FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION
OF HIGHER EDUCATION
ITMO UNIVERSITY

# Report
on the practical task No.8
«Practical analysis of advanced algorithms»

Performed by
Tiulkov Nikita
J4133c

Accepted by
Dr Petr Chunaev

St.Petersburg
2021

# 1. Goal

Practical analysis of advanced algorithms

# 2. Formulation of the problem

I. Choose two algorithms (interesting to you and not considered in the course) from the Cormen's "Introduction to Algorithms Third Edition" book sections.

II. Analyse the chosen algorithms in terms of time and space complexity, design technique used, etc. Implement the algorithms and produce several experiments. Analyse the results.

# 3. Brief theoretical part

## 3.1. The maximum-subarray problem

The maximum sum subarray problem is the task of finding a contiguous subarray with the largest sum, within a given one-dimensional array A[1...n] of numbers. Formally, the task is to find indices $i$ and $j$ with $1 \leqslant i \leqslant j \leqslant n$ such that the sum $\sum_{x=i}^{j} A[x]$ is as large as possible. Each number in the input array A could be positive, negative, or zero.

### 3.1.1. Algorithm

Based on divide and conquer design technique the algorithm is:

- Divide the array into two equal subarrays;
- Recursively calculate the maximum subarray sum for the left subarray;
- Recursively calculate the maximum subarray sum for the right subarray;
- Find the maximum subarray sum that crosses the middle element;
- Return the maximum of the above three sums.

### 3.1.2. Time and space complexity

The time complexity of the above divide-and-conquer solution is O($n \log(n)$) as for the given array of size n, we make two recursive calls on input size n/2 and finding the maximum subarray crosses midpoint takes O(n) time in the worst case. Therefore, T(n) = 2T(n/2) + O(n) = O($n \log(n)$). The space complexity of the above algorithm is O(n).

## 3.2. Huffman Coding

Huffman coding is a lossless data compression algorithm. The idea is to assign variable-length codes to input characters, lengths of the assigned codes are based on the frequencies of corresponding characters. The most frequent character gets the smallest code and the least frequent character gets the largest code.

The variable-length codes assigned to input characters are Prefix Codes, means the codes (bit sequences) are assigned in such a way that the code assigned to one character is not the prefix of code assigned to any other character. This is how Huffman Coding makes sure that there is no ambiguity when decoding the generated bitstream.

### 3.2.1. Algorithm

Based on greedy design technique the algorithm is:

- Build a Huffman Tree from input characters;
- Traverse the Huffman Tree and assign codes to characters.

Steps to build Huffman Tree:

1. Create a leaf node for each unique character and build a min heap of all leaf nodes (Min Heap is used as a priority queue. The value of frequency field is used to compare two nodes in min heap. Initially, the least frequent character is at root)
2. Extract two nodes with the minimum frequency from the min heap.
3. Create a new internal node with a frequency equal to the sum of the two nodes frequencies. Make the first extracted node as its left child and the other extracted node as its right child. Add this node to the min heap.
4. Repeat steps 2 and 3 until the heap contains only one node. The remaining node is the root node and the tree is complete.

### 3.2.2. Time and space complexity

The time complexity of the above greedy solution is O($n \log(n)$), because for encoding each unique character based on its frequency is O($n \log(n)$). Extracting minimum frequency from the priority queue takes place 2*(n-1) times and its complexity is O($\log(n)$). Thus the overall complexity is O($n \log(n)$). The space complexity of the above algorithm is O(n) + O(n) + O(n) = O(3*n). Because we need to store n characters, frequency for each character, and then nodes for each character.

# 4. Results

Two algorithms, that are mentioned above, were implemented. Then experiments for these algorithms were performed. The experiment was as follows:

1. For maximum-subarray problem random integer arrays with size $1 \leq n \leq 2000$ were generated, than for each of arrays 5 experiments of calculation of time were performed. Then the mean value for these experiments were taken and plotted. The results of comparison experimental and theoretical time complexities are shown on Figure 1.
2. For Huffman Coding random characters array with size $1 \leq n \leq 2000$ were generated, than frequencies arrays were generated, for each of arrays 5 experiments of calculation of time were performed. Then the mean value for these experiments were taken and plotted. The results of comparison experimental and theoretical time complexities are shown on Figure 2.

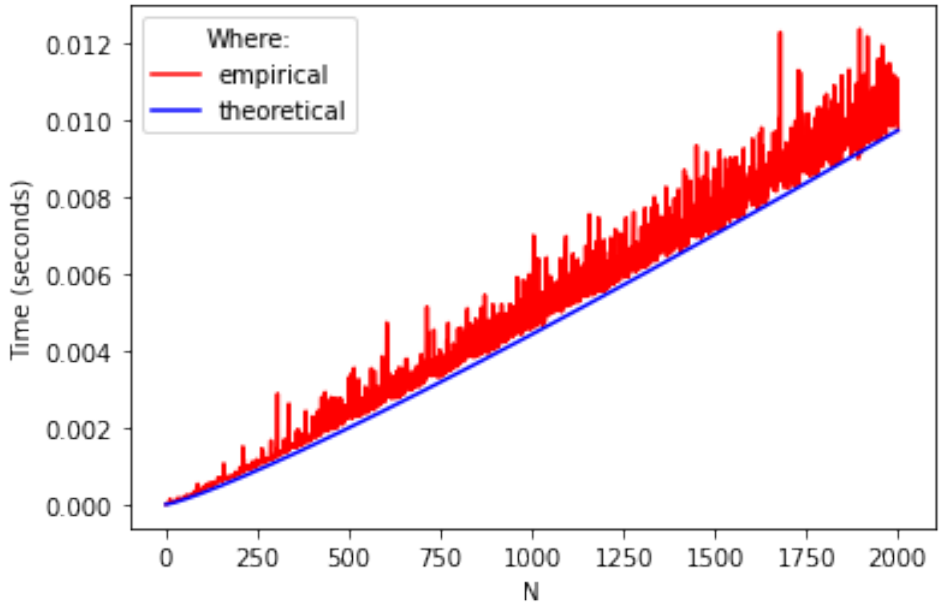


Figure 1: Maximum-subarray problem time

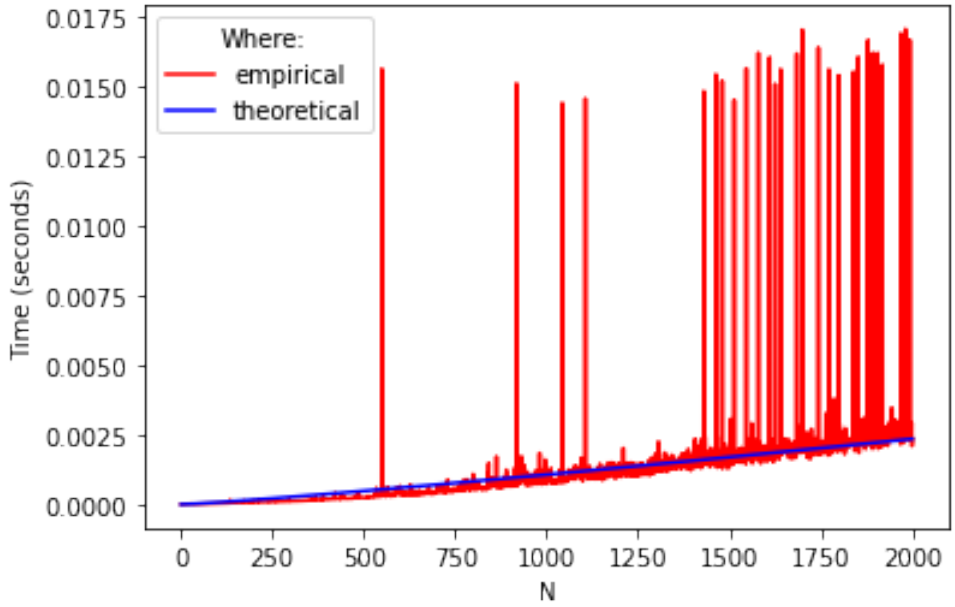


Figure 2: Huffman Coding time

As we can see the empirical and theoretical plots are similar, but certainly there are some "outliers" due to computer implementation.

# 5. Conclusions

As a result of this work, two advanced algorithms from the Cormen's "Introduction to Algorithms Third Edition" book were chosen, learned, implemented, and analyzed. The knowledge gained during the course was applied to perform this work.

# Appendix

The code of algorithms you can find on GitHub: https://github.com/FranticLOL/ITMO_Algorithms