ANÁLISE DA EFICIÊNCIA DO ALGORITMO DE MERGE SORT

ISABELLA BASSO DO AMARAL — 11810773

1. Introdução

Esta é uma análise de eficiência simplificada, utilizando o método de *doubling*[1], conforme fora estudado na disciplina CCM0128, do curso de Ciências Moleculares da USP.

2. Método

2.1. ***doubling.*** Na presente análise utilizaremos o método de *doubling*, o qual nos dá uma estimativa da complexidade do algoritmo estudado através do processo de dobrar a quantidade de entradas a cada iteração, de tal forma que facilita a visualização da taxa de crescimento da complexidade algorítmica do que estamos analisando. I.e. para inputs sucessivos de $1, 2$ e 4, para uma algoritmo com complexidade de ordem $\mathcal{O}(n)$ esperamos um resultado com tempos $\alpha, 2\alpha$ e 4α, respectivamente, enquanto que para um algoritmo com complexidade de ordem $\mathcal{O}(n^2)$ esperamos tempos de $\alpha, 4\alpha$ e 16α.

2.2. **Medidas.** Para que não haja *overhead* da função de geração (veja 5.2) nos tempos medidos, criaremos os arquivos a serem organizados de antemão.

Em cada passo do *doubling* realizaremos 100 medições, tirando sua média ao final (veja 5.4).

3. Resultados

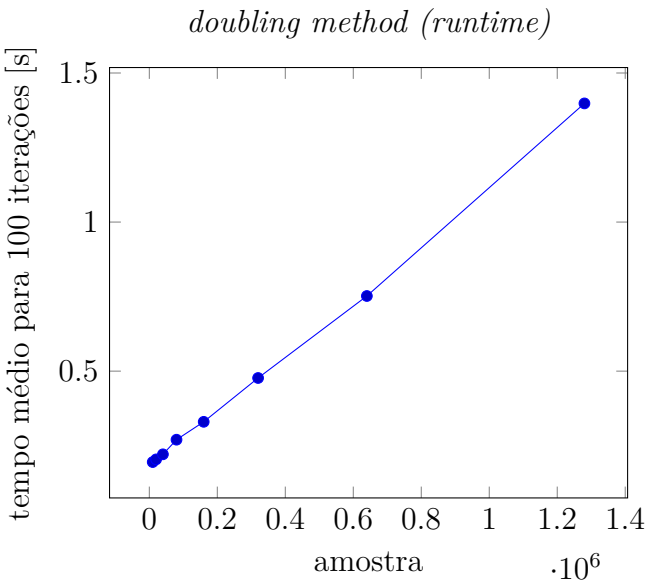
Podemos ver que o script gerou os seguintes dados no arquivo `.csv`.

isabellabdoamaral@usp.br.

[1]Aviso: A implementação do algoritmo, gerador e scripts utilizados para análise de dados foram feitos pela autora e não serão discutidos em profundidade no presente trabalho.

TABELA 1. *runtime* do `merge sort`

sample size	avg time
10 000	0.195
20 000	0.204
40 000	0.221
80 000	0.270
160 000	0.330
320 000	0.477
640 000	0.752
1 280 000	1.398



O que se aproxima de uma reta com coeficiente angular aproximado

$$\alpha = \frac{1.398 - 0.195}{(1.28 - 0.01) \cdot 10^6} \approx 0.95\,\text{ns/sample}$$

4. DISCUSSÃO

4.1. **Complexidade.** Dado que o `runtime` do algoritmo aproxima uma reta, podemos inferir que sua complexidade é $\mathcal{O}(n)$.

4.2. **Estabilidade.** Observando a função `merge` (linhas 19–44 do arquivo 5.1), nota-se que, enquanto o índice `k` aumenta, o índice `i` que acompanha o array inferior é comparado ao índice `j` que acompanha o array superior e, dessa forma, para garantir estabilidade é necessário que preservemos essa relação.

No presente trecho (retirado do mesmo arquivo) notamos que, conforme desejado, o array inferior é favorecido em casos de igualdade e, conforme iteramos com o índice `k` aumentando, a implementação é estável.

```
37              if (lower[i].compareTo(upper[j]) <= 0)
38                  merger[k] = lower[i++];
39              else
40                  merger[k] = upper[j++];
```

5.1. **Apêndice A.** Arquivo de `merge sort` (em Java, reescrito por simplicidade).

```
1  public class minimalMergeSort {
2      public static void main(String[] args) {
3          String[] list = mergeSort(StdIn.readAllStrings());
4      }
5
6      private static String[] mergeSort(String[] list) {
7          if (list.length == 1) return list;
8
9          int upper_bound = list.length - 1;
10         int lower_bound = 0;
11         int middle = (upper_bound - lower_bound)/2;
12
13         return merge(
14                 mergeSort(slice(list, lower_bound, middle)),
15                 mergeSort(slice(list, middle + 1, upper_bound))
16                 );
17     }
18
19     private static String[] merge(String[] lower, String[] upper)
20     {
21         int i = 0, j = 0, k = 0;
22         String[] merger = new String[lower.length + upper.length
               ];
23
24         while (k < merger.length)
25         {
26             if (i == lower.length) {
27                 for (; j < upper.length; j++)
28                     merger[k++] = upper[j];
29                 break;
30             }
31             else if (j == upper.length) {
32                 for (; i < lower.length; i++)
```

```
                    merger[k++] = lower[i];
                break;
            }

            if (lower[i].compareTo(upper[j]) <= 0)
                merger[k] = lower[i++];
            else
                merger[k] = upper[j++];

            k++;
        }
        return merger;
    }

    private static String[] slice(String[] list, int s, int e) {
        String[] n_list = new String[e - s + 1];
        for (int i = s; i <= e; i++)
             n_list[i - s] = list[i];
        return n_list;
    }
}
```

5.2. **Apêndice B.** Arquivo gerador de strings aleatórias (em Python).

```
#!/usr/bin/python
def main():
    import sys
    num = int(sys.argv[1])
    length = int(sys.argv[3]) if len(sys.argv) > 3 else 10
    with open(sys.argv[2], 'w') as f:
        for word in get_random(num, length):
            f.write(word + '\n')

def get_random(num, length):
    import random
    lower_alphabet = [chr(65 + 32 + i) for i in range(26)]

    for i in range(num):
        word = ''
        for ci in range(length):
            word += lower_alphabet[random.randint(0, 25)]

        yield word

main()
```

5.3. **Apêndice C.** Arquivo de média (em Python).

```
#!/usr/bin/python
def main():
    import sys
    #with open(sys.argv[1]) as f:
    first = True
    s = last_val = count = 0
    #for line in f.readlines():
    for line in sys.argv[1:]:
        val, time = line.split(',')
        time = float(time)
        if first:
            first = False
            last_val = val
        if last_val == val:
            s += time
            count += 1
            continue
        elif count > 0:
            result = s/count
            sys.stdout.write("{0},{1:.3f}\n".format(last_val,
               result))
            last_val = val
            s = count = 0
    if count > 0:
        result = s/count
        sys.stdout.write("{0},{1:.3f}\n".format(last_val, result)
           )

main()
```

5.4. **Apêndice D.** Script de *timing*.

```
#!/bin/sh

java_file="minimalMergeSort"

javac-introcs "$java_file.java"

current_iter=$1
samples=$3
rand_str=$(mktemp)

gnu_time_path=$(which time)
final_results="results.csv"
echo 'sample size,avg time' > $final_results
for i in $(seq $2)
do
    current_avg=$(\
    for j in $(seq $samples); do
        ./str_gen $current_iter $rand_str
        output=$({
            $gnu_time_path -f'%e' java-introcs $java_file <
               $rand_str
            } 2>&1 >/dev/null)
        echo "$current_iter,$output"
        rm $rand_str
    done )
    ./get_averaged_table $current_avg >> $final_results
    current_iter=$((2 * $current_iter))
done
```