Projekt 1

Zadanie:

Zweryfikować przedstawioną na wykładzie ocenę średniej i pesymistycznej złożoności wyszukiwania liniowego i binarnego przy użyciu instrumentacji, oraz pomiarów czasu.

Implementacja metody wyszukiwania liniowego:

```
for (int i = 0; i < Vector.Length; i++)
{
    if (i == Number) return true;
}
return false;
```

Implementacja metody wyszukiwania binarnego:

```
int Left = 0, Right = Vector.Length - 1, Middle;
while (Left <= Right)
{
   Middle = (Left + Right) / 2;

  if (Vector[Middle] == Number) return true;

  if (Vector[Middle] > Number) Right = Middle - 1;
  else Left = Middle + 1;

}
return false;
```

Na potrzeby eksperymentu wprowadzę następujące oznaczenia używane w dalszej części opisu:
- operacja porównania „równa się”: **= =**
- operacja porównania „mniejsze od”: **<**
- operacja porównania „mniejsze/równe”: **<=**
- operacja porównania „większe/równe”: **>=**
- operacja porównania „większe od”: **>**
- operacja przypisania: **=**
- operacja zwiększania o 1: **++**
- operacja dzielenia: **/**

Poniżej będę przedstawiał jedynie fragmenty kodu, dlatego pragnę zaznaczyć, iż każdy z fragmentów znajduje się w miejscu pokazanym poniżej (oznaczonym jako /// TUTAJ ZNAJDUJĄ SIĘ PRZEDSTAWIONE FRAGMENTY KODU ///), a zmienne NumberOfOperations, ElapsedSeconds, oraz NumberOfTests są zerowane przed każdym wywołaniem funkcji. Całość kodu znajduje się na końcu dokumentu tekstowego.

W eksperymentach wykorzystano 10 punktów pomiarowych o wartościach od 26843545 do 268435450.

Dla uzyskania uśrednionych wyników w ocenach średniej złożoności przeprowadzono wiele operacji wyszukiwań; pierwsza szukana liczba była zerem, a kolejne szukane liczby powiększane były o milion, aż do momentu w którym kolejne powiększenie przekroczyłoby rozmiar tablicy.

```
using System;
using System.Diagnostics;

namespace Project_1
{
    class MainClass
    {
        static ulong NumberOfOperations;
        static double ElapsedSeconds;
        static ulong NumberOfTests;


////////////////////////////// TUTAJ ZNAJDUJĄ SIĘ PRZEDSTAWIONE FRAGMENTY KODU ////////////////////////////////


                public static void Main(string[] args)
        {
           int choice = 0;
           int[] ArraySize = new int[10];
           ArraySize[0] = 26843545;

           for (int i = 1; i < ArraySize.Length; i++)
           {
               ArraySize[i] = ArraySize[i-1] + 26843545;
           }

           do
           {
               Console.WriteLine("Jaką oprację chcesz wykonać?\n");
               Console.WriteLine("1. Wyszukiwanie liniowe - pesymistyczne - instrumentacja");
               Console.WriteLine("2. Wyszukiwanie liniowe - pesymistyczne - czas\n");
               Console.WriteLine("3. Wyszukiwanie liniowe - średnie - instrumentacja");
               Console.WriteLine("4. Wyszukiwanie liniowe - średnie - czas\n");
               Console.WriteLine("5. Wyszukiwanie binarne - pesymistyczne - instrumentacja");
               Console.WriteLine("6. Wyszukiwanie binarne - pesymistyczne - czas\n");
               Console.WriteLine("7. Wyszukiwanie binarne - średnie - instrumentacja");
               Console.WriteLine("8. Wyszukiwanie binarne - średnie - czas\n");
               Console.WriteLine("0. Zakończ prorgram\n");
               Console.Write("Twó wybó:");
               choice = int.Parse(Console.ReadLine());
               Console.WriteLine("\n\n");

               if (choice == 1
                   || choice == 2
                   || choice == 3
                   || choice == 4
                   || choice == 5
                   || choice == 6
                   || choice == 7
                   || choice == 8)
               {
                   switch (choice)
                   {
                       case 1:
                       case 3:
                       case 5:
                       case 7:
                           Console.WriteLine("n\t\t\tOperations");
                           break;

                       case 2:
                       case 4:
                       case 6:
                       case 8:
                           Console.WriteLine("n\t\tElapsed seconds");
                           break;
                   }
```

```
                           for (int i = 0; i < ArraySize.Length; i++)
                    {
                          int[] Vector = new int[ArraySize[i]];
                          for (int j = 0; j < ArraySize[i]; j++)
                          {
                              Vector[j] = j;
                          }

                          NumberOfOperations = 0;
                          ElapsedSeconds = 0;
                          NumberOfTests = 0;

                          switch (choice)
                                 {
//////////////////////////////////////  MIEJSCE WYWYOŁYWANIA FUNKCJI ////////////////////////////////////////
                              case 1:
                                  LinearMaxInstr(Vector);
                                  Console.WriteLine("{0}\t\t{1}", Vector.Length, NumberOfOperations);
                                  break;

                              case 2:
                                  LinearMaxTime(Vector);
                                  Console.WriteLine("{0}\t{1}", Vector.Length, ElapsedSeconds.ToString("F4"));
                                  break;

                              case 3:
                                  LinearAvgInstr(Vector);
                                  Console.WriteLine("{0}\t\t{1}", Vector.Length, NumberOfOperations);
                                  break;

                              case 4:
                                  LinearAvgTime(Vector);
                                  Console.WriteLine("{0}\t{1}", Vector.Length, ElapsedSeconds.ToString("F4"));
                                  break;

                              case 5:
                                  BinaryMaxInstr(Vector);
                                  Console.WriteLine("{0}\t\t{1}", Vector.Length, NumberOfOperations);
                                  break;

                              case 6:
                                  BinaryMaxTime(Vector);
                                  Console.WriteLine("{0}\t{1}", Vector.Length, ElapsedSeconds.ToString("F4"));
                                  break;

                              case 7:
                                  BinaryAvgInstr(Vector);
                                  Console.WriteLine("{0}\t\t{1}", Vector.Length, NumberOfOperations);
                                  break;

                              case 8:
                                  BinaryAvgTime(Vector);
                                  Console.WriteLine("{0}\t{1}", Vector.Length, ElapsedSeconds.ToString("F4"));
                                  break;

                                  }
                             }
                        }
                    } while (choice != 0);
                }
        }
}
```

1. Ocena pesymistycznej złożoności wyszukiwania liniowego

a) Przy użyciu instrumentacji:

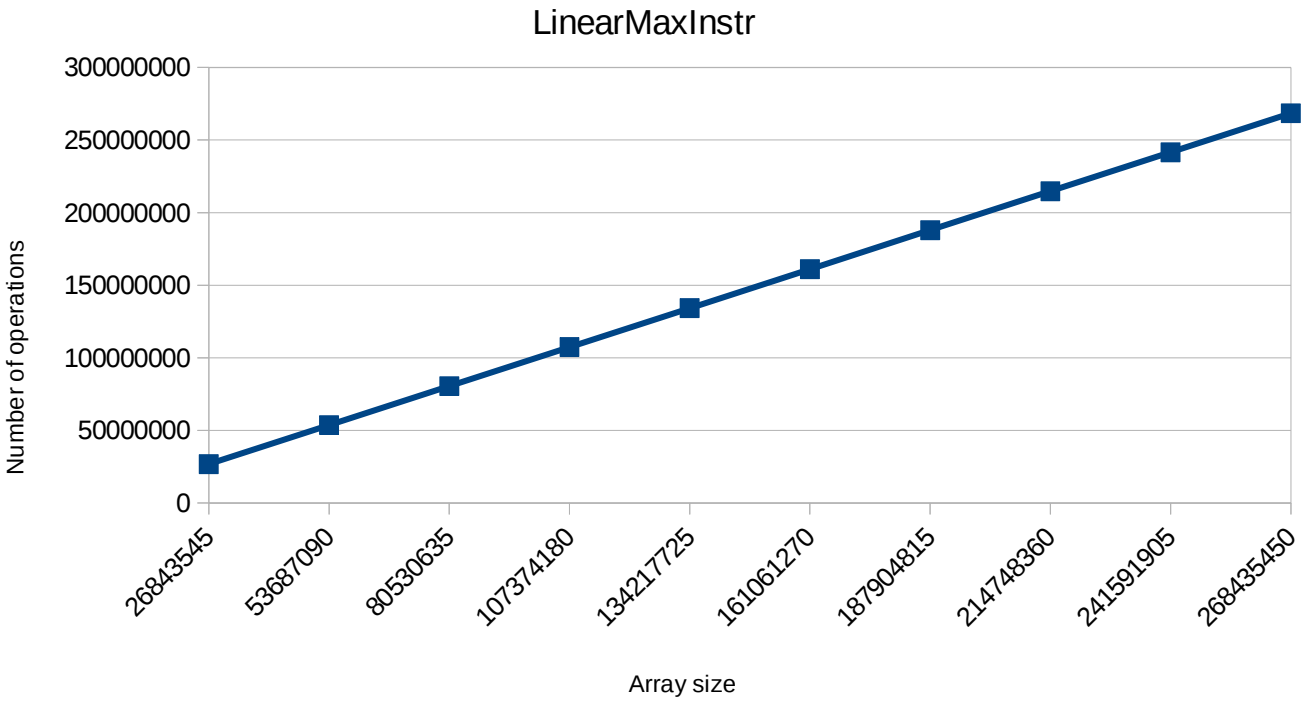
Do oceny złożoności algorytmu wybrano zliczanie liczby = = jako operacji dominującej.
Liczba = zawsze wynosi 1.
Liczba < jest równa liczbie = =.
Liczba operacji ++ jest mniejsza o 1 od operacji = =

```
static bool LinearSearchInstr(int[] Vector, int Number)
{
      for (int i = 0; i < Vector.Length; i++)
      {
            NumberOfOperations++;
            if (i == Number) return true;
      }
      return false;
}


static void LinearMaxInstr(int[] Vector)
{
      LinearSearchInstr(Vector, Vector.Length - 1);
}
```

Zebrane dane:

n	Operations
26843545	26843545
53687090	53687090
80530635	80530635
107374180	107374180
134217725	134217725
161061270	161061270
187904815	187904815
214748360	214748360
241591905	241591905
268435450	268435450

Zależność liczby operacji od wielkości tablicy:



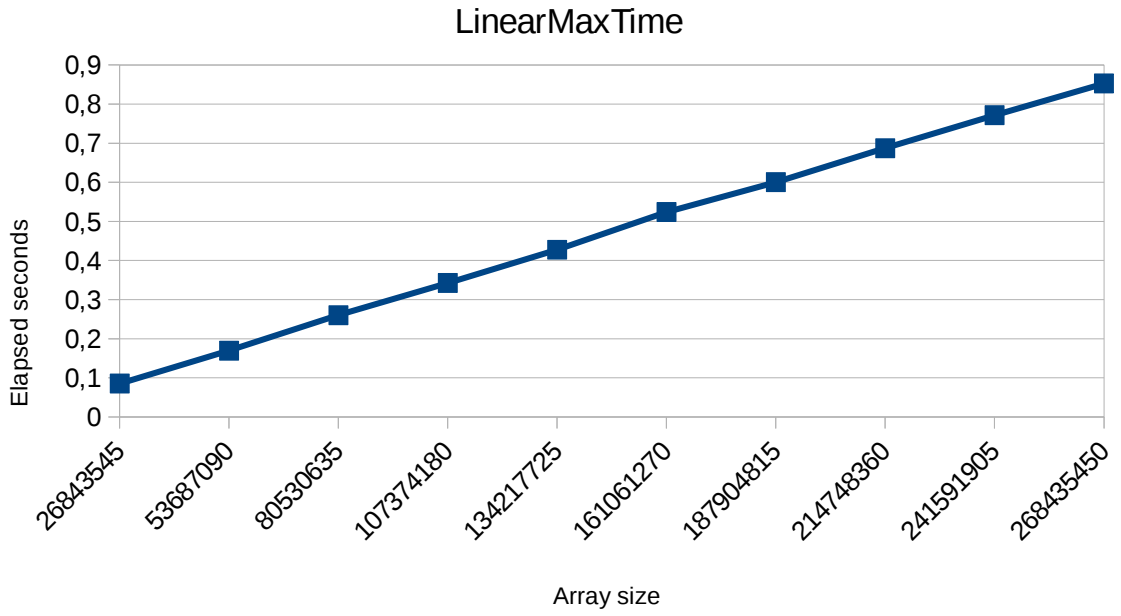
b) Przy użyciu pomiarów czasu:

```
static bool LinearSearchTime(int[] Vector, int Number)
{
      for (int i = 0; i < Vector.Length; i++)
      {
            if (i == Number) return true;
      }
      return false;
}

static void LinearMaxTime(int[] Vector)
{
      long ElapsedTime=0, MinTime=long.MaxValue, MaxTime=long.MinValue,
IterationElapsedTime;
       for (int NumberOfControlTests = 0; NumberOfControlTests < 12;
NumberOfControlTests++)
       {
            long StartingTime = Stopwatch.GetTimestamp();

            LinearSearchTime(Vector, Vector.Length - 1);

            long EndingTime = Stopwatch.GetTimestamp();
            IterationElapsedTime = EndingTime - StartingTime;
            ElapsedTime += IterationElapsedTime;

            if (IterationElapsedTime < MinTime) MinTime = IterationElapsedTime;
            if (IterationElapsedTime > MaxTime) MaxTime = IterationElapsedTime;
       }

        ElapsedTime -= (MinTime + MaxTime);
        ElapsedSeconds = ElapsedTime * (1.0 / (10 * Stopwatch.Frequency));
}
```

Zebrane dane:

n	Elapsed sec
26843545	0,0854
53687090	0,1696
80530635	0,2602
107374180	0,3424
134217725	0,4276
161061270	0,5241
187904815	0,6003
214748360	0,6872
241591905	0,7717
268435450	0,853

Zależność czasu wykonania od wielkości tablicy:



2. Ocena pesymistycznej złożoności wyszukiwania binarnego:

Do oceny złożoności algorytmu wybrano zliczanie liczby = = jako operacji dominującej.

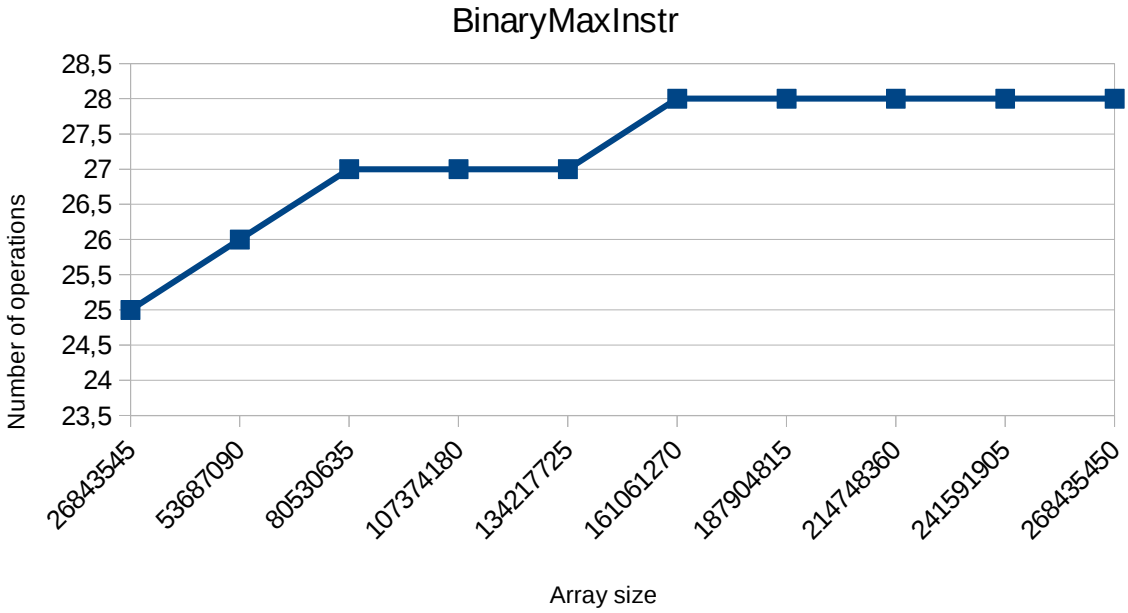
a) Przy użyciu instrumentacji:

```
static bool BinarySearchInstr(int[] Vector, int Number)
{
      int Left = 0, Right = Vector.Length - 1, Middle;
      while (Left <= Right)
      {
            Middle = (Left + Right) / 2;

            NumberOfOperations++;
            if (Vector[Middle] == Number) return true;

            if (Vector[Middle] > Number) Right = Middle - 1;
            else Left = Middle + 1;
      }
      return false;
}

static void BinaryMaxInstr(int[] Vector)
{
      BinarySearchInstr(Vector, Vector.Length - 1);
}
```

Zebrane dane:

n	Operations
26843545	25
53687090	26
80530635	27
107374180	27
134217725	27
161061270	28
187904815	28
214748360	28
241591905	28
268435450	28

Zależność liczby operacji od wielkości tablicy:



b) Przy użyciu pomiarów czasu:

```
static bool BinarySearchTime(int [] Vector, int Number)
{
      Left = 0, Right = Vector.Length - 1, Middle;
      while (Left <= Right)
      {
            Middle = (Left + Right) / 2;

            if (Vector[Middle] == Number) return true;
            if (Vector[Middle] > Number) Right = Middle - 1;
            else Left = Middle + 1;
      }
      return false;
}

static void BinaryMaxTime(int[] Vector)
{
      long ElapsedTime=0, MinTime=long.MaxValue, MaxTime=long.MinValue, IterationElapsedTime;
      for (int NumberOfControlTests = 0; NumberOfControlTests < 12; NumberOfControlTests++)
      {
      long StartingTime = Stopwatch.GetTimestamp();

            BinarySearchTime(Vector, Vector.Length - 1);

            long EndingTime = Stopwatch.GetTimestamp();
            IterationElapsedTime = EndingTime - StartingTime;
            ElapsedTime += IterationElapsedTime;

            if (IterationElapsedTime < MinTime) MinTime = IterationElapsedTime;
            if (IterationElapsedTime > MaxTime) MaxTime = IterationElapsedTime;
        }

        ElapsedTime -= (MinTime + MaxTime);
        ElapsedSeconds = ElapsedTime * (1.0 / (10 * Stopwatch.Frequency));
}
```

Zebrane dane:

n	Elapsed seconds
26843545	0
53687090	0
80530635	0
107374180	0
134217725	0
161061270	0
187904815	0
214748360	0
241591905	0
268435450	0

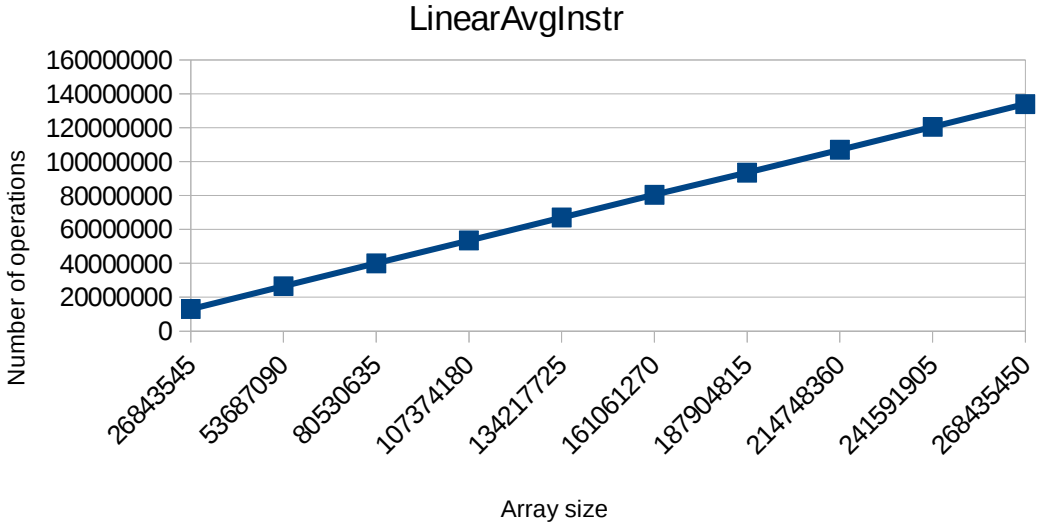
3. Ocena średniej złożoności wyszukiwania liniowego

a) Przy użyciu instrumentacji:

```
static bool LinearSearchInstr(int[] Vector, int Number)
{
	for (int i = 0; i < Vector.Length; i++)
	{
		NumberOfOperations++;
	  if (i == Number) return true;
	}
	return false;
}

static void LinearAvgInstr(int[] Vector)
{
	for (int k = 0; k < Vector.Length; k += 1000000)
	{
		NumberOfTests++;
		LinearSearchInstr(Vector, k);
	}
	NumberOfOperations = (NumberOfOperations / NumberOfTests);
}
```

Zebrane dane:

n	Operations
26843545	13000001
53687090	26500001
80530635	40000001
107374180	53500001
134217725	67000001
161061270	80500001
187904815	93500001
214748360	107000001
241591905	120500001
268435450	134000001



Zależność liczby operacji od wielkości tablicy:

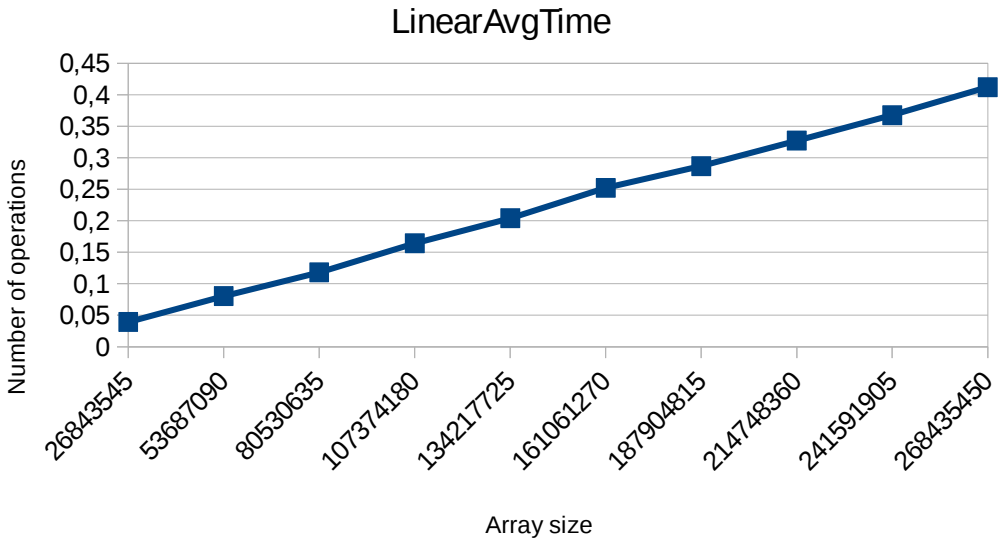
b) Przy użyciu pomiarów czasu:

```
static bool LinearSearchTime(int[] Vector, int Number)
{
      for (int i = 0; i < Vector.Length; i++)
      {
            if (i == Number) return true;
      }
      return false;
}

static void LinearAvgTime(int[] Vector)
{
      for (int k = 0; k < Vector.Length; k += 1000000)
      {
            NumberOfTests++;
            long ElapsedTime=0, MinTime=long.MaxValue, MaxTime=long.MinValue, IterationElapsedTime;

            for (int NumberOfControlTests = 0; NumberOfControlTests < 12; NumberOfControlTests++)
            {
                  long StartingTime = Stopwatch.GetTimestamp();

                  LinearSearchTime(Vector, k);

                  long EndingTime = Stopwatch.GetTimestamp();
                  IterationElapsedTime = EndingTime - StartingTime;
                  ElapsedTime += IterationElapsedTime;

                  if (IterationElapsedTime < MinTime) MinTime = IterationElapsedTime;
                  if (IterationElapsedTime > MaxTime) MaxTime = IterationElapsedTime;
            }

            ElapsedTime -= (MinTime + MaxTime);
            ElapsedSeconds += ElapsedTime * (1.0 / (10 * Stopwatch.Frequency));
      }
      ElapsedSeconds = (ElapsedSeconds / NumberOfTests);
}
```

Zebrane dane:

n	Elapsed sec
26843545	0,0392
53687090	0,0802
80530635	0,118
107374180	0,1643
134217725	0,204
161061270	0,2521
187904815	0,2868
214748360	0,3273
241591905	0,3676
268435450	0,4121

Zależność czasu wykonania od wielkości tablicy:



4. Ocena średniej złożoności wyszukiwania binarnego

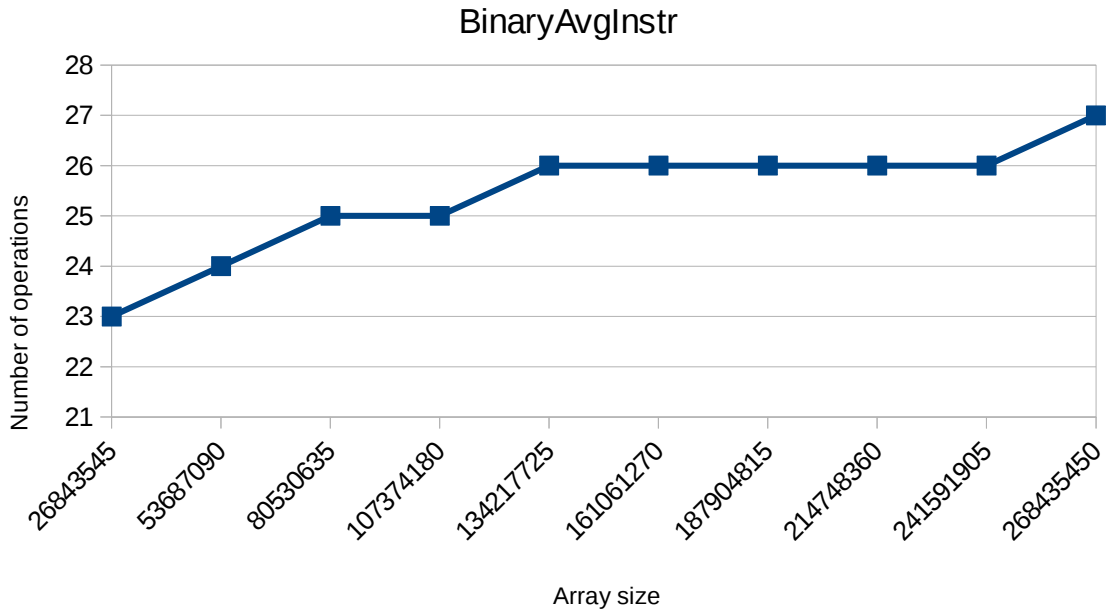
a) Przy użyciu instrumentacji:

```
static bool BinarySearchInstr(int[] Vector, int Number)
{
      int Left = 0, Right = Vector.Length – 1, Middle;
      while (Left <= Right)
      {
            Middle = (Left + Right) / 2;

            NumberOfOperations++;
            if (Vector[Middle] == Number) return true;
            if (Vector[Middle] > Number) Right = Middle - 1;
            else Left = Middle + 1;

      }
      return false;
}

static void BinaryAvgInstr(int[] Vector)
{
      for (int k = 0; k < Vector.Length; k += 1000000)
      {
            NumberOfTests++;
            BinarySearchInstr(Vector, k);
      }
      NumberOfOperations = (NumberOfOperations / NumberOfTests);
}
```

Zebrane dane:

n	Operations
26843545	23
53687090	24
80530635	25
107374180	25
134217725	26
161061270	26
187904815	26
214748360	26
241591905	26
268435450	27

Zależność liczby operacji od wielkości tablicy:



a) Przy użyciu pomiarów czasu:

```
static void BinaryAvgTime(int[] Vector)
{
	for (int k = 0; k < Vector.Length; k += 1000000)
	{
		NumberOfTests++;
		long ElapsedTime=0, MinTime=long.MaxValue, MaxTime=long.MinValue, IterationElapsedTime;

		for (int NumberOfControlTests = 0; NumberOfControlTests < 12; NumberOfControlTests++)
		{
			long StartingTime = Stopwatch.GetTimestamp();

			BinarySearchTime(Vector, k);

			long EndingTime = Stopwatch.GetTimestamp();
			IterationElapsedTime = EndingTime - StartingTime;
			ElapsedTime += IterationElapsedTime;

			if (IterationElapsedTime < MinTime) MinTime = IterationElapsedTime;
			if (IterationElapsedTime > MaxTime) MaxTime = IterationElapsedTime;
		}

		ElapsedTime -= (MinTime + MaxTime);
		ElapsedSeconds += ElapsedTime * (1.0 / (10 * Stopwatch.Frequency));
	}
	ElapsedSeconds = (ElapsedSeconds / NumberOfTests);
}
```

Zebrane dane:

n	Elapsed second
26843545	0
53687090	0
80530635	0
107374180	0
134217725	0
161061270	0
187904815	0
214748360	0
241591905	0
268435450	0

W eksperymentach wykonano pomiary na komputerze wyposażonym w procesor Intel i5-6300U z wykorzystaniem MonoDevelop. Projekt został skompilowany z ustawieniami Relase/Any CPU.

Wnioski:

Czas wyszukiwania, oraz liczba operacji wyszukiwania binarnego są nieporównywalnie mniejsze od wartości prezentowanych przez wyszukiwanie liniowe.

Wyszukiwanie liniowe jest bardzo nieefektywne w porównaniu do wyszukiwania binarnego.

Cały kod:

```
using System;
using System.Diagnostics;

namespace Project_1
{
    class MainClass
    {
        static ulong NumberOfOperations;
        static double ElapsedSeconds;
        static ulong NumberOfTests;

        static bool LinearSearchInstr(int[] Vector, int Number)
        {
            for (int i = 0; i < Vector.Length; i++)
            {
                NumberOfOperations++;
                if (i == Number) return true;
            }
            return false;
        }

        static bool LinearSearchTime(int[] Vector, int Number)
        {
            for (int i = 0; i < Vector.Length; i++)
            {
                if (i == Number) return true;
            }
            return false;
        }

        static bool BinarySearchInstr(int[] Vector, int Number)
        {
            int Left = 0, Right = Vector.Length - 1, Middle;
            while (Left <= Right)
            {
                Middle = (Left + Right) / 2;

                NumberOfOperations++;
                if (Vector[Middle] == Number)return true;

                if (Vector[Middle] > Number) Right = Middle - 1;
                else Left = Middle + 1;

            }
            return false;
        }

        static bool BinarySearchTime(int [] Vector, int Number)
        {
            int Left = 0, Right = Vector.Length - 1, Middle;
            while (Left <= Right)
            {
                Middle = (Left + Right) / 2;

                if (Vector[Middle] == Number) return true;

                if (Vector[Middle] > Number) Right = Middle - 1;
                else Left = Middle + 1;

            }
            return false;
        }
```

```
//1. LinearMaxInstr ---------------------------------------------------------------------------------
static void LinearMaxInstr(int[] Vector)
{
    LinearSearchInstr(Vector, Vector.Length - 1);
}

//2. LinearMaxTime  ---------------------------------------------------------------------------------
static void LinearMaxTime(int[] Vector)
{
    long ElapsedTime = 0, MinTime = long.MaxValue, MaxTime = long.MinValue, IterationElapsedTime;
    for (int NumberOfControlTests = 0; NumberOfControlTests < 12; NumberOfControlTests++)
    {
        long StartingTime = Stopwatch.GetTimestamp();
        LinearSearchTime(Vector, Vector.Length - 1);
        long EndingTime = Stopwatch.GetTimestamp();

        IterationElapsedTime = EndingTime - StartingTime;
        ElapsedTime += IterationElapsedTime;

        if (IterationElapsedTime < MinTime) MinTime = IterationElapsedTime;
        if (IterationElapsedTime > MaxTime) MaxTime = IterationElapsedTime;
    }

    ElapsedTime -= (MinTime + MaxTime);
    ElapsedSeconds = ElapsedTime * (1.0 / (10 * Stopwatch.Frequency));
}

//3. LinearAvgInstr ---------------------------------------------------------------------------------
static void LinearAvgInstr(int[] Vector)
{
    for (int k = 0; k < Vector.Length; k += 1000000)
    {
        NumberOfTests++;
        LinearSearchInstr(Vector, k);
    }
    NumberOfOperations = (NumberOfOperations / NumberOfTests);
}

//4. LinearAvgTime  ---------------------------------------------------------------------------------
static void LinearAvgTime(int[] Vector)
{
    for (int k = 0; k < Vector.Length; k += 1000000)
    {
        NumberOfTests++;
        long ElapsedTime = 0, MinTime = long.MaxValue, MaxTime = long.MinValue, IterationElapsedTime;

        for (int NumberOfControlTests = 0; NumberOfControlTests < 12; NumberOfControlTests++)
        {
            long StartingTime = Stopwatch.GetTimestamp();

            LinearSearchTime(Vector, k);

            long EndingTime = Stopwatch.GetTimestamp();
            IterationElapsedTime = EndingTime - StartingTime;
            ElapsedTime += IterationElapsedTime;

            if (IterationElapsedTime < MinTime) MinTime = IterationElapsedTime;
            if (IterationElapsedTime > MaxTime) MaxTime = IterationElapsedTime;
        }

        ElapsedTime -= (MinTime + MaxTime);
        ElapsedSeconds += ElapsedTime * (1.0 / (10 * Stopwatch.Frequency));
    }
    ElapsedSeconds = (ElapsedSeconds / NumberOfTests);
}

//5. BinaryMaxInstr ---------------------------------------------------------------------------------
static void BinaryMaxInstr(int[] Vector)
{
    BinarySearchInstr(Vector, Vector.Length - 1);
}
```

```
//6. BinaryMaxTime  ------------------------------------------------------------------------------
static void BinaryMaxTime(int[] Vector)
{
    long ElapsedTime = 0, MinTime = long.MaxValue, MaxTime = long.MinValue, IterationElapsedTime;
    for (int NumberOfControlTests = 0; NumberOfControlTests < 12; NumberOfControlTests++)
    {
        long StartingTime = Stopwatch.GetTimestamp();

        BinarySearchTime(Vector, Vector.Length - 1);

        long EndingTime = Stopwatch.GetTimestamp();
        IterationElapsedTime = EndingTime - StartingTime;
        ElapsedTime += IterationElapsedTime;

        if (IterationElapsedTime < MinTime) MinTime = IterationElapsedTime;
        if (IterationElapsedTime > MaxTime) MaxTime = IterationElapsedTime;
    }

    ElapsedTime -= (MinTime + MaxTime);
    ElapsedSeconds = ElapsedTime * (1.0 / (10 * Stopwatch.Frequency));
}

//7. BinaryAvgInstr ------------------------------------------------------------------------------
static void BinaryAvgInstr(int[] Vector)
{
    for (int k = 0; k < Vector.Length; k += 1000000)
    {
        NumberOfTests++;
        BinarySearchInstr(Vector, k);
    }
    NumberOfOperations = (NumberOfOperations / NumberOfTests);
}

//8. BinaryAvgTime  ------------------------------------------------------------------------------
static void BinaryAvgTime(int[] Vector)
{
    for (int k = 0; k < Vector.Length; k += 1000000)
    {
        NumberOfTests++;
        long ElapsedTime = 0, MinTime = long.MaxValue, MaxTime = long.MinValue, IterationElapsedTime;

        for (int NumberOfControlTests = 0; NumberOfControlTests < 12; NumberOfControlTests++)
        {
            long StartingTime = Stopwatch.GetTimestamp();

            BinarySearchTime(Vector, k);

            long EndingTime = Stopwatch.GetTimestamp();
            IterationElapsedTime = EndingTime - StartingTime;
            ElapsedTime += IterationElapsedTime;

            if (IterationElapsedTime < MinTime) MinTime = IterationElapsedTime;
            if (IterationElapsedTime > MaxTime) MaxTime = IterationElapsedTime;
        }

        ElapsedTime -= (MinTime + MaxTime);
        ElapsedSeconds += ElapsedTime * (1.0 / (10 * Stopwatch.Frequency));
    }
    ElapsedSeconds = (ElapsedSeconds / NumberOfTests);
}

//------- MAIN ------------------------------------------------------------------------
public static void Main(string[] args)
{
    int choice = 0;
    int[] ArraySize = new int[10];
    ArraySize[0] = 26843545;

    for (int i = 1; i < ArraySize.Length; i++)
    {
        ArraySize[i] = ArraySize[i-1] + 26843545;
    }

    do
    {
        Console.WriteLine("Jaką oprację chcesz wykonać?\n");
        Console.WriteLine("1. Wyszukiwanie liniowe - pesymistyczne - instrumentacja");
        Console.WriteLine("2. Wyszukiwanie liniowe - pesymistyczne - czas\n");
        Console.WriteLine("3. Wyszukiwanie liniowe - średnie - instrumentacja");
        Console.WriteLine("4. Wyszukiwanie liniowe - średnie - czas\n");
        Console.WriteLine("5. Wyszukiwanie binarne - pesymistyczne - instrumentacja");
        Console.WriteLine("6. Wyszukiwanie binarne - pesymistyczne - czas\n");
        Console.WriteLine("7. Wyszukiwanie binarne - średnie - instrumentacja");
        Console.WriteLine("8. Wyszukiwanie binarne - średnie - czas\n");
        Console.WriteLine("0. Zakończ prorgram\n");
        Console.Write("Twó wybó:");
        choice = int.Parse(Console.ReadLine());
        Console.WriteLine("\n\n");
```

```
            if (choice == 1
                || choice == 2
                || choice == 3
                || choice == 4
                || choice == 5
                || choice == 6
                || choice == 7
                || choice == 8)
            {
                switch (choice)
                {
                    case 1:
                    case 3:
                    case 5:
                    case 7:
                        Console.WriteLine("n\t\t\tOperations");
                        break;

                    case 2:
                    case 4:
                    case 6:
                    case 8:
                        Console.WriteLine("n\t\tElapsed seconds");
                        break;
                }


                for (int i = 0; i < ArraySize.Length; i++)
                {
                    int[] Vector = new int[ArraySize[i]];
                    for (int j = 0; j < ArraySize[i]; j++)
                    {
                        Vector[j] = j;
                    }

                    NumberOfOperations = 0;
                    ElapsedSeconds = 0;
                    NumberOfTests = 0;

                    // ----- MIEJSCE WYWYOŁYWANIA FUNKCJI
------------------------------------------------------------
                    switch (choice)
                    {
                        case 1:
                            LinearMaxInstr(Vector);
                            Console.WriteLine("{0}\t\t{1}", Vector.Length, NumberOfOperations);
                            break;

                        case 2:
                            LinearMaxTime(Vector);
                            Console.WriteLine("{0}\t{1}", Vector.Length, ElapsedSeconds.ToString("F4"));
                            break;

                        case 3:
                            LinearAvgInstr(Vector);
                            Console.WriteLine("{0}\t\t{1}", Vector.Length, NumberOfOperations);
                            break;

                        case 4:
                            LinearAvgTime(Vector);
                            Console.WriteLine("{0}\t{1}", Vector.Length, ElapsedSeconds.ToString("F4"));
                            break;

                        case 5:
                            BinaryMaxInstr(Vector);
                            Console.WriteLine("{0}\t\t{1}", Vector.Length, NumberOfOperations);
                            break;

                        case 6:
                            BinaryMaxTime(Vector);
                            Console.WriteLine("{0}\t{1}", Vector.Length, ElapsedSeconds.ToString("F4"));
                            break;

                        case 7:
                            BinaryAvgInstr(Vector);
                            Console.WriteLine("{0}\t\t{1}", Vector.Length, NumberOfOperations);
                            break;

                        case 8:
                            BinaryAvgTime(Vector);
                            Console.WriteLine("{0}\t{1}", Vector.Length, ElapsedSeconds.ToString("F4"));
                            break;
                    }
                }
```

```
                    if (choice != 0)
                    {
                        Console.WriteLine("\n\n#########################################################################################
########################################");
                    }
                }
                else if (choice == 0)
                {
                    Console.WriteLine("Koniec programu");
                }
                else
                {
                    Console.Write("\n\n\n\nDokonałeś złego wyboru, naciśnij dowolny przycisk i spróuj ponownie\n");
                }
            } while (choice != 0);
        }
    }
}
```