18	102005471	102005471
19	107374180	107374180
20	112742889	112742889

21	118111598	118111598
22	123480307	123480307
23	128849016	128849016
24	134217725	134217725
25	139586434	139586434
26	144955143	144955143
27	150323852	150323852
28	155692561	155692561
29	161061270	161061270
30	166429979	166429979
31	171798688	171798688
32	177167397	177167397
33	182536106	182536106
34	187904815	187904815
35	193273524	193273524
36	198642233	198642233
37	204010942	204010942
38	209379651	209379651
39	214748360	214748360
40	220117069	220117069

Wykres dla instrumentacji wygląda następująco:



Ocena przy wykorzystaniu pomiaru czasu.

Zebrane dane:

Iteracja	czas
0	64677
1	132649
2	244173
3	273022
4	334547
5	413747
6	469302

7	521114
8	590689
9	650892
10	719238
11	1079240
12	1140629
13	914037
14	987400
15	1046192
16	1120108
17	1185610
18	1238817
19	1317094
20	1377321
21	1460945
22	1506589
23	2576278
24	2740311
25	1724242
26	1759132
27	1842792
28	1940851
29	1961720
30	2088897
31	4781767
32	2165058
33	2203772
34	2395150
35	2391621
36	4162543
37	2574802
38	2847287
39	3943160
40	5961103

Wykres dla pomiaru czasu:

W tym przypadku również widać, że złożoność również jest liniowa.

Skąd wzięły się piki?

Są to momenty w których procesor potrzebował więcej czasu na wyszukiwanie. Spowodowane jest to sytuacją, w której komputer wykorzystywał swoje zasoby do innych działań.

Jak widać na powyższych wykresach, w wyszukiwaniu liniowym stosunek operacji/czasu do wielkości tablicy jest wprost proporcjonalny. Im większa tablica, tym więcej operacji i czasu jaki należy poświęcić aby znaleźć (lub nie) wyszukiwaną przez nas wartość.

Wyszukiwanie liniowe – przypadek średni.

Wykorzystany kod:

```
static void Main(string[] args)
{
    Random rand = new Random();
    int lookUpValue = 1001;
    int result;
    Console.WriteLine("rozmiar;szukana;wynik;czas;ile operacji");
    for (int k = 5368709; k < (int)Math.Pow(2, 28); k += 5368709)
    {

        int[] tab = new int[k];
        for (int i = 0; i < tab.Length; i++)
        {
            tab[i] = rand.Next(1, 2600000);
        }
        Array.Sort(tab);
        long start = Stopwatch.GetTimestamp();
        result = simpleSearch(tab, lookUpValue);
        long stop = Stopwatch.GetTimestamp();
        result = SimpleSearchOperations(tab, lookUpValue);
        Console.WriteLine(k + ";" + lookUpValue + ";" + result + ";" + (stop - start) + ";" + counter);
    }
}
```

Wielkości tablic zostały takie same jak w przypadku pesymistycznym. Zmieniły się jednak nakład operacji:

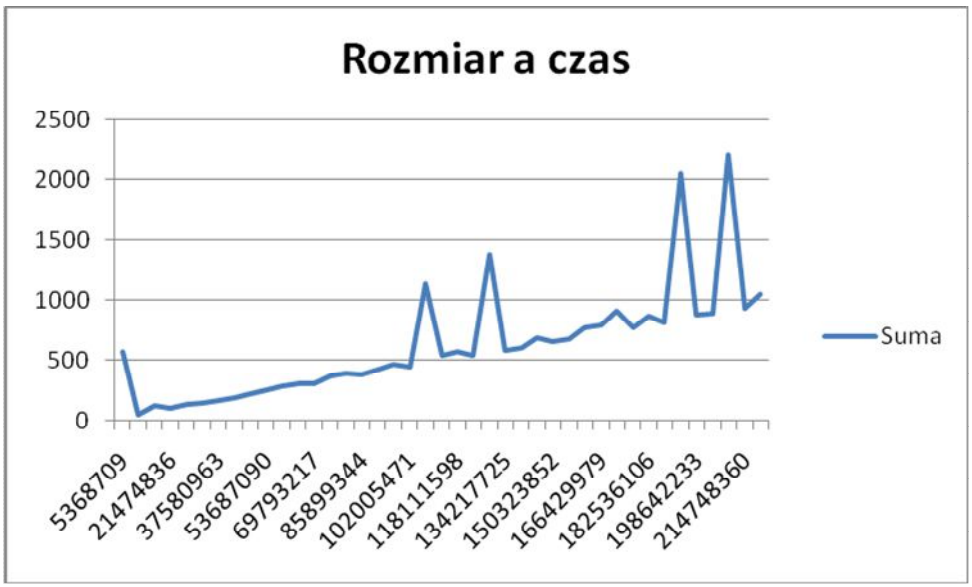
rozmiar	szukana	wynik	ile operacji
5368709	1001	2070	2071
10737418	1001	4069	4070
16106127	1001	6224	6225
21474836	1001	8330	8331
26843545	1001	10407	10408
32212254	1001	12513	12514
37580963	1001	14459	14460
42949672	1001	16455	16456
48318381	1001	18695	18696
53687090	1001	20725	20726
59055799	1001	22653	22654
64424508	1001	24717	24718
69793217	1001	26887	26888
75161926	1001	29085	29086
80530635	1001	30824	30825
85899344	1001	33164	33165
91268053	1001	35159	35160

96636762	1001	37572	37573
1,02E+08	1001	38944	38945
1,07E+08	1001	41374	41375
1,13E+08	1001	43201	43202
1,18E+08	1001	45693	45694
1,23E+08	1001	47760	47761
1,29E+08	1001	50164	50165
1,34E+08	1001	51538	51539
1,4E+08	1001	53661	53662
1,45E+08	1001	55503	55504
1,5E+08	1001	57711	57712
1,56E+08	1001	59998	59999
1,61E+08	1001	62097	62098
1,66E+08	1001	64297	64298
1,72E+08	1001	66112	66113
1,77E+08	1001	68229	68230
1,83E+08	1001	70240	70241
1,88E+08	1001	72263	72264
1,93E+08	1001	74247	74248
1,99E+08	1001	76729	76730
2,04E+08	1001	78778	78779
2,09E+08	1001	80284	80285
2,15E+08	1001	82875	82876
2,2E+08	1001	84735	84736

Oraz nakłady czasu:

Etykiety wierszy	Suma z czas
0	564
1	46
2	118
3	104
4	130
5	141
6	164

7	186
8	213
9	253
10	281
11	306
12	303
13	361
14	383
15	374
16	420
17	466
18	440
19	1138
20	535
21	566
22	537
23	1380
24	582
25	605
26	687
27	649
28	676
29	769
30	797
31	909
32	769
33	871
34	814
35	2046
36	877
37	888
38	2210
39	933
40	1052
Suma końcowa	**25543**



W tym przypadku skoki czasowe, również spowodowane były wykorzystaniem zasobów do innych działań komputera.

Porównując obie sytuacje, wyszło nam, że ilość czasu oraz operacji znacznie się zmniejszyła.
Średni przypadek w wyszukiwaniu liniowym można opisać przy pomocy poniższego wzoru :

$$\frac{\sum_{i=1}^{n} i}{n} = \frac{1+2+\ldots+n}{n} = \frac{1}{n} * \frac{1+(1+(n-1)*1)}{2} * n = \frac{1+n}{2} = \boxed{\frac{1}{2} + \frac{n}{2}}$$

Źródło: wykład Algorytmy i struktury danych CS01. - https://moodle2.e-wsb.pl/mod/folder/view.php?id=1939667

Implementacja metody wyszukiwania binarnego.

Metoda wyszukiwania binarnego:

```
public static int BinarySearchPesymistic(int[] vector, int search)
{
    int left = 0;
    int right = vector.Length - 1;

    while (left <= right)
    {
        int currentPosition = left + (right - left) / 2;

        if (vector[currentPosition] == search)
        {
            return currentPosition;
        }

        if (vector[currentPosition] < search)
        {
            left = currentPosition + 1;
        }

        if (vector[currentPosition] > search)
        {
            right = currentPosition - 1;
        }
    }
    return -1;
}
```

Metoda ta jako dominantę również wykorzystuje ilość porównań (skoków) w celu znalezienia szukanej wartości.

Metoda wykorzystująca instrumentację:

```
private static int counter;
public static int BinarySearchPesymistic(int[] vector, int search)...
public static int BinarySearchPesymisticOptions(int[] vector, int search)
{
    int left = 0;
    int right = vector.Length - 1;
    counter = 0;
    while (left <= right)
    {
        counter++;
        int currentPosition = left + (right - left) / 2;

        if (vector[currentPosition] == search)
        {
            return currentPosition;
        }

        if (vector[currentPosition] < search)
        {
            left = currentPosition + 1;
        }

        if (vector[currentPosition] > search)
        {
            right = currentPosition - 1;
        }
    }
```

Kod wykorzystany w eksperymencie przy badaniu przypadku pesymistycznego:

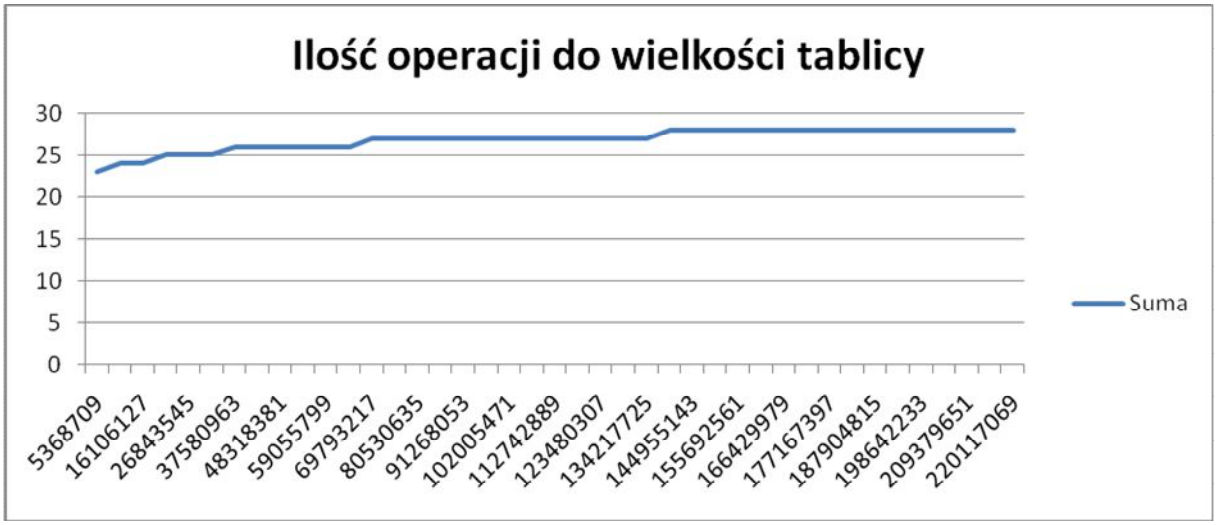
```
static void Main(string[] args)
{
    Random rand = new Random();
    int lookUpValue = 1001;
    int result;

    Console.WriteLine("rozmiar;szukana;wynik;czas;ile operacji");
    for (int k = 5368709; k < (int)Math.Pow(2, 28); k += 5368709)
    {

        int[] tab = new int[k];
        for (int i = 0; i < tab.Length; i++)
        {
            tab[i] = rand.Next(1, 1000);
        }
        Array.Sort(tab); // sortowanie tablicy rosnąco
        long start = Stopwatch.GetTimestamp(); // start czasu
        result = BinarySearchPesymistic(tab, lookUpValue); // pomiar czasu dla binarnego bez instrumentacji
        long stop = Stopwatch.GetTimestamp();   // stop czasu
        result = BinarySearchPesymisticOptions(tab, lookUpValue); // wynik instrumentacji
        Console.WriteLine(k + ";" + lookUpValue + ";" + result + ";" + (stop - start) + ";" + counter); // t
    }
}
```

Zebrane dane przy pomiarze operacji – wykorzystano 40 punktów pomiarowych.

rozmiar	ile operacji
5368709	23
10737418	24
16106127	24
21474836	25
26843545	25
32212254	25

37580963	26
42949672	26
48318381	26
53687090	26
59055799	26
64424508	26
69793217	27
75161926	27
80530635	27
85899344	27
91268053	27
96636762	27
1,02E+08	27
1,07E+08	27
1,13E+08	27
1,18E+08	27
1,23E+08	27
1,29E+08	27
1,34E+08	27
1,4E+08	28
1,45E+08	28
1,5E+08	28
1,56E+08	28
1,61E+08	28
1,66E+08	28
1,72E+08	28
1,77E+08	28
1,83E+08	28
1,88E+08	28
1,93E+08	28
1,99E+08	28
2,04E+08	28
2,09E+08	28
2,15E+08	28
2,2E+08	28

Oraz wykres do zaznaczonych danych:



Zebrane dane przy pomiarze czasu – również dla 40 punktów.

Iteracja	czas	ile operacji
0	719	23
1	7	24
2	7	24
3	10	25
4	8	25
5	8	25
6	9	26
7	8	26
8	10	26
9	8	26
10	9	26
11	9	26
12	9	27
13	9	27
14	10	27
15	9	27
16	9	27
17	9	27
18	8	27
19	10	27
20	15	27
21	20	27
22	9	27
23	9	27
24	9	27
25	9	28
26	10	28
27	21	28
28	9	28
29	9	28
30	10	28
31	10	28
32	11	28
33	17	28
34	12	28
35	17	28
36	14	28
37	21	28
38	10	28
39	10	28
40	15	28

Oraz wykres danych:

W tym przypadku skoki czasowe, również spowodowane były wykorzystaniem zasobów do innych działań komputera.

Już teraz widać, że różnica między wyszukiwaniem liniowym a binarnym w samym tylko wariancie pesymistycznym jest dużo szybsza zarówno pod względem czasu jak i ilości operacji.

Wyszukiwanie binarne – przypadek średni.

Wykorzystany kod:

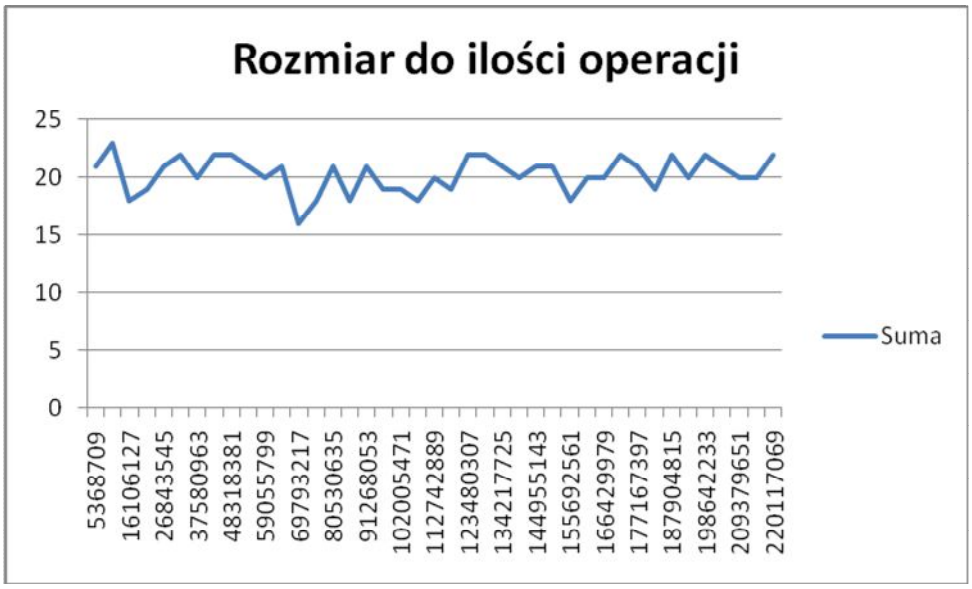
```
static void Main(string[] args)
{
    Random rand = new Random();
    int lookUpValue = 1001;
    int result;

    Console.WriteLine("rozmiar;szukana;wynik;czas;ile operacji");
    for (int k = 5368709; k < (int)Math.Pow(2, 28); k += 5368709)
    {

        int[] tab = new int[k];
        for (int i = 0; i < tab.Length; i++)
        {
            tab[i] = rand.Next(1, 2600000);
        }
        Array.Sort(tab); // sortowanie tablicy rosnąco
        long start = Stopwatch.GetTimestamp(); // start czasu
        result = BinarySearchPesymistic(tab, lookUpValue); // pomiar czasu dla binarnego bez instrumentacji
        long stop = Stopwatch.GetTimestamp();   // stop czasu
        result = BinarySearchPesymisticOptions(tab, lookUpValue); // wynik instrumentacji
        Console.WriteLine(k + ";" + lookUpValue + ";" + result + ";" + (stop - start) + ";" + counter); // tablica; szukana; index; czas; za którym razem
    }
}
```

Przy ocenie przypadku średniego, również zostało wykorzystane 40 punktów pomiarowych.

rozmiar	ile operacji
5368709	21
10737418	23
16106127	18
21474836	19
26843545	21
32212254	22
37580963	20
42949672	22
48318381	22
53687090	21
59055799	20
64424508	21

69793217	16
75161926	18
80530635	21
85899344	18
91268053	21
96636762	19
102005471	19
107374180	18
112742889	20
118111598	19
123480307	22
128849016	22
134217725	21
139586434	20
144955143	21
150323852	21
155692561	18
161061270	20
166429979	20
171798688	22
177167397	21
182536106	19
187904815	22
193273524	20
198642233	22
204010942	21
209379651	20
214748360	20
220117069	22

Poniżej wykres danych do przypadku średniego.



Sposób ułożenia wykresu w ten sposób spowodowany jest losowością jaką wprowadza nam funkcja random. Ilość operacji jednak mieści się w przedziale 16-23.

W przypadku pomiaru czasu sytuacja wygląda następująco:

Średnia złożoność w tym przypadku nie różni się zbytnio od założenia pesymistycznego.

Podsumowanie.

Dla wyszukiwania liczb/elementów w tablicy wydajniejsze w poszukiwaniu liczb/elementów jest metoda binarna. Oszczędza nam to czas oraz zasoby komputera wykorzystując ilość poziomów węzłów drzewa binarnego, tzw. skoków, do ilości elementów w tablicy.

Należy jednak pamiętać o tym, że różnica w wyszukiwaniu przy wariancie średnim i pesymistycznym zarówno czasowo jak i operacyjnie różni się minimalnie.

Aby algorytm był jak najbardziej wydajny, należy jednak pamiętać o zastosowaniu sortowania tablicy.

W samym wyszukiwaniu liniowym różnica między wariantem pesymistycznym a średnim jest już bardziej zauważalna. Mając posortowaną tablicę oraz pewność że znajduje się w niej wyszukiwana pozycja szybciej dotrzemy do szukanej wartości.

Osobnym przypadkiem jest tutaj sytuacja w której wyszukiwany element jest zawsze na tej samej pozycji w tablicy, jednak ten przypadek nie był tutaj badany.