Comparing programming languages

Sorts

Insertion sort

Execution time of in-place insertion sort for different programming languages.
The function was executed on the array of 1000 integer numbers in range (0, 999).
The average time (in seconds) of 10 execution is shown.

Insertion sort	Arbitrary sequence	Sorted sequence	Sorted in reverse order
C	0.001140	0.000008	0.002255
C++	0.000002	0.000002	0.000003
Java	0.012230	0.000097	0.010782
Kotlin	0.008786	0.000094	0.026997
Python	0.071487	0.000277	0.149306

Source code for C/C++:

```
void insertionSort(int arr[], int len){
    for (int i = 1; i < len; i++){
        int temp = arr[i];
        int j = i - 1;
        while(j >= 0 & arr[j] > temp){
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = temp;
    }
}
```

Source code for Java:

```
    public static void insertionSort(int[] array) {
        for (int i = 1; i < array.length; i++) {
            int current = array[i];
            int j = i - 1;
            while (j >= 0 && current < array[j]) {
                array[j + 1] = array[j];
                j--;
            }
            array[j + 1] = current;
        }
    }
```

Source code for Kotlin:

```
fun insertionSort(arr: IntArray): Unit{
    for (i in 1 until arr.size){
        val current = arr[i]
        var j = i - 1
        while(j >= 0 && current < arr[j]){
            arr[j + 1] = arr[j]
            j--
        }
        arr[j + 1] = current
    }
}
```

Source code for Python:

```
def insertion_sort(arr: [int]):
    for i in range(1, len(arr)):
        current = arr[i]
        j = i - 1
        while(arr[j] > current and j >= 0):
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = current
```

Merge sort

A function merge sort was executed on the array of 1000 integer numbers in range (0, 999). The average time (in seconds) of 10 execution is shown. The algorithm requires additional O(n) space.

Merge sort	Arbitrary sequence	Sorted sequence	Sorted in reverse order sequence
C	0.000135	0.000138	0.000138
C++	0.000001	0.000001	0.000001
Java	0.005378	0.004801	0.003206
Kotlin	0.035176	0.036189	0.035762
Python	0.009343	0.007439	0.004438

Source code for C/C++:

```
void mergeSort(int arr[], int l, int r){
    if (l < r){
        int m = l + (r - l)/2;
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}
```

```
void merge(int arr[], int l, int m, int r){
    int n1 = m - l + 1;
    int n2 = r - m;
    int L[n1];
    int R[n2];
    for (int i = 0; i < n1; i++){
        L[i] = arr[i + l];
    }
    for (int j = 0; j < n2; j++){
        R[j] = arr[j + m + 1];
    }
    int i = 0;
    int j = 0;
    int k = l;
    while (i < n1 && j < n2){
        if (L[i] <= R[j]){
            arr[k] = L[i];
            i++;
        }
        else{
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    while(i < n1){
        arr[k] = L[i];
        i++;
        k++;
    }
    while(j < n2){
        arr[k] = R[j];
        j++;
        k++;
    }
}
```

Source code for Java:

```
    public void mergeSort(int[] array, int left, int right) {
        if (right <= left) return;
        int mid = (left+right)/2;
        mergeSort(array, left, mid);
        mergeSort(array, mid+1, right);
        merge(array, left, mid, right);
    }

    void merge(int[] array, int left, int mid, int right) {
        int lengthLeft = mid - left + 1;
        int lengthRight = right - mid;

        int leftArray[] = new int [lengthLeft];
        int rightArray[] = new int [lengthRight];

        for (int i = 0; i < lengthLeft; i++)
```

```
            leftArray[i] = array[left+i];
        for (int i = 0; i < lengthRight; i++)
            rightArray[i] = array[mid+i+1];

        int leftIndex = 0;
        int rightIndex = 0;

        for (int i = left; i < right + 1; i++) {
            if (leftIndex < lengthLeft && rightIndex < lengthRight) {
                if (leftArray[leftIndex] < rightArray[rightIndex]) {
                    array[i] = leftArray[leftIndex];
                    leftIndex++;
                }
                else {
                    array[i] = rightArray[rightIndex];
                    rightIndex++;
                }
            }
            else if (leftIndex < lengthLeft) {
                array[i] = leftArray[leftIndex];
                leftIndex++;
            }
            else if (rightIndex < lengthRight) {
                array[i] = rightArray[rightIndex];
                rightIndex++;
            }
        }
    }
```

Source code for Kotlin:

```
fun mergeSort(arr: IntArray, l: Int, r: Int){
    if (l < r){
        var m: Int = l + (r - l)/2
        mergeSort(arr, l, m)
        mergeSort(arr, m + 1, r)
        merge(arr, l, m, r)
    }
}

fun merge(arr: IntArray, l: Int, m: Int, r: Int): Unit{
    val n1 = m - l + 1
    val n2 = r - m
    val L = IntArray(n1)
    val R = IntArray(n2)
    for (i in 0 until n1){
        L[i] = arr[i + l]
    }
    for (j in 0 until n2){
        R[j] = arr[j + m + 1];
    }
    var i = 0
    var j = 0
    var k = l
    while (i < n1 && j < n2){
        if (L[i] <= R[j]){
            arr[k] = L[i]
            i++
        }
        else{
            arr[k] = R[j]
            j++
        }
        k++
    }
    while(i < n1){
        arr[k] = L[i]
        i++
        k++
    }
    while(j < n2){
        arr[k] = R[j]
        j++
        k++
    }
}
```

Source code for Python:

```
def merge_sort(arr: [int], l: int, r: int):
    if l < r:
        m = (l + r)//2
        merge_sort(arr, l, m)
        merge_sort(arr, m + 1, r)
        merge(arr, l, m, r)

def merge(arr: [int], l: int, m: int, r: int):
    n1 = m - l + 1
    n2 = r - m

    L = [0] * n1
    R = [0] * n2
    for i in range(0, n1):
        L[i] = arr[i + l]

    for j in range(0, n2):
        R[j] = arr[j + m + 1]
    i = 0
    j = 0
    k = l
    while i < n1 and j < n2:
        if L[i] < R[j]:
            arr[k] = L[i]
            i += 1
        else:
            arr[k] = R[j]
            j += 1
        k += 1

    while i < n1:
        arr[k] = L[i]
        i += 1
        k += 1
    while j < n2:
        arr[k] = R[j]
        j += 1
        k += 1


def counting_sort(arr: [int], max: int):
    c = [0] * max
    for i in range(0, len(arr)):
        c[arr[i]] += 1
    k = 0
    for i in range(0, max):
        while (c[i] > 0):
            arr[k] = i;
            k += 1
            c[i] -= 1
```

Counting sort

A function was executed on the array of 1000 integer numbers in range (0, 999).
The time shown is the average time of 10 executions. The algorithm requires additional O(range) space to store the number of occurrences of each element.

Counting sort	Arbitrary sequence	Sorted sequence	Sorted in reverse order
C	0.000006	0.000005	0.000009
C++	0.000012	0.000016	0.000015
Java	0.000283	0.000276	0.000293
Kotlin	0.000198	0.000118	0.000196
Python	0.000736	0.000717	0.000457

Source code for C/C++:

```
void countingSort(int arr[], int len){
    int count[RANGE];
    memset(count, 0, sizeof(count));
    for (int i = 0; i < len; i++){
        count[arr[i]]++;
    }
    int k = 0;
    for (int i = 0; i < RANGE; i++){
        while(count[i] > 0){
            arr[k] = i;
            k++;
            count[i]--;
        }
    }
}
```

Source code for Java:

```
    public void countingSort(int[] arr, int range) {
        int[] count = new int[range];
        Arrays.fill(count, 0);
        for (int i : arr) {
            count[i] += 1;
        }

        int k = 0;
        for (int i = 0; i < range; i++){
            while(count[i] > 0){
                arr[k] = i;
                k++;
                count[i]--;
            }
        }
    }
```

Source code for Kotlin:

```
fun countingSort(arr: IntArray, range: Int = 1000): Unit{
    val count = IntArray(range)
    for (element in arr){
        count[element]++
    }
    var k = 0
    for (i in 0 until range){
        while (count[i] > 0){
            arr[k] = i
            count[i]--
            k++
        }
    }
}
```

Source code for Python:

```
def counting_sort(arr: [int], max: int):
    c = [0] * max
    for i in range(0, len(arr)):
        c[arr[i]] += 1
    k = 0
    for i in range(0, max):
        while (c[i] > 0):
            arr[k] = i;
            k += 1
            c[i] -= 1
```