Relatório 07 - Baldes

SCC0220 - Laboratório de Introdução à Ciência da Computação II (2022)

Prof. Diego Furtado Silva

Departamento de Ciências da Computação (SCC)

Instituto de Ciências Matemáticas e de Computação (ICMC)

Universidade de São Paulo

Discentes:

- Felipi Yuri Santos (*11917292*)

Sumário

I - Implementações do BucketSort() e do MergeSort()

i. BucketSort()

```
void bucketSort(float *arr, int n, int m, int b) {
    vector<float> buck[n];

    float div = (float) m / b;
    for (int i = 0; i < n; i++) {
        int bi;
        if (arr[i] != m){
            bi = arr[i] / div;
            buck[bi].push_back(arr[i]);
        }
        else
            buck[b - 1].push_back(arr[i]);
    }

    for (int i = 0; i < b; i++)
        sort(buck[i].begin(), buck[i].end());

    int index = 0;
    for (int i = 0; i < n; i++)
```

```
        for (int j = 0; j < (int) buck[i].size(); j++)
            arr[index++] = buck[i][j];

}
```

ii. MergeSort()

```
void mergeSort(int* vector, int start, int end){
  if (end <= start)
    return;

  int center = (int)((start + end) / 2.0);
  mergeSort(vector, start, center);
  mergeSort(vector, center+1, end);

  intercalate(vector, start, center, end);
}

void intercalate(int *vector, int start, int center, int end){
  int* vectorAux = (int*) malloc(sizeof(int) * (end - start) + 1);

  int i = start;
  int j = center + 1;
  int k = 0;

  while(i <= center && j <= end){
    if (vector[i] <= vector[j]){
      vectorAux[k] = vector[i];
      i++;
    }
    else{
      vectorAux[k] = vector[j];
      j++;
    }
    k++;
  }

  while(i <= center){
    vectorAux[k] = vector[i];
    i++;
    k++;
  }

  while(j <= end){
    vectorAux[k] = vector[j];
    j++;
    k++;
  }

  for(i = start, k = 0; i <= end; i++,k++)
    vector[i] = vectorAux[k];
  free(vectorAux);
}
```

III - Comparação dos tempos de execução

	Caso 1 (ms)	Caso 2 (ms)	Caso 3 (ms)	Caso 4 (ms)	Caso 5 (ms)	Caso 6 (ms)	Caso 7 (ms)
BucketSort()	1	3	6	6	59	1	2
MergeSort()	1	1	4	6	36	1	1

Como podemos observar, há dois casos onde pesa-se muito o custo de processamento. O 5 e o 10, mas é claro: são os casos-teste com as maiores entradas. Conseguimos ver uma pequena vantagem do mergesort() nesses casos. Ou seja, por via apenas de comparações, conseguimos ver que o

BucketSort() foi mais custoso que o MergeSort(), que tem complexidade $O(nlog(n))$.

É um pouco injusto talvez a comparação em certo sentido porque foi utilizada uma linguagem diferente, de paradigma diferente que usa um compilador diferente para o BucketSort(), que tem complexidade temporal de $O(n + k)$, onde n é a quantidade de elementos a ser ordenada e k é o número de baldes.