Analysis and Design of Algorithms

Jorge Rios

April 2019

1 Warm up

Lets modify the classic merge sort algorithm a little bit. What happens if instead of splitting the array in 2 parts we divide it in 3? You can assume that exists a three-way merge subroutine. What is the overall asymptotic running time of this algorithm?

BONUS: Implement the three-way merge sort algorithm.

1.1 Solution: $\Theta(nlg_3n)$

1.1.1 Divide-and-conquer approach

I used divide-and-conquer's approach to solve the problem.

If $T(n)$ is the running time of a problem with n, in this problem, the size of the collection (e.g array). As is defined on Cormen's book explain, the problem generate a subproblems, where each of which is $1/b$ the size of the previous problem. The algorithm will take $T(n/b)$ to solve one of the subproblems of size n/b, so, it takes $aT(n/b)$ to solve a of them. Futhermore, the algorithm use $D(n)$ time to divide the problem into subproblems and $C(n)$ time to combine the solutions to the subproblems into the solution to the original problem, we get the recurrence equation:

$$T(n) = aT(n/b) + D(n) + C(n)$$

The problem of split your collection in a 3-way, is explained using the chosen approach:

Divide: It define 2 "middles" based on the size of the collection to break in 3 parts. It will take constant time, this means that $D(n) = \Theta(1)$.

Conquer: It recursively solve 3 subproblems, each of the size $n/3$. Thus, $aT(n/b) = 3T(n/3)$.

Combine: If it assumes that a 3-way subroutine exists with $\Theta(n)$ runtime. Thus, $C(n) = \Theta(n)$.

So, we get the recurrence:

$$T(n) = 3T(n/3) + \Theta(1) + \Theta(n)$$

It could use the recurrence tree intuitive steps below to get the result of $T(n/3)$ run time.

1.1.2 Recurrence Tree

The recurrence tree is showed below, splitting the collection in a 3-way.

The algorithm progressively expands the 3-way tree, it finish when it get a 1-size array, or 1 element.

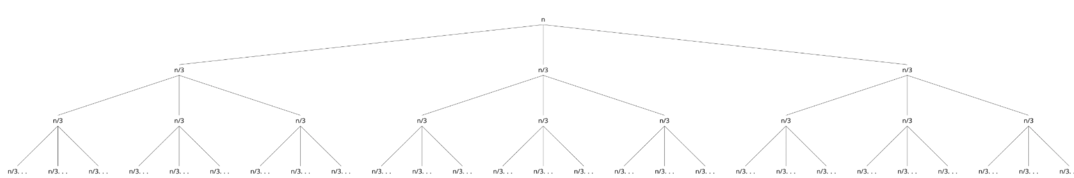


Figure 1: 3-way tree

This tree has $lg_3 + 1n$ levels. Then, it has a height of $lg_3 n$, where each level get a cost of n. Thus, the total cost is $nlg_3 n + n$, which means $\Theta(nlg_3 n)$.

2 Competitive programming

Welcome to your first competitive programming problem!!!

- Sign-up in Uva Online Judge (`https://uva.onlinejudge.org`) and in CodeChef if you want (we will use it later).
- Rest easy! This is not a contest, it is just an introductory problem. Your first problem is located in the "Problems Section" and is **100 - The 3n + 1 problem**.
- Once that you finish with that problem continue with **458 - The Decoder**. Again, this problem is just to build your confidence in competitive programming.
- *BONUS:* **10855 - Rotated squares**

2.1 100 - The 3n + 1 problem

Listing 1: "The 3n+1 problem source code"

```
#include <iostream>

unsigned int stop(int n) {
  unsigned int saltos = 0;

  while (n != 1) {
    if (n % 2 == 1) {
      n = 3 * n + 1;
      n = n / 2;
      saltos++;
    } else {
      n = n / 2;
    }

    saltos++;
  }

  return ++saltos;
}

int main() {
  int a, b;
```

```
    int aToPrint = 0;
    int tmp = 0;
    while (std::cin >> a >> b) {
        aToPrint = a;
        tmp = b;
        int max = 0;
        unsigned int result = 0;

        if (a > b) {
            b = a;
            a = tmp;
        }

        for (int j = a; j <= b; ++j) {
            result = stop(j);
            if (result > max) {
                max = result;
            }
        }

        std::cout << aToPrint << " " << tmp << " " << max << std::endl;
    }
    return 0;
}
```

2.1.1 Proof of submission acceptance

Attached in Appendice 2.

2.2 458 - The Decoder

Listing 2: "The Decoder problem source code"

```
#include <iostream>

int main() {
    std::string line;
    while (getline(std::cin, line)) {
```

```
    for (auto &l : line) {
      std::cout << char(l - 7);
    }

    std::cout << std::endl;
  }

  return 0;
}
```

2.2.1 Proof of submission acceptance

Attached in Appendice 3.

3 Simulation

Write a program to find the minimum input size for which the merge sort algorithm always beats the insertion sort.

- Implement the insertion sort algorithm
- Implement the merge sort algorithm
- Just compare them? No !!! Run some simulations or tests and find the average input size for which the merge sort is an asymptotically “better” sorting algorithm.

Note: Include (.tex) and attach(.cpp) your source code and use a dockerfile to interact with python and plot your results.

BONUS: Compare both algorithms against any other sorting algorithm

3.1 Insertion sort algorithm

Listing 3: ”Insertion sort algorithm implementation”

```
void insertion() {
  for (int i = 1; i < this->collection.size(); i++) {
```

```
        int j = i;
        AnyObject key = this->collection[i];

        while (j > 0 && this->collection[j - 1] > key) {
            this->collection[j] = this->collection[j - 1];
            j--;
        }

        this->collection[j] = key;
    }
}
```

3.2 Merge sort algorithm

Listing 4: "Merge sort algorithm implementation"

```
void mergeCollection(std::vector<AnyObject> &collection, int p, int q, int r) {
    int n1 = q - p + 1;
    int n2 = r - q;

    std::vector<AnyObject> L(n1 + 1);
    std::vector<AnyObject> R(n2 + 1);

    for (int i = 0; i < n1; ++i) {
        L[i] = collection[p + i];
    }

    for (int j = 0; j < n2; ++j) {
        R[j] = collection[q + j + 1];
    }

    L[n1] = std::numeric_limits<AnyObject>::max();
    R[n2] = std::numeric_limits<AnyObject>::max();

    int i = 0;
    int j = 0;

```

```
    for (int k = p; k <= r; k++) {
      if (L[i] <= R[j]) {
        collection[k] = L[i];
        i++;
      } else {
        collection[k] = R[j];
        j++;
      }
    }
  }

  void merge(std::vector<AnyObject> &collection, int p, int r) {
    if (p < r) {
      int q = std::floor((p + r) / 2.0);

      merge(collection, p, q);
      merge(collection, q + 1, r);

      mergeCollection(collection, p, q, r);
    }
  }
```

3.3 Insert and Merge Sort comparison

Merge Sort is better from **270 onwards**, on average. For more information, see Appendice A, attached in this paper.

3.3.1 Simulation

Listing 5: "Simulation was made by C++"

```
#include <iostream>
#include <vector>
#include <cmath>
#include <limits>
#include <time.h>
#include <iomanip>
#include <algorithm>
```

```
#include <random>
#include <fstream>

#define FILE_PATHNAME "plotter/input/data.txt"
#define STEP 10
#define MAX 800
#define MAX_RANDOM 20000

template<class AnyObject>
class Sort {
 private:
  std::vector<AnyObject> collection;
 public:
  Sort(std::vector<AnyObject> collection) {
    this->collection = collection;
  }

  std::vector<AnyObject> &getCollection() {
    return collection;
  }

  void setCollection(const std::vector<AnyObject> &collection) {
    Sort::collection = collection;
  }

  void insertion() {
    for (int i = 1; i < this->collection.size(); i++) {
      int j = i;
      AnyObject key = this->collection[i];

      while (j > 0 && this->collection[j - 1] > key) {
        this->collection[j] = this->collection[j - 1];
        j--;
      }

      this->collection[j] = key;
    }
  }
```

```
void mergeCollection(std::vector<AnyObject> &collection, int p, int q, int r) {
  int n1 = q - p + 1;
  int n2 = r - q;

  std::vector<AnyObject> L(n1 + 1);
  std::vector<AnyObject> R(n2 + 1);

  for (int i = 0; i < n1; ++i) {
    L[i] = collection[p + i];
  }

  for (int j = 0; j < n2; ++j) {
    R[j] = collection[q + j + 1];
  }

  L[n1] = std::numeric_limits<AnyObject>::max();
  R[n2] = std::numeric_limits<AnyObject>::max();

  int i = 0;
  int j = 0;

  for (int k = p; k <= r; k++) {
    if (L[i] <= R[j]) {
      collection[k] = L[i];
      i++;
    } else {
      collection[k] = R[j];
      j++;
    }
  }
}

void merge(std::vector<AnyObject> &collection, int p, int r) {
  if (p < r) {
    int q = std::floor((p + r) / 2.0);

    merge(collection, p, q);
```

```
      merge(collection, q + 1, r);

      mergeCollection(collection, p, q, r);
    }
  }

  void describe() {
    for (auto &element : collection) {
      std::cout << element << " ";
    }

    std::cout << std::endl;
  }
};

void writeFile(std::vector<std::pair<float, float>> measures) {
  std::fstream file(FILE_PATHNAME, std::ios::out);

  if (!file.is_open()) {
    std::cout << "Error al leer el archivo" << std::endl;
  }

  for (auto &x : measures) {
    file << std::fixed << x.first << " " << x.second << std::endl;
  }

  file.close();
}

int main() {
  std::vector<std::pair<float, float>> measures;
  std::pair<float, float> measure;

  for (int i = 1; i < (MAX / STEP) + 1; i++) {
    std::vector<int> collection(i * STEP);

    std::generate(collection.begin() + i - 1, collection.end(), []() {
      return rand() % MAX_RANDOM;
```

```
    });

    auto sort = new Sort<int>(collection);

    // Insertion Sort
    auto t0 = clock();
    sort->insertion();
    auto t1 = clock();
    auto insertionTime = (float) (t1 - t0) / CLOCKS_PER_SEC;

    // Merge sort
    sort->setCollection(collection);
    t0 = clock();
    sort->merge(sort->getCollection(), 0, collection.size() - 1);
    t1 = clock();
    auto mergeTime = (float) (t1 - t0) / CLOCKS_PER_SEC;

    measure.first = insertionTime;
    measure.second = mergeTime;
    measures.push_back(measure);
  }

  writeFile(measures);

  return 0;
}
```

3.3.2 Plotting

Listing 6: "Plotter was coded on Python"

```
import matplotlib
import matplotlib.pyplot as plt
import sys
import os

# Pathname
dir_path = os.path.dirname(os.path.realpath(__file__))
```

```

# Config variables
step = 10
maximum = 800
height = 600
width = 1800
measuresData = range(1, maximum + 1, step)

if (len(sys.argv) == 1):
    print("You need provide pathname: (*.txt)")
    exit()

try:
    pathname = dir_path + sys.argv[1]
    f = open(pathname, "r")
except OSError:
    print(dir_path)
    print(sys.argv[1:])
    print("Error opening the file, verify and try again, please.")
    exit()

# Read data from file
mergeMeasures = []
insertionMeasures = []
lines = f.readlines()
for line in lines:
    measures = line.split(" ")
    insertionMeasures.append(float(measures[0]))
    mergeMeasures.append(float(measures[1]))

# Plot merge measures
plt.plot(measuresData, mergeMeasures, linewidth=4.0, label='Merge')

# Plot insertion measures
plt.plot(measuresData, insertionMeasures, linewidth=2.0, label='Insertion', linestyle='dashed')

# Custom graph
plt.ylabel('seconds')
```

```
plt.xlabel('collection\'s length')
plt.legend()

# Save graph
fig = plt.gcf()
DPI = fig.get_dpi()
fig.set_size_inches(width/float(DPI), height/float(DPI))
plt.savefig(dir_path + '/output/insertion_vs_merge_sort.png')

# End
print("Grafico generado correctamente")
exit()
```

3.3.3 Dockerfile

Listing 7: "Docker to execute plotting script"

```
FROM python:3.7-alpine
RUN apk add --no-cache bash
RUN apk add g++ freetype-dev
RUN pip3 --no-cache-dir install --upgrade pip
RUN pip3 --no-cache-dir install -U setuptools
RUN pip3 --no-cache-dir install matplotlib
WORKDIR /usr/src/app
COPY / ./
ENTRYPOINT ["python", "plotter/plotter.py"]
```

4 Research

Everybody at this point remembers the quadratic "grade school" algorithm to multiply 2 numbers of k_1 and k_2 digits respectively.

Your assignment now is to compare the number of operations performed by the quadratic grade school algorithm and Karatsuba multiplication.

- Define Karatsuba multiplication

- Implement grade school multiplication
- Implement Karatsuba multiplication
- Compare Karatsuba algorithm against grade school multiplication
- Use any of your implemented algorithms to multiply $a * b$ where:

  a: 3141592653589793238462643383279502884197169399375105820974944592

  b: 2718281828459045235360287471352662497757247093699959574966967627

Note: Include(.tex) and attach(.cpp) your source code, of course.

BONUS: How about Schönhage-Strassen algorithm ?

4.1 Define Karatsuba multiplication

The Karatsuba (or Karatsuba-Ofman) algorithm was published in 1962 by Anatolii Alexeevitch Karatsuba and Yuri Petróvich Ofman, through their paper *Multiplication of many-digital numbers by automatic computers.*

The algorithm is defined as a recursive algorithm that shall multiply large numbers in a quickly time, almost faster than the naive method (e.g. $\Theta(n^2)$). The Karatsuba algorithm has a running time of $\Theta(n^{\log 3})$, where $\log 3 \approx 1.58$.

4.2 Karatsuba implementation

Disclaimer: This implementation almost support **long double** data type in C++.

Listing 8: "Karatsuba implementation"

```
#include <iostream>
#include <cmath>
#include <assert.h>
#include <iomanip>

#define number long double
#define maxDigits unsigned long

class Multiplication {
```

```
  number numberOne;
  number numberTwo;
  number base = 2;

public:
  Multiplication(number a, number b) {
    this->numberOne = a;
    this->numberTwo = b;
  }

  number getNumberOne() const {
    return numberOne;
  }

  number getNumberTwo() const {
    return numberTwo;
  }

  void setBase(int newBase) {
    this->base = newBase;
  }

  maxDigits maxNumberOfDigits(number first, number second) {
    auto a = (int) first;
    auto b = (int) second;

    if (a > b) {
      return std::to_string(a).length();
    }

    return std::to_string(b).length();
  }

  number karatsuba(number first, number second) {
    if (first <= 9 && second <= 9) {
      return first * second;
    }

```

```
    maxDigits digits = maxNumberOfDigits(first, second);
    auto m = std::ceil(digits / 2);

    auto exp = std::pow(this->base, m);

    auto x1 = std::floor(first / exp);
    auto x2 = std::fmod(first, exp);

    auto y1 = std::floor(second / exp);
    auto y2 = std::fmod(second, exp);

    number z2 = karatsuba(x1, y1);
    number z0 = karatsuba(x2, y2);
    number z1 = karatsuba(x1 + x2, y1 + y2) - z2 - z0;

    return z2 * std::pow(this->base, m * 2) + z1 * (exp) + z0;
  }
};

int main() {
  auto calculator = new Multiplication(3456, 8750);
  number result = calculator->karatsuba(calculator->getNumberOne(),
                                        calculator->getNumberTwo());

  std::cout << std::setprecision(68);
  std::cout << result;

  return 0;
}
```

5 Wrapping up

Arrange the following functions in increasing order of growth rate with $g(n)$ following $f(n)$ if $f(n) = \mathcal{O}(g(n))$

1. $n^2 log(n)$

2. 2^n
3. 2^{2^n}
4. $n^{log(n)}$
5. n^2

5.1 Solution

1. n^2
2. $n^2 log(n)$
3. 2^n
4. $n^{log(n)}$
5. 2^{2^n}

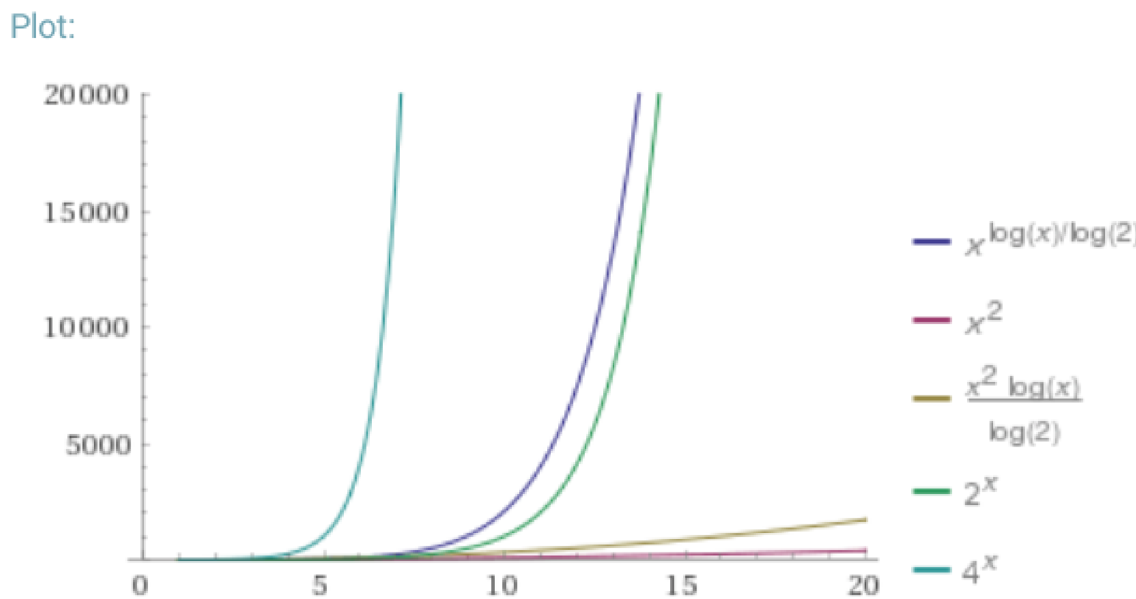


Figure 2: Functions

6 Appendices

6.1 Insert and Merge Sort comparison

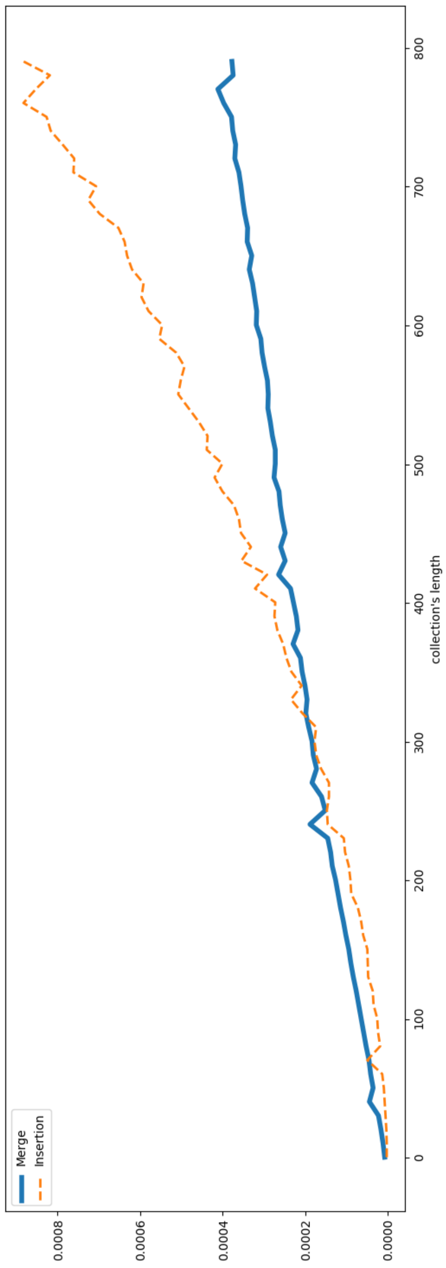
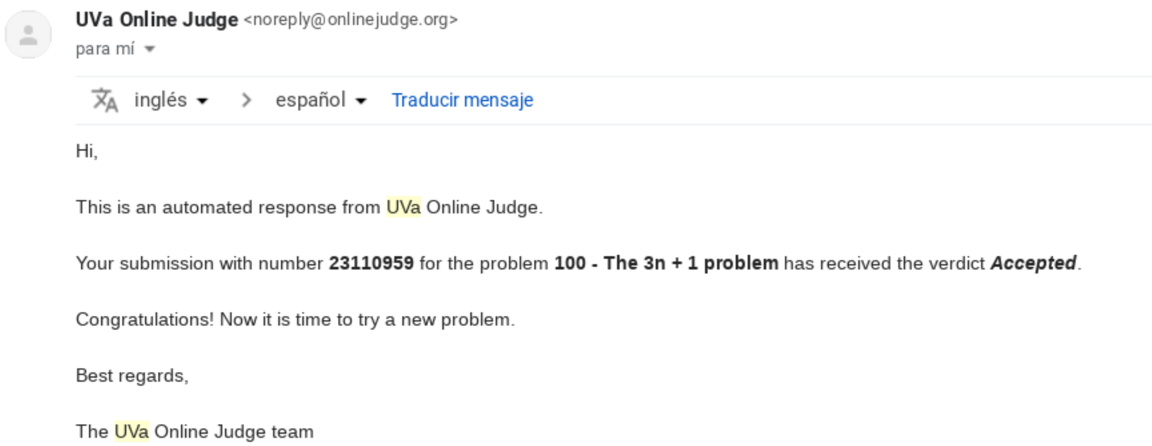


Figure 3: Insertion and Merge Sort comparison

6.2 Proof of submission acceptance



UVa Online Judge <noreply@onlinejudge.org>
para mí

inglés > español Traducir mensaje

Hi,

This is an automated response from UVa Online Judge.

Your submission with number **23110959** for the problem **100 - The 3n + 1 problem** has received the verdict ***Accepted***.

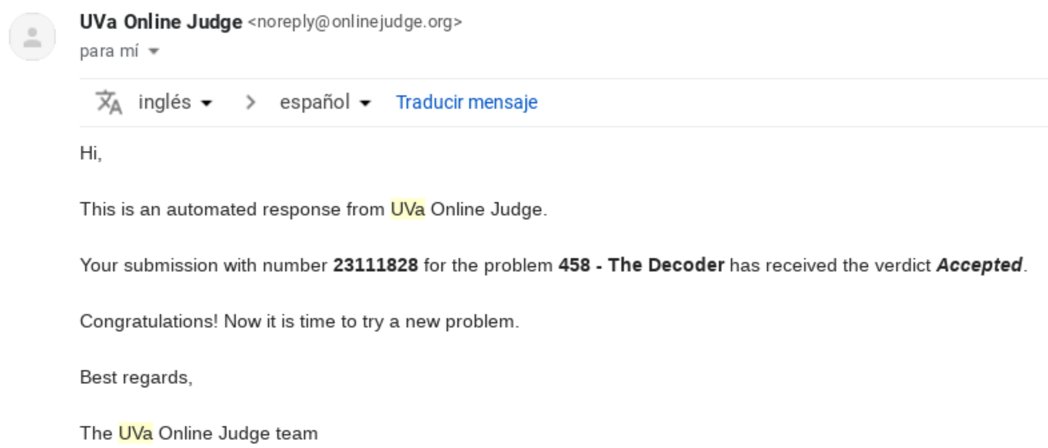
Congratulations! Now it is time to try a new problem.

Best regards,

The UVa Online Judge team

Figure 4: The 3n+1 problem was accepted

6.3 Proof of submission acceptance



UVa Online Judge <noreply@onlinejudge.org>
para mí

inglés > español Traducir mensaje

Hi,

This is an automated response from UVa Online Judge.

Your submission with number **23111828** for the problem **458 - The Decoder** has received the verdict ***Accepted***.

Congratulations! Now it is time to try a new problem.

Best regards,

The UVa Online Judge team

Figure 5: The decoder problem was accepted