CSCI 4118/6105 Lab 3:

# External Sort and Search Algorithms

Winter 2022

---

## Objective

1. Implement and analyze Sort and Search algorithms on increasingly large data sets with and without large memory data structures.
2. Compare external memory algorithms with in-memory algorithms.

## Pre-read/Terminologies

1. Merge Sort and K-way Merge Algorithm.
2. Replacement Selection Sort Algorithm.
3. Inverted Index.
4. Matplotlib: Visualization with Python.

## Resources

- For submission: https://git.cs.dal.ca/courses/2022-winter/csci-4118-6105/lab3/???? where ???? is your CSID

## Procedure

**Note:** Undergraduate students are encouraged to attempt additional questions meant for graduate students and get bonus points.

1. Download and install the dependencies (Refer to the installation section in `README.md`)
2. Plot a graph to compare the performance of external merge sort with other in-memory algorithms such as merge sort and Tim sort:, run: `python3 plot_graph.py`
3. Experiment varying the following:
    a. Array Input Size (Length of input array): `input_size`
    b. Chunk/Buffer size (External Sorting): `chunk_size`
    c. Range (Input number range): `input_range`

Improvements and suggestions:

- In case the file sizes of the temporary files for chunks are too large, consider using a binary format instead of storing the integers as strings [4].
- Increase `repeats` and `n_tests` for more accurate results.
- Experiment with different sorting algorithms within external sorting for chunks.

## Questions

1. **All Students:** Follow the procedure and report your findings. Show your work by summarizing in `output.txt` and include graph images in the submission.

2. **Under-graduate students:**
   Write a program to search for an element in an array that cannot be fit into memory. Return the position of the element if present. Explain the worst-case time and space complexity.
   Assumptions:
   a. The array is sorted and contains unique positive integers.
   b. For simplicity, working with a file system is not necessary; instead, write a program for a given input size of the array and block size (Example: input array size is 100 and block size is 10), where `c*block size` is the available memory (`c` is a constant).

3. **Graduate students:**
   Write a program to search for an element in an array that cannot be fit into memory. Return the position of the first occurrence and the number of occurrences of the element if present. Explain the worst-case time and space complexity.

   Assumptions:
   a. The array is sorted and contains positive integers (can contain duplicates).
   b. For simplicity, working with a file system is not necessary; instead, write a program for a given input size of the array and block size (Example: input array size is 100 and block size is 10), where `c*block size` is the available memory (`c` is a constant).

**Hint:** Consider using a map/table to store pre-processed information and use binary search.
**Note:** The worst-case time is measured in the number of I/O operations rather than elementary operations.

## Explanation

Refer to [1] and `README.md` in the GitLab repository of lab3.

## Submission

**Note**:

- For submission - git add, commit and push the answers to the lab3/???? repository and verify the submission in the GitLab web interface.
- Don't commit the input and output CSV files.
- Use any programming language.

1. **All Students:** Answer Q1 (Refer to "questions" section above).

2. **Under-graduate Students:** Write a program for external memory search operation (Refer to Q2 in "questions" section).

3. **Graduate Students:** Write a program for external memory search operation and find the number of occurrences (Refer to Q3 in "questions" section).

## Grading

| Task | 5 Points | 3 Points | 0 Points |
|---|---|---|---|
| 1 | Q1 Thoroughly tested and reported findings in "*output.txt*". | Q1 tested partially and reported findings in "*output.txt*". | No evidence of testing. |
| 2 | Q2 program implemented, tested (works for different inputs) and handles edge cases. | Algorithm implemented, logically correct (pseudo-code), and/or evidence for testing and explanation. | Incorrect answer or not answered. |
| 3 | Q3 program implemented, tested (works for different inputs) and handles edge cases. | Algorithm implemented, logically correct (pseudo-code), and/or evidence for testing and explanation. | Incorrect answer or not answered. |

Grading for Under-graduate Students:

| | **Task 1** | **Task 2** | **Task 3** | **Total** |
|---|---|---|---|---|
| **Points** | 5 | 5 | - | 10 |

Grading for Graduate Students:

| | **Task 1** | **Task 2** | **Task 3** | **Total** |
|---|---|---|---|---|
| **Points** | 5 | - | 5 | 10 |

However, under-graduate students can get an additional +1% for attempting graduate questions.

## References


[1] “External Sorting - Algorithmica,” en.algorithmica.org. https://en.algorithmica.org/hpc/external-memory/sorting/ (accessed Feb. 27, 2022).

[2] “STXXL: STXXL Sorter,” stxxl.org. https://stxxl.org/tags/1.4.1/tutorial_sorter.html (accessed Feb. 27, 2022).

[3] “amh-code/ext-sort at main · sslotin/amh-code,” GitHub. https://github.com/sslotin/amh-code/tree/main/ext-sort (accessed Feb. 27, 2022).

[4] A. Carattino, “Storing Binary Data and Serializing,” Python For The Lab. https://www.pythonforthelab.com/blog/storing-binary-data-and-serializing/ (accessed Feb. 28, 2022).