|  |  |  |  |
| --- | --- | --- | --- |
| МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені Тараса Шевченка  ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ  **Кафедра програмних систем і технологій**  Дисципліна  **«** **Структури даних, аналіз і алгоритми  комп'ютерної обробки інформації »**  **Лабораторна робота № 1** | | | |
| **Виконав:** | Фесак Андрій Віталійович | **Перевірив**: | Бичков Олексій Сергійович |
| Група | ІПЗ-12 | Дата перевірки |  |
| Форма навчання | денна | Оцінка |  |
| Спеціальність | 121 |
| 2022 | | | |

**Мета**: аналіз швидкодії алгоритмів пошуку на різних структурах даних для дослідження питання стосовно доцільності їх використання.

**Умова задачі:**

Написати програму мовою C# з можливістю вибору різних алгоритмів пошуку. Продемонструвати роботу (ефективність, час виконання) програм на різних структурах даних (масив, лінійний зв’язаний список), з різними умовами, що забезпечують зменшення часу виконання. Навести аналіз отриманих результатів.

Реалізувати алгоритми:

• пошуку перебором елемента масиву, що дорівнює заданому значенню.

• пошуку з бар'єром елемента масиву, що дорівнює заданому значенню.

• бінарного пошуку елемента масиву рівного заданому значенню.

• бінарного пошуку елемента масиву, рівного заданому значенню, в якій нове значення індексу m визначалося б не як середнє значення між L і R, а згідно з правилом золотого перерізу

**Аналіз задачі:**    
Нам потрібно проаналізувати роботу різних структур даних(зв’язний список та масиву даних) на алгоритмах пошуку. Для цього ми створимо статичний клас «*Search*», де реалізуємо наші алгоритми. У основній частині коду ми повинні застосувати клас «*Random*» та «*Stopwatch*»:

* *Random –* клас у мові c#, який дає змогу нам генерувати псевдовипадкові числа
* *Stopwatch –* клас у мові c#, який ми використаємо для підрахунку часу виконання алгоритму.

З цим базисом ми можемо перейти до частини підготовки для роботи:

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**Структура основних вхідних та вихідних даних:**

Для справедливої оцінки роботи всі алгоритми повинні бути в однакових умовах і тому всі вони будуть опрацьовувати один і той самий набір даних. У цьому випадку цей код, який повинен генерувати дані, які ми будемо опрацьовувати у алгоритмах буде справедливим для всього аналізу.

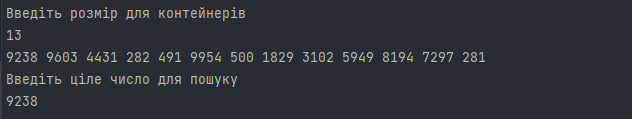
*var* interval = *new* Stopwatch();  
Random ran = *new* Random();  
*int* size;  
Console.WriteLine("Введiть розмiр для контейнерiв");  
*try*{  
 size = *int*.Parse(Console.ReadLine());  
}  
*catch* (Exception e)  
{  
 Console.WriteLine(e);  
 *throw*;  
}  
*int*[] arr = *new int* [size];  
List<*int*> list = *new* List<*int*>(size);  
*for* (*int* i = 0; i < arr.Length; i++)  
{  
 arr[i] = ran.Next(9999);  
 list.Add(arr[i]);  
 Console.Write($"{arr[i]} ");  
}  
  
Console.WriteLine("\nВведiть цiле число для пошуку");  
*int* number;  
*try*{  
 number = *int*.Parse(Console.ReadLine());  
}  
*catch* (Exception e)  
{  
 Console.WriteLine(e);  
 *throw*;  
}

Ми створили масив та список, які заповнені однаковими даними:

* Розмір контейнера:довільне натуральне н
* Набір вхідних даних: контейнер цілочисленних данних з н елементів

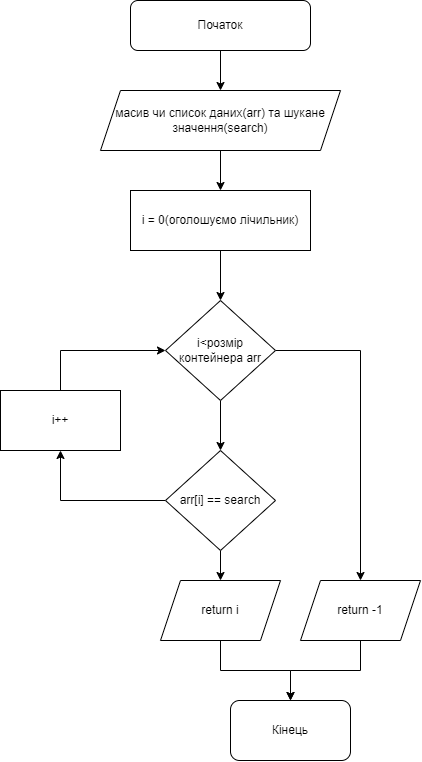
Також ми задали значення для пошуку, яке також є однаковим для всіх алгоритмів та структур: одне з значень в конейнері

Приклад опису цих дій в консолі:



**1. Пошук перебором елемента масиву, що дорівнює заданому значенню:**

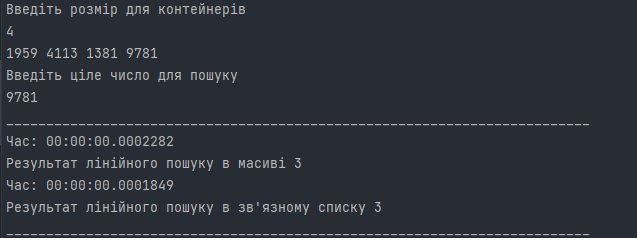
**Алгоритм:**



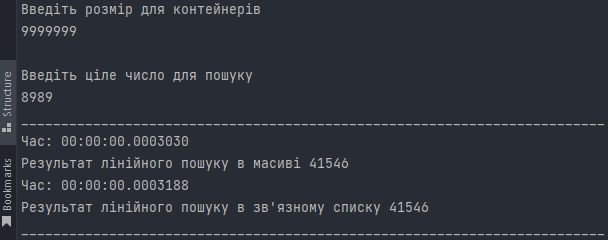
**Текст програми:**

*//Лінійний пошук  
//Повинен повернути індекс шуканого значення, або -1 у випадку відсутності значення.  
//\_\_\_\_Масив\_\_\_*interval.Start();  
*int* index = Search.Linear\_Search(arr, number);  
interval.Stop();  
Console.WriteLine($"Час: {interval.Elapsed}");  
*if* (index == -1)  
{  
 Console.WriteLine("Значення не знайдено");  
}  
*else*{  
 Console.WriteLine($"Результат лiнiйного пошуку в масивi {index}");  
}  
*//\_\_\_\_Ліст\_\_\_*interval.Start();  
index = Search.Linear\_Search(list, number);  
interval.Stop();  
Console.WriteLine($"Час: {interval.Elapsed}");  
*if* (index == -1)  
{  
 Console.WriteLine("Значення не знайдено");  
}  
*else*{  
 Console.WriteLine($"Результат лiнiйного пошуку в зв'язному списку {index}");  
}

**Набір тестів:**

На 4 елементах: 

На 9999999 елементах:

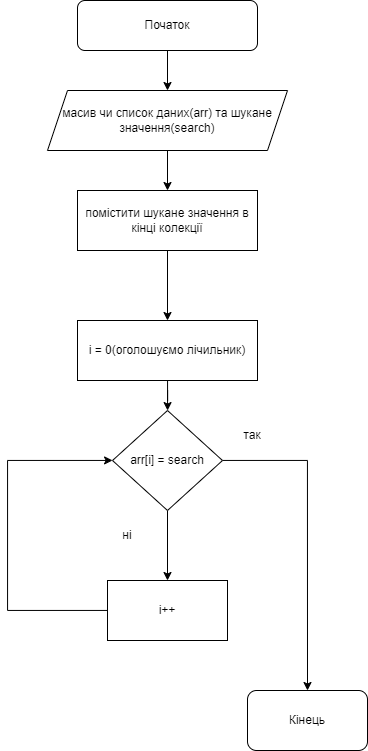


**Результат:**

Лінійний пошук краще працює на масиві. Це пов’язано з будовою зв’язного списку

**2. Пошук з бар'єром елемента масиву, що дорівнює заданому значенню.:**

**Алгоритм:**

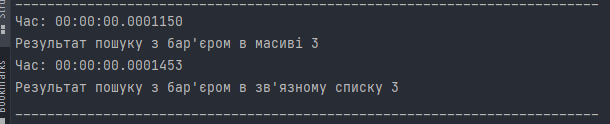


**Текст програми:**

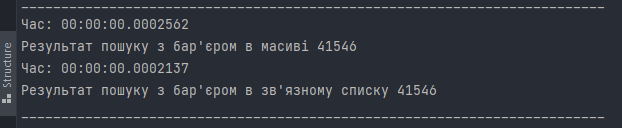
*//Пошук з бар'єром  
//Повертає індекс елемента, або значення, яке на 1 більше за розмір контейнера  
int*[] new\_arr = *new int*[size+1];  
*for* (*int* i = 0; i < size; i++)  
{  
 new\_arr[i] = arr[i];  
}  
new\_arr[size] = number;  
list.Add(number);  
*//\_\_\_\_Масив\_\_\_*interval.Start();  
index = Search.Barrier\_Search(new\_arr,number);  
interval.Stop();  
Console.WriteLine($"Час: {interval.Elapsed}");  
*if* (index>size)  
{  
 Console.WriteLine("Значення не знайдено");  
}  
*else*{  
 Console.WriteLine($"Результат пошуку з бар'єром в масивi {index}");  
}  
  
*//\_\_\_\_Ліст\_\_\_*interval.Start();  
index = Search.Barrier\_Search(list,number);  
interval.Stop();  
Console.WriteLine($"Час: {interval.Elapsed}");  
*if* (index>size)  
{  
 Console.WriteLine("Значення не знайдено");  
}  
*else*{  
 Console.WriteLine($"Результат пошуку з бар'єром в зв'язному списку {index}");  
}  
  
list.RemoveAt(list.Count-1);

**Набір тестів:**

На 4 елементах:

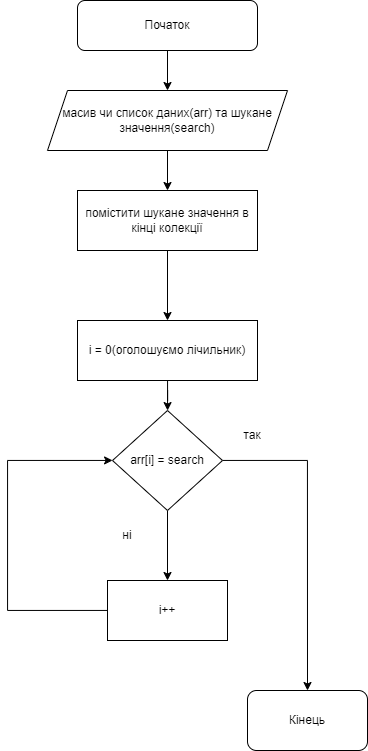


На 9999999 елементах:



**3. Бінарний пошук елемента масиву рівного заданому значенню.:**

**Алгоритм:**

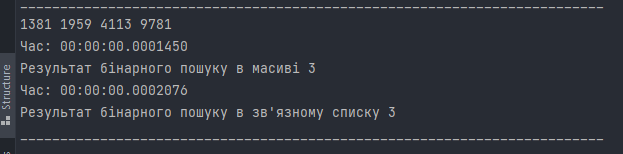


**Текст програми:**

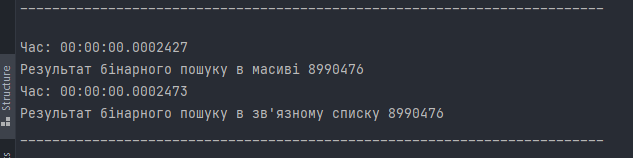
QuickSort.Sort(arr, 0,arr.Length-1);  
*for* (*int* i = 0; i < arr.Length; i++)  
{  
 Console.Write($"{arr[i]} ");  
}  
interval.Start();  
index = Search.Binary\_Search(arr,0,arr.Length-1,number);  
interval.Stop();  
Console.WriteLine($"\nЧас: {interval.Elapsed}");  
*if* (index == -1)  
{  
 Console.WriteLine("Значення не знайдено");  
}  
*else*{  
 Console.WriteLine($"Результат бiнарного пошуку в масивi {index}");  
}  
*//\_\_\_\_Ліст\_\_\_*list.Sort();  
interval.Start();  
index = Search.Binary\_Search(list,0,list.Count,number);  
interval.Stop();  
Console.WriteLine($"Час: {interval.Elapsed}");  
*if* (index == -1)  
{  
 Console.WriteLine("Значення не знайдено");  
}  
*else*{  
 Console.WriteLine($"Результат бiнарного пошуку в зв'язному списку {index}");  
}

**Набір тестів:**

На 4 елементах:

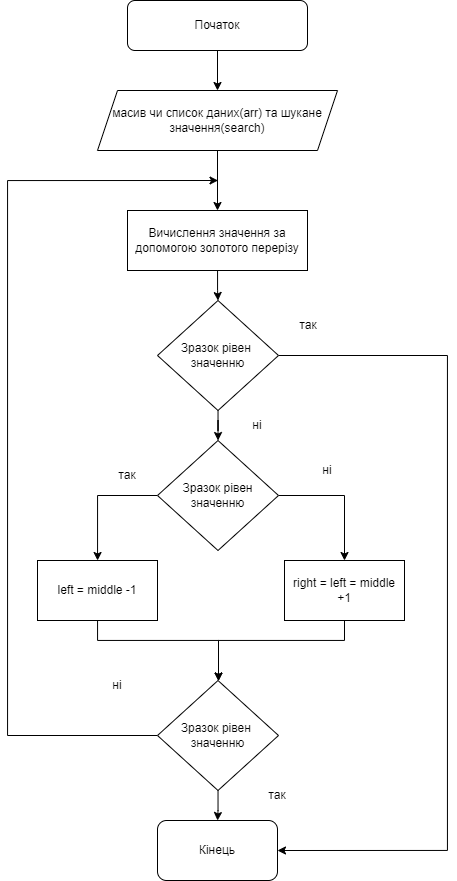


На 9999999 елементах:



**4. Бінарний пошук елемента масиву рівного заданому значенню на основі золотого перерізу:**

**Алгоритм:**

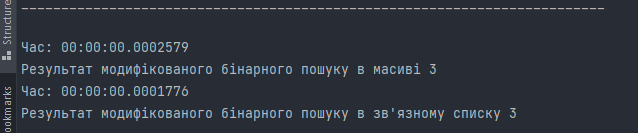


**Текст програми:**

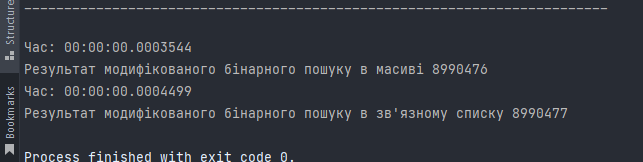
interval.Start();  
index = Search.Binary\_Search\_Modify(arr,0,arr.Length-1,number);  
interval.Stop();  
Console.WriteLine($"\nЧас: {interval.Elapsed}");  
*if* (index == -1)  
{  
 Console.WriteLine("Значення не знайдено");  
}  
*else*{  
 Console.WriteLine($"Результат модифiкованого бiнарного пошуку в масивi {index}");  
}  
*//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*list.Sort();  
interval.Start();  
index = Search.Binary\_Search\_Modify(list,0,list.Count,number);  
interval.Stop();  
Console.WriteLine($"Час: {interval.Elapsed}");  
*if* (index == -1)  
{  
 Console.WriteLine("Значення модифiкованого не знайдено");  
}  
*else*{  
 Console.WriteLine($"Результат модифiкованого бiнарного пошуку в зв'язному списку {index}");  
}

**Набір тестів:**

На 4 елементах:



На 9999999 елементах:

  
  
**Висновок:**Проаналізувавши роботу різних алогитмів на структурах «масив» та «зв’зний список» ми зробили висновок у роботі з ними:

* Зв’язний список подібен за використанням до масиву  
  але програє йому у швидкості
* Масив програє зв’язному списку при маніпуляціях з самим контейнером