# CC371: Analysis and Design of Algorithms
# Bonus Programming Assignment

Due: January 10th, 2022 (9:00 PM)

## Instructions and Policies

Please read all the following well before starting the assignment.

- This is a bonus optional assignment to help you improve your coursework grades and also to encourage you to understand some of the basic algorithms you studied in the course by coding them. Also, in part 2 of this assignment, you will develop an application that works with large arbitrary files. This will be a good opportunity to learn how to write efficient implementations and train yourselves to think about edge cases.
- If you get a full grade in this assignment, you will get 5 bonus points out of the total course grade. The bonus points will be added to the coursework grades only (midterm and sheets).
- **Academic integrity:** Any violation of academic integrity in this **optional** assignment will not be tolerated. Any cheating in this assignment will result in a zero grade in the coursework and failing the course.
- The assignment has two parts. Both parts must be implemented. Part 1 will not be accepted without a working implementation for part 2.
- This is a group project. You are required to work in groups of three. Use this form to register your group. Note: the ID of student # 1 in your registration will be the ID of your group.
- Coding is to be done in Java.
- Deliverables and discussion.
    - You will deliver your code and a report that has the analysis results as required below.
    - There will also be a discussion on campus. During the discussion, all team members must be ready to answer any question. This means that each team member needs to review and understand each part of the submission well.
    - Submissions (A PDF report and a ZIP file).
        - The PDF should include the analysis required in both parts. The report should be named as report_<id>.pdf, where <id> is the ID of your group.
        - The ZIP file should be named as code_<id>.zip. This should have two directories and a jar.
            - part1_<id>: code for part 1
            - part2_<id>: code for part 2
            - huffman_<id>.jar: The runnable jar for part 2.
        - This is a link to the submission form.

## Part 1 (25%) - Selection Algorithms

We discussed three ways to compute the median in our course. The first task of this assignment is to compare their performance. The three methods are:

- The randomized divide-and-conquer approach [CLRS 9.2].
- The deterministic linear-time selection algorithm using median-of-medians [CLRS 9.3].
- The naive method using sorting and returning the k-th smallest number.

Implement the first two methods only. You can use native sorting for the last one.
Make sure that the returned results of your implementations are correct before doing the analysis to avoid repeating your work unnecessarily.
Study the performance of your implementations as the size of the input array grows (up to $10^7$ elements). The arrays you use for the analysis should be randomly generated. For each data point, you should compute the average of multiple runs.

## Part 2 (75%) - Huffman Compression and Decompression

In this part, it is required to implement Huffman's algorithm that we discussed in the greedy algorithms lecture. Your implementation should allow compressing and decompressing arbitrary files. As discussed in class, the implementation should collect statistics from the input file first, then apply the compression algorithm. Note that you will need to store a reasonable representation of the codewords in the compressed file, so that you can decompress the file back.
Your program should have the capability of considering more than one byte. For example, instead of just collecting the frequencies and finding codewords for single bytes. The same can be done assuming the basic unit is *n* bytes, where *n* is an integer.
The implementation will be graded based on correctness and performance.

### Specifications

- You will submit a single runnable jar that will be used for both compression and decompression. Your jar should be named as **huffman_<id>.jar**. Replace id with your group id. The jar must include the source code files.
  - To use it for compressing an input file, the following will be called:
    ```
    java -jar huffman_<id>.jar c absolute_path_to_input_file n
    ```
    - c means compressing the file.
    - n is the number of bytes that will be considered together.
  - To use it for decompressing an input file, the following be called:
    ```
    java -jar huffman_<id>.jar d absolute_path_to_input_file
    ```

- If the user chooses to compress a file with the name **abc.exe**, the compressed file should have the name **<id>.<n>.abc.exe.hc** where <id> should be replaced by your group id number, and <n> should be replaced by **n** (the number of bytes per group). The compressed file should appear in the same directory of the input file. The program should print the compression ratio and the compression time in seconds.

- If the user chooses to decompress a file with name **abc.exe.hc**, the output file should be named **extracted.abc.exe.** This should appear in the same directory of the input file. You don't need to include the id number here.
  In this case, the program should only print the decompression time in seconds.

## Analysis Requirements

- Study the compression ratio when running your implementation on the following files for different values of n = 1, 2, 3, 4, 5.
  - File 1: This file is from the NIH genetic sequence database. Note that the file is already compressed. Extract the file first, then run the experiments on the uncompressed file (gbbct10.seq). Note that the experiments here could take time, as the file is large. You should test your program and make sure of correctness on small inputs first. Also, try to implement your code efficiently to save time.
  - File 2: The PDF of the greedy algorithms lecture.
- Compare the compression ratio you got in both cases with the compression ratio of 7-zip: https://www.7-zip.org/. Try to explain the observations.
- You can use the default settings of 7-zip in your comparison.
- You can calculate the compression ratio above as the ratio of the size of the compressed file to the size of the uncompressed file.