

Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

Кафедра автоматики та управління в технічних системах

Лабораторна робота №6

**Алгоритми та структури даних – 1. Основи алгоритмізації**

*«Алгоритми сортування. Bucket, counting, radix sort»*

Команда №6

|  |  |  |
| --- | --- | --- |
| Виконали  студенти групи ІТ-03: |  | Перевірив: |
|  |  |  |
| Чабан А.Є. |  |  |
| Філянін Н.С. |  |  |
| Хамад І.А. |  | ас. Цимбал С. І. |

Київ 2020

*Мета роботи:* навчитися сортувати масиви даних за допомогою різних

алгоритмів сортувань та дослідити вплив кількості елементів у масивах на

швидкість сортування.

*Хід роботи:*

1. Створити 3 масиви на 5к/50к/500к елементів, заповнені випадковими

числами, та 3 масиви на 5к/50к/500к елементів, заповнені послідовними числами

(наприклад, a[i]=i+1).

2. Клонувати масиви (C#/Java – Clone(), Python List copy()), щоб кожен

алгоритм сортування працював зі своїм масивом.

3. Почати замір часу.

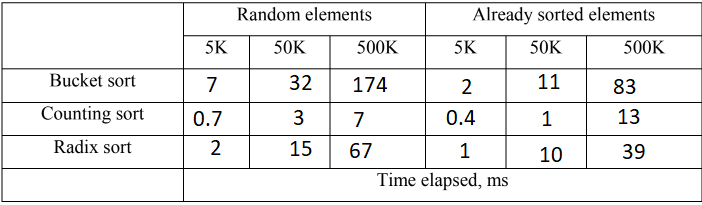
4. Виконати перший алгоритм сортування.

5. Зупинити таймер, вивести на екран затрачений на сортування час.

6. Повторити п. 2-5 для двох інших алгоритмів сортування.

7. Скласти порівняльну таблицю з замірами часу для всіх використаних

алгоритмів сортувань (табл. 6.1).



[\* Код лабки \*](https://github.com/it03team6/asdlabs/tree/master/asd6)

*Висновки:* Під час виконання данної лабораторної роботи наша команда вивчила нові алгоритми сортування та реалізувала їх на практиці.

Bucket sort (quick + insertion inside):

**package** ua.kpi.fict.acts.it03.asd6;

**import** **java.util.Collections**;

**import** **java.util.LinkedList**;

**import** **java.util.List**;

**final** **class** **BucketSort** {

**static** **void** **bucketSort**(**int**[] intArr, **int** noOfBuckets)

{

List<Integer>[] buckets = **new** List[noOfBuckets];

**for**(**int** i = **0**; i < noOfBuckets; i++)

{

buckets[i] = **new** LinkedList<>();

}

**for**(**int** num : intArr){

//System.out.println("hash- " + hash(num));

buckets[hash(num)].add(num);

}

**for**(List<Integer> bucket : buckets){

//InsertionSort(bucket);

//sort(bucket);

Collections.sort(bucket);

}

**int** i = **0**;

**for**(List<Integer> bucket : buckets){

**for**(**int** num : bucket){

intArr[i++] = num;

}

}

}

**private** **static** **int** **hash**(**int** num){

**return** num/**10**;

}

**private** **static** List<Integer> **InsertionSort**(List<Integer> list)

{

**for** (**int** j = **1**; j < list.size(); j++) {

Integer current = list.get(j);

**int** i = j-**1**;

**while** ((i > -**1**) && ((list.get(i).compareTo(current)) == **1**)) {

list.set(i+**1**, list.get(i));

i--;

}

list.set(i+**1**, current);

}

**return** list;

}

**public** **static** **void** **sort**(List<Integer> list) {

sort(list, **0**, list.size() - **1**);

}

**public** **static** **void** **sort**(List<Integer> list, **int** from, **int** to) {

**if** (from < to) {

**int** pivot = from + (to - from)/**2**;

**int** left = from + **1**;

**int** right = to;

**int** pivotValue = list.get(pivot);

**while** (left <= right) {

**while** (left <= to && pivotValue >= list.get(left)) {

left++;

}

**while** (right > from && pivotValue < list.get(right)) {

right--;

}

**if** (left < right) {

Collections.swap(list, left, right);

}

}

Collections.swap(list, pivot, left - **1**);

sort(list, from, right - **1**);

sort(list, right + **1**, to);

}

}

}

Counting Sort:

**package** ua.kpi.fict.acts.it03.asd6;

**final** **class** **CountingSort** {

**static** **void** **sort**(**int**[] array, **int** length)

{

**int** max = array[**0**];

**for** (**int** i = **1**; i < length; i++)

{

**if** (array[i] > max)

max = array[i];

}

**int**[] count = **new** **int**[max + **1**];

/\* for (int i = 0; i < max; ++i) {

count[i] = 0;

}\*/

**for** (**int** i = **0**; i < length; i++)

{

count[array[i]]++;

}

**for** (**int** i = **1**; i <= max; i++)

{

count[i] += count[i - **1**];

}

**int**[] output = **new** **int**[length + **1**];

**for** (**int** i = length - **1**; i >= **0**; i--)

{

output[count[array[i]] - **1**] = array[i];

count[array[i]]--;

}

**for** (**int** i = **0**; i < length; i++)

{

array[i] = output[i];

}

}

}

Radix Sort:

**package** ua.kpi.fict.acts.it03.asd6;

**final** **class** **RadixSort**

{

**static** **void** **radixSort**(**int** array[], **int** size) {

**int** max = getMax(array, size);

**for** (**int** place = **1**; max / place > **0**; place \*= **10**)

countingSort(array, size, place);

}

**static** **void** **countingSort**(**int**[] array, **int** size, **int** place) {

**int**[] output = **new** **int**[size + **1**];

**int** max = array[**0**];

**for** (**int** i = **1**; i < size; i++) {

**if** (array[i] > max)

max = array[i];

}

**int**[] count = **new** **int**[max + **1**];

**for** (**int** i = **0**; i < size; i++)

count[(array[i] / place) % **10**]++;

**for** (**int** i = **1**; i < **10**; i++)

count[i] += count[i - **1**];

**for** (**int** i = size - **1**; i >= **0**; i--) {

output[count[(array[i] / place) % **10**] - **1**] = array[i];

count[(array[i] / place) % **10**]--;

}

**for** (**int** i = **0**; i < size; i++)

array[i] = output[i];

}

**static** **int** **getMax**(**int**[] array, **int** n)

{

**int** max = array[**0**];

**for** (**int** i = **1**; i < n; i++)

**if** (array[i] > max)

max = array[i];

**return** max;

}

}