Analysis and Design of Algorithms

April 2019

1 Warm up

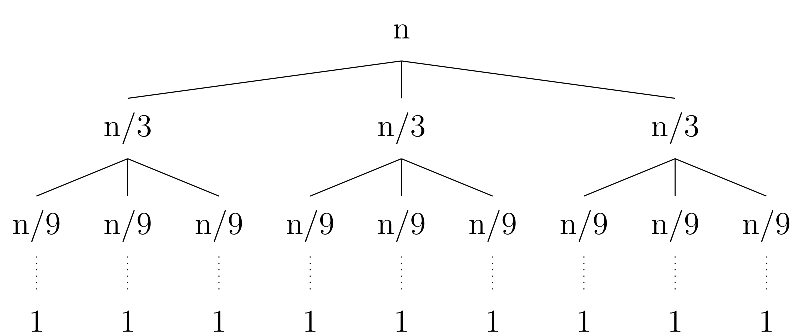
Lets modify the classic merge sort algorithm a little bit. What happens if instead of splitting the array in 2 parts we divide it in 3? You can assume that exists a three-way merge subroutine. What is the overall asymptotic running time of this algorithm?

BONUS: Implement the three-way merge sort algorithm.

Solution

The time execution T(n) for three-way merge, where n is the input size. So *T(n)=3T(n/3)+cn*, where *T(n/3)* solves the subproblems through recursive calls and *cn* do the merger in a constant time.

We have the next tree for three-way merge:



The top level has total cost cn, the next level down has total cost $3c(n/3){=}cn$. In general, each level has total cost cn.

The total number of levels of the recursion tree is $log_3(n) + 1$, where n is the number of leaves, corresponding to the input size. The total time for three-way merge, then, is $cn(log_3(n) + 1) = cn * log_3(n) + cn$. When we use big-O notation to describe this running time, we have a running time of $nlog_3 n$.

Implementation

Three Way Merge Sort File = ('cpp/three-way-merge.cpp')

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. ==> File: cpp/p100-The3n+1.cpp**
- Once that you finish with that problem continue with **458 - The Decoder**. Again, this problem is just to build your confidence in competitive programming. **==> File: cpp/p458-TheDecoder.cpp**
- *BONUS:* **10855 - Rotated squares** ==> **File: cpp/p10855-RotatedSquares**

My Submissions

#	Problem	Verdict	Language	Run Time	Submission Date
23123680	458 The Decoder	Accepted	C++11	0.020	2019-04-07 13:14:12
23123178	100 The 3n + 1 problem	Accepted	C++11	0.000	2019-04-07 10:58:26
23123103	10855 Rotated square	Accepted	C++11	0.010	2019-04-07 10:28:19

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

```
#include <iostream>
#include <vector>

using namespace std;

void InsertionSort(vector<int> &list){
    if(list.size()>1){
        int key;
        for(int i = 1; i < list.size(); i++){
            key = list[i];
            int j = i-1;
            while(j >= 0 && list[j] > key){
                list[j+1] = list[j];
                j--;
            }
            list[j+1] = key;
        }
    }
}
```

3.2 Merge Sort

```
#include <iostream>
#include <vector>

using namespace std;

void Merge(vector<int>vector1,vector<int>vector2,vector<int>&vectorMerge
    vectorMerge.clear();
    long int pos1 = 0, pos2 = 0;
        while(pos1 < vector1.size() && pos2 < vector2.size()){
        if(vector1[pos1]<vector2[pos2]){
            vectorMerge.push_back(vector1[pos1]);
            pos1++;
        }
        else{
            vectorMerge.push_back(vector2[pos2]);
            pos2++;
        }
    }

    while (pos1<vector1.size()) {
        vectorMerge.push_back(vector1[pos1]);
        pos1++;
    }
    while (pos2<vector2.size()) {
        vectorMerge.push_back(vector2[pos2]);
        pos2++;
    }
}

void MergeSort(vector<int> &list){
    if(list.size()>1) {
        vector<int> first, second;
        for (long int i = 0; i < list.size() / 2; i++)
            first.push_back(list[i]);
        for (long int i = (list.size() / 2); i < list.size(); i++)
            second.push_back(list[i]);
```

```
        MergeSort(first);
        MergeSort(second);

        Merge(first, second, list);
    }
}
```

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 ?

Karatsuba multiplication

Given two numbers, x and y, represented as n-digits strings. For $m < n(m = n/2$ is most efficient).
We have :

$$x_1, x_0 : x = x_1 10^m + x_0 \text{ and } y_1, y_0; y = y_1 10^m + y_0 \ ; x, y < 10^m.$$

And the product is

$$xy = (x_1 10^m + x_0)(y_1 10^m + y_0)$$
$$xy = z_2 10^{2m} + z_1 10^m + z_0$$

Where:

$$z_2 = x_1 y_1,\ z_1 = x_1 y_0 + x_0 y_1,\ z_0 = x_0 y_0.$$

And we can observe that:

$$z_1 = (x_1 + x_0)(y_1 + y_0) - z_2 - z_0$$

When we use the algorithm, we have to solve three smaller multiplications. If n is two or more, those products can be computed by recursive calls of the algorithm.

Implementations

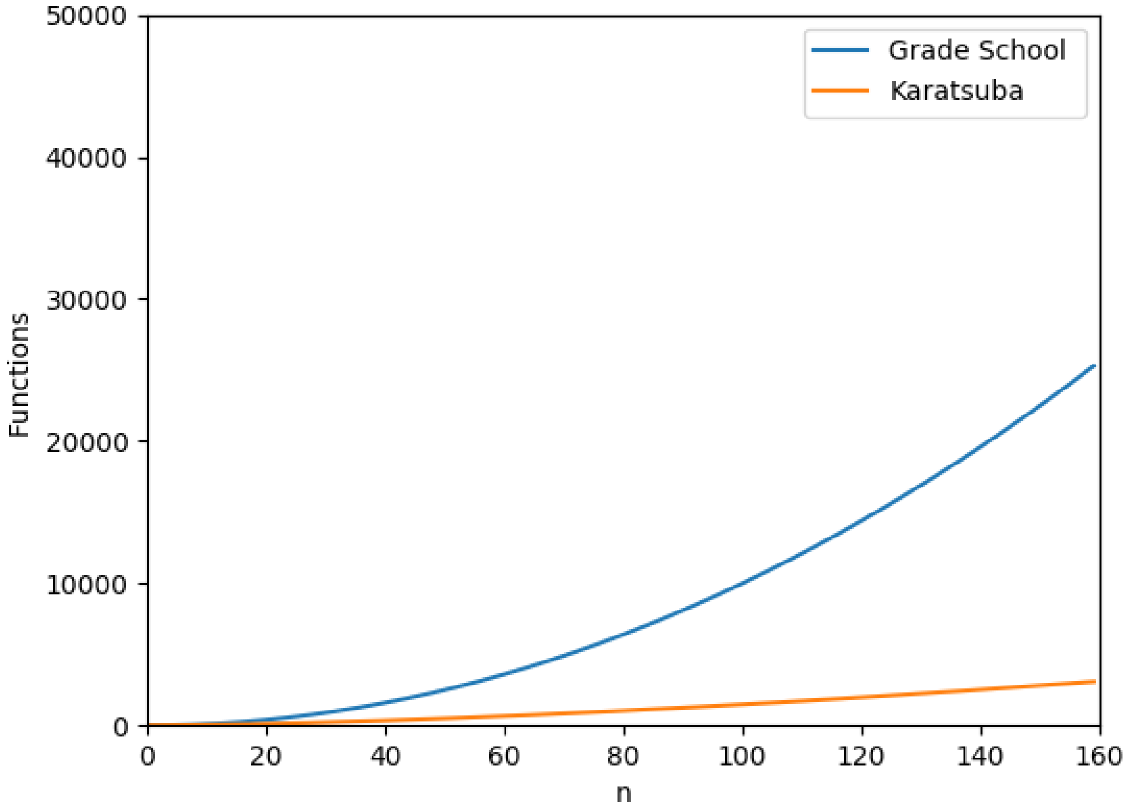
Grade School File: cpp/gsm.cpp
Karatsuba File: cpp/Karatsuba.cpp

Comparison

For any $n > 1$, the number of single-digit multiplications is at most $3n^{lg3}$. Then:

$$T(n) = 3T(n/2) + cn$$

Where, $3T(n/2)$ represents the three multiplications of the two separations of the number and cn is the time proportional it takes to execute the additions, subtractions and digits shifts. This algorithm gives: $T(n) = O(n^{lg3})$, that is faster than grade school multiplication $O(n^2)$.



Exercise

With Grade School Multiplication with:

a: 3141592653589793238462643383279502884197169399375105820974944592

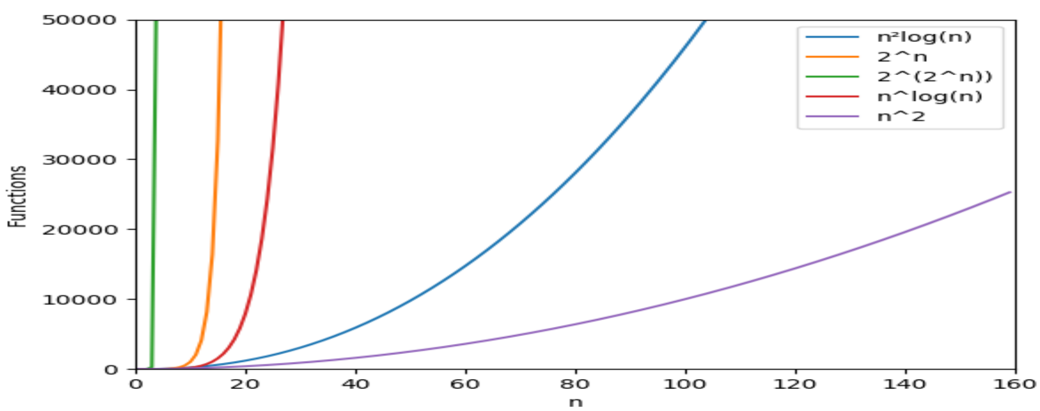
b: 2718281828459045235360287471352662497757247093699959574966967627

a*b= 8539734222673567065463550869546574495034888535765114961879601127067743044893204848617875072216249073013374895871952806582723184

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



Functions in order:

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