Bucket sort

Student : Hussin Almoustafa

Studentnummer : 1776495

Bucket sort Class :

```
class Bucket
{
    private:
    int bucket_entries;
    int *   entry;
    public:
    Bucket(int entries) : bucket_entries(0) {
        int index;
        entry = new int[entries];
        for ( index = 0; index < entries; entry[index] = 0, index++ );
    }
    void add_entry(int entry_value) { entry[bucket_entries++] = entry_valu
    int bucket_count() { return bucket_entries; };
    int bucket_entry(int index) { return entry[index]; };
};

class BucketSort
{
    private:
    void internalBucketSort(int * array, int entries, int digit, int place
    public:
        void bucketSort(int * array, int entries);

};
```

Hierbij wordt class Bucket sort gemaakt.

De algoritme werkt als volgt :

1. Bepaal de lengte van de array
2. Maak op basis van dat een lege arraymet n aantal lege lists
3. Loop over de array met deze functie (elemet % 10 ** digits)
4. Append de getallen in de buckets
5. Append de gesorteerde lijst in een een nieuwe lijst
6. Return de lijst

```
void BucketSort::internalBucketSort(int * array, int entries, int digit, int places) {
    int array_index, digit_index, bucket_index, gather_index;
    Bucket ** bucket = new Bucket *[10];
    int max_value = 0;

    // initialisatie van buckets
    for ( bucket_index = 0; bucket_index < 10; bucket_index++ ) {
    bucket[bucket_index] = new Bucket(entries);
    }

    // distributiepas
    for ( array_index = 0; array_index < entries; array_index++ ) {
    int array_value = array[array_index];
    if ( array_value > max_value ) max_value = array_value;
    for ( digit_index = 0; digit_index < digit; array_value/=10, digit_index++);
    bucket[array_value%10]->add_entry(array[array_index]);
    }

    // gathering pass
    array_index = 0;
    for ( bucket_index = 0; bucket_index < 10; bucket_index++ ) {
    int bucket_entries = bucket[bucket_index]->bucket_count();
    for ( gather_index = 0; gather_index < bucket_entries; gather_index++ ) {
        array[array_index++] = bucket[bucket_index]->bucket_entry(gather_index);
    }
    }
    if ( max_value > places ) internalBucketSort(array, entries, digit+1, places*10);
}

void BucketSort::bucketSort(int * array, int arraySize) {
    internalBucketSort(array, arraySize, 0, 10);
}
```

Tijd en ruimtecomplexitie van het algoritme

Best case:

De beste case hangt van veel factoren af, denk hierbij aan de aantal digits van de langste getal, de lengte van de lijst en of de lijst al gesorteerd is.

Als we een blangrijkste factor moeten uit kiezen dat zou dat de aantal digits van de langste getal. Dat komt omdat er wanneer de langste getal uit 1 dingit bestaat dat de distribution step 1 keer uitgevoerd moet worden, geldt ook voor de gathering step.

In dit geval zouden we de best case als volgt beschrijven:

$$O(n+ l + k)$$

Wat altijd een rol speelt bij het sorteren van een algoritme is de lengte van de array, oftwel de (n) hoe hoger de (n) is hoe langer het duurt voordat een array gesorteerd is. Hiervoor kan je naar de runtime analyse kijken.

De ruimtecomplexiteit is in het algemeen iets minder belangrijk. Maar we kunnen als volgt samenvatten:

Als we n aantaal getallen hebben, dan zou k aantaal bukets opnieuw gevuld moeten worden met n getallen.

Oftewel:

$$O(n.k)$$

Worse Case :

$$O((2n) + (n) + l(n) + 2(k) + (2n))$$

$$O(5(n) + l(n) + (k)$$

$$O(5(n) + l(n) + (k))$$