Звіт про експеримент

студента КП - 62

Благодатного Дмитра Романовича

Лабораторна робота: “Емпіричний аналіз алгоритмів”

**Частина 1.**

1. Опис середовища:

|  |  |
| --- | --- |
|  | Комп’ютер |
| Апаратна частина | ACER ASPIRE V |
| Операційна система | Ubuntu 17.04 |
| Інтегроване середовище розробки | Eclipse 3.8 |
| Мова програмування | Java |

2. Псевдокод алгоритму counting sort та асимптотичний аналіз:

|  |
| --- |
| KEY\_INDEXING\_NONSTABLE\_COUNTING\_SORT (A[0...n-1])  Input: Масив A[0...n-1] цілих ключів в діапазоні від 0 до k  Output: Масив A [0...n-1] відсортований за зростанням  for i = 0 to k do  C[i] = 0  end for  for i = 0 to n – 1 do  C[A[i]] = C[A[i]] + 1  end for  i = 0  for j = 0 to k do  for l = 0 to C[j]  А [i] = j  i = i + 1  end for  end for |

Асимптотичний аналіз:

Cкладність по часу:

Найкраща - O(n)

Найгірша - O()

Cкладність по додатковій памяті - O(n + k)

Псевдокод алгоритму selection sort та його асимптотичний аналіз:

|  |
| --- |
| Selection sort(A,d)  for i = 1 to n-1 do  int smallest = i  for j = i + 1 to n do  if A[j] < A[smallest] smallest = j  A[i] <-> A[j] |

Асимптотичний аналіз:

Cкладність по часу:

Найкраща - O()

Середня - O()

Найгірша - O()

Cкладність по додатковій памяті:

Загальна - O(n)

Додаткова - O(1)

3. Код на мові Java:

|  |
| --- |
| **private** **static** **int**[] countSort(**int**[] read) {  **int**[] array = **new** **int**[read.length];  **int** min = read[0];  **int** max = read[0];  **for** (**int** i = 1; i < read.length; i++) {  **if** (read[i] < min) {  min = read[i];  } **else** **if** (read[i] > max) {  max = read[i];  }  }    **int**[] counts = **new** **int**[max - min + 1];    **for** (**int** i = 0; i < read.length; i++) {  counts[read[i] - min]++;  }    counts[0]--;  **for** (**int** i = 1; i < counts.length; i++) {  counts[i] = counts[i] + counts[i-1];  }    **for** (**int** i = read.length - 1; i >= 0; i--) {  array[counts[read[i] - min]--] = read[i];  }  **return** array;  } |

|  |
| --- |
| **private** **static** **int**[] insertionSort(**int**[] read) {  **int** temp = 0;  **for** (**int** i = 1; i < read.length; i++) {  **for** (**int** j = i ; j > 0 ; j--){  **if**(read[j] < read[j-1]){  temp = read[j];  read[j] = read[j-1];  read[j-1] = temp;  }  }  }  **return** read;  } |

4. Специфікація вхідного набору даних.

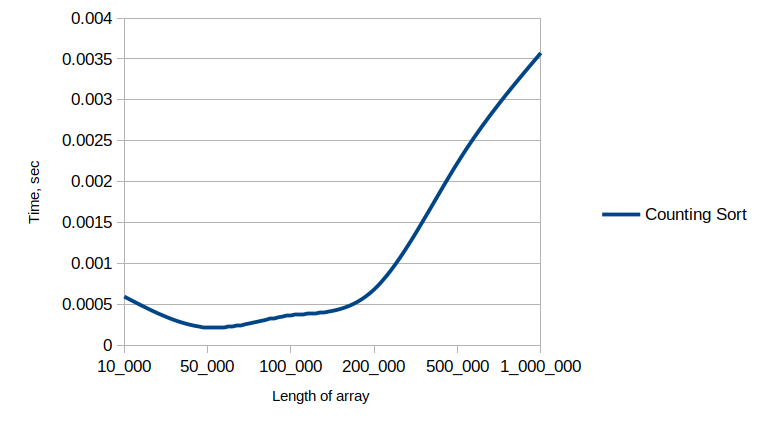
Вхідними даними слугує масив цілих чисел. Масив генерується за домогою методу makeRandom, класу RandomGen.

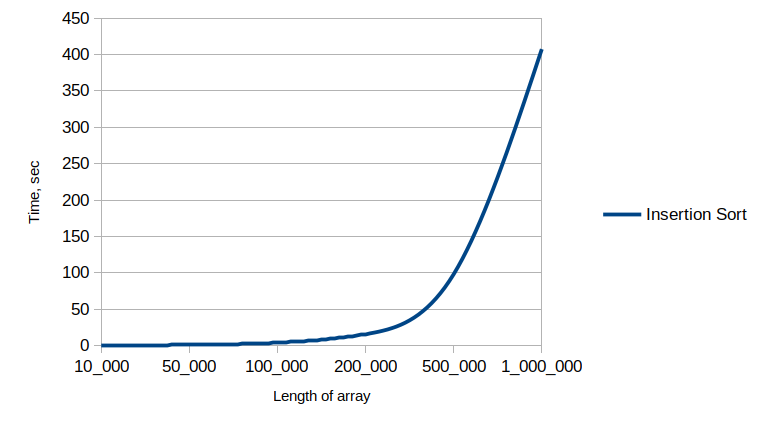
5. Таблиці з результатами вимірювання:

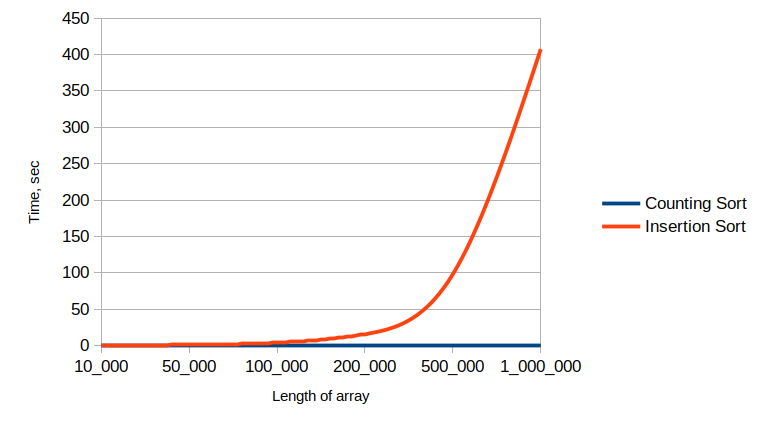
|  |  |  |
| --- | --- | --- |
| Counting Sort | | |
| # | Довжина масиву | Час, с |
| 1 | 10.000 | 0.000781409 |
| 2 | 20.000 | 0.00105828 |
| 3 | 40.000 | 0.00147194 |
| 4 | 80.000 | 0.000951859 |
| 5 | 160.000 | 0.00107322 |
| 6 | 320.000 | 0.00184097 |

|  |  |  |
| --- | --- | --- |
| Insertion Sort | | |
| # | Довжина масиву | Час, с |
| 1 | 10.000 | 0.0293210 |
| 2 | 20.000 | 0.0954155 |
| 3 | 40.000 | 0.391507 |
| 4 | 80.000 | 1.54927 |
| 5 | 160.000 | 6.33086 |
| 6 | 320.000 | 25.0398 |

6. Графіки







7. Висновок

В ході лабораторної роботи емпіричними методами було порівняно алгоритми counting sort та insertion sort. Порівняння виконувалось на апаратній платформі ACER ASPIRE V. В результаті отримані виміри часу роботи алгоритму при сортуванні масивів цілих чисел різної довжини. Алгоритм counting sort показав набагато кращий час на великих масивах, при сортуванні масивів не великої довжини, час роботи алгоритмів відрізнявся на невелику величину.