CCP6214 Algorithm Design and Analysis
Trimester March/April 2025 (Term 2510)

ASSIGNMENT

A. GENERAL INFORMATION

- Assignment mark: **40%**
- Group: **Four (4) members per group**
- Assignment deadline: **29 Jun 2025 (Sunday)**
- Presentation date: **Study Week**
- Presentation duration: **30 minutes per group**

B. REGISTRATION OF GROUP

1. Check with your lab lecturer. Your lab lecturer will assess your assignment.
2. You are by default not allowed to register a group with members from different lab sections to prevent your mark from getting lost accidentally (compiling 800 marks from 22 lab sections is error-prone if cross section is allowed).

C. TASK

It is possible to implement searching in an AVL by using just an array. The steps are:

1. Sort the array.
2. Use binary-search to search the target in the sorted array.

Your tasks:

1. Perform a comparative analysis on the following two sorting algorithms.
    a. Merge-sort
    b. Quick-sort (last element as pivot)
2. Perform an analysis on the best, average, and worst case for binary search.
3. The analysis shall cover the following:
    a. Theoretical analysis and experiment study
    b. Time and space complexities

c. Two programming languages – the sorting and binary search algorithms must be implemented in 2 programming languages. A group can choose any 2 languages.

4. Using a sorting or searching library is not allowed. Using a data structure that performs sorting internally is also not allowed, e.g. TreeSet, TreeMap, or PriorityQueue in Java. The safe data structures to use are array and list (array list or linked list), remember not to use their built-in sorting or searching function.
5. A group should implement an algorithm(s) to generate the dataset as the input to the algorithm. The requirements for the dataset are specified in section DATASET below.
6. For the experiment study:
   a. The running time captured should not include the time for I/O (reading input or printing output).
   b. 10 or more input sizes should be captured.
   c. Each member should run all sorting and searching algorithms using at least one language and include the results in the presentation slide.
7. Conclude the following:
   a. Findings on sorting and searching algorithms in same language and different languages on same and different hardware.
   b. The best sorting algorithm for the array based AVL implementation.
   c. Compare theoretically the implementation of AVL using an array vs linked structure.

D. ALGORITHMS

The algorithms to be implemented are listed below:

No	Algorithm (File Name)	Programming Language	Input	Output
1.	merge_sort_step (add .java extension for Java implementation)	2	• dataset_sample_1000.csv • start row (row number in csv file) • end row	• A file named merge_sort_step_*startrow_endrow*.txt listing the **sorting steps** for elements from start row to end row.
2.	quick_sort_step	2	• dataset_sample_1000.csv • start row • end row	• A file named quick_sort_step_*startrow_endrow*.txt listing the **sorting steps** for elements from start row to end row.
3.	binary_search_step	2	• dataset filename • target (integer)	• A file named binary_search_step_*target*.txt listing the **search path** for target (all the elements that are compared and their row number until the target is found or not found.

4.	dataset_generator	1	• size n	• A file named dataset_*n*.csv with **n randomized unique elements**.
5.	merge_sort	2	• dataset filename	• A file named merge_sort_*n*.csv listing all the elements from the dataset in a **sorted** order. • Print the **running time**.
6.	quick_sort	2	• dataset filename	• A file named quick_sort_*n*.csv listing all the elements from the dataset in a **sorted** order. • Print the **running time**.
7.	binary_search	2	• dataset filename	• A file named binary_search_*n*.txt listing the **running time** for **best, average, and worst cases**. • A single binary search is too fast to be captured. Perform n searches where n is the dataset size.

E. DATASET

A sample dataset of 1,000 elements (dataset_sample_1000.csv) is provided. The first 7 rows are listed below. Each row has 2 fields separated by a comma: integer and string.

```
1981761604,uoren
56205740,igerk
467728380,qouezp
136601853,sitew
1869583452,gslagi
339673152,ufnj
1025900554,rezop
```

A group should generate datasets that are similar to the sample dataset. The requirements for the dataset:

1. The maximum size of the dataset is not specified. However, it should be large enough so that the running time of the two sorting algorithms differs by at least 60 seconds.
2. The integers should be 32-bit, unique, random, positive, up to at least 1 billion (1,000,000,000).
3. The elements in the datasets should be in random order before sorting.

F. PRESENTATION SLIDE

Your presentation slide should contain the following items:

1. Lab section, lab lecturer name, group no, group member's ID, name, and contribution.
2. All items stated in the Task section. If the charts or algorithms do not fit into the slide, put them in a separate PDF/Word/Excel.
3. References in APA format

G. PRESENTATION AND Q&A

1. Present according to the slide contents.
2. Demo your algorithms as listed in the DEMO section below. Explain the complexities of the algorithm using your demo code (no code explanation using slide).
3. Every member must present at least one algorithm.
4. Answer the questions presented.
5. Zero mark for the whole assignment for the absentees.
6. Zero mark for the whole assignment if a group is found plagiarized or shares the solution with another group.

H. DEMO

Perform the following for the demo:

Step No	**Algorithm**	**Input**	**Output**
1.	dataset_generator	• Suggest a dataset size that is not too small to see the result and does not take a dozen seconds to sort. • 1 million (1000000) is used as an illustration. • The lecturer may specify a different size.	• dataset_1000000.csv (unsorted)
2.	merge_sort_step	• dataset_sample_1000.csv • start row (lecturer specifies) • end row (lecturer specifies)	• merge_sort_step_*startrow_endrow*.txt
3.	quick_sort_step	• dataset_sample_1000.csv • start row (lecturer specifies) • end row (lecturer specifies)	• quick_sort_step_*startrow_endrow*.txt
4.	quick_sort	• dataset_sample_1000.csv	• quick_sort_1000.csv (sorted)

5.	binary_search_step	• quick_sort_1000.csv • a found target (lecturer specifies) • a not-found target (lecturer specifies)	• binary_search_step_*target*.txt
6.	merge_sort	• dataset_1000000.csv	• merge_sort_1000000.csv (sorted) • Print the running time
7.	quick_sort	• dataset_1000000.csv	• quick_sort_1000000.csv (sorted) • Print the running time
8.	binary_search	• merge_sort_1000000.csv	• binary_search_1000000.txt (running time for best, average, and worst cases)
9.	Repeat step 2 – 8 for the other language		

Sample merge_sort_step_1_7.txt

```
[1981761604/uoren, 56205740/igerk, 467728380/qouezp, 136601853/sitew, 1869583452/gslagi, 339673152/ufnj, 1025900554/rezop]
[56205740/igerk, 1981761604/uoren, 467728380/qouezp, 136601853/sitew, 1869583452/gslagi, 339673152/ufnj, 1025900554/rezop]
[56205740/igerk, 1981761604/uoren, 136601853/sitew, 467728380/qouezp, 1869583452/gslagi, 339673152/ufnj, 1025900554/rezop]
[56205740/igerk, 136601853/sitew, 467728380/qouezp, 1981761604/uoren, 1869583452/gslagi, 339673152/ufnj, 1025900554/rezop]
[56205740/igerk, 136601853/sitew, 467728380/qouezp, 1981761604/uoren, 339673152/ufnj, 1869583452/gslagi, 1025900554/rezop]
[56205740/igerk, 136601853/sitew, 467728380/qouezp, 1981761604/uoren, 339673152/ufnj, 1025900554/rezop, 1869583452/gslagi]
[56205740/igerk, 136601853/sitew, 339673152/ufnj, 467728380/qouezp, 1025900554/rezop, 1869583452/gslagi, 1981761604/uoren]
```

Sample quick_sort_step_1_7.txt

```
[1981761604/uoren, 56205740/igerk, 467728380/qouezp, 136601853/sitew, 1869583452/gslagi, 339673152/ufnj, 1025900554/rezop]
pi=4  [56205740/igerk, 467728380/qouezp, 136601853/sitew, 339673152/ufnj, 1025900554/rezop, 1981761604/uoren, 1869583452/gslagi]
pi=2  [56205740/igerk, 136601853/sitew, 339673152/ufnj, 467728380/qouezp, 1025900554/rezop, 1981761604/uoren, 1869583452/gslagi]
pi=1  [56205740/igerk, 136601853/sitew, 339673152/ufnj, 467728380/qouezp, 1025900554/rezop, 1981761604/uoren, 1869583452/gslagi]
pi=5  [56205740/igerk, 136601853/sitew, 339673152/ufnj, 467728380/qouezp, 1025900554/rezop, 1869583452/gslagi, 1981761604/uoren]
```

Sample merge_sort.csv (only the first 5 rows are shown)

```
875538,iliheq
1659492,ziuujh
2487583,odtu
2558672,iyavou
```

Sample binary_search_step_2008864030.txt (target found)

500: 1027377159/biaog
750: 1622029193/vveed
875: 1859709030/uiib
938: 1989726533/woiw
969: 2069520854/rauo
953: 2026816381/zfabau
945: 2006875427/aweim
949: 2016987573/avzeq
947: 2011399257/eiiof
946: 2008864030/rdie

Sample binary_search_step_123456789.txt (target not found)

```
500: 1027377159/biaog
250: 511138138/zgeiv
125: 236835705/qoequu
62: 121630136/eafn
93: 173497570/lfepv
77: 141824118/gwizki
69: 128570716/ahia
65: 125960602/pjbiu
63: 123399639/zlvzoi
64: 125177725/iooh
-1
```

I. SUBMISSION FORMAT

Check with your lab lecturer for the submission channel. The submission shall include the following items:

1. The presentation slide, and the supporting document if any.
2. The code

There is no need to submit any dataset file.

J. ASSESSMENT CRITERIA

No	Component	0 – No attempt	1 – Very poor	2 – Poor	3 – Moderate	4 – Good	5 – Excellent
1.	Dataset generation (group)	**No** implementation or **hard-coded** data	• Generate **integers only** Or • Elements **not randomized**	• Generate (integer, string) • **E**lements **randomized** • Integers are **not unique** Or • Generate (string, integer)	• Generate (integer, string) • Elements are **randomized** • Integers are **unique** • Integer range is 0 to **< 1 billion**	• Generate (integer, string) • Elements are **randomized** • Integers are **unique** • Integer range is 0 to > **1 billion** • **Minor issue**	• Generate (integer, string) • Elements are **randomized** • Integers are **unique** • Integer range is 0 to > **1 billion** • All instructions are followed
2.	Merge-sort (group)	**No** complexity analysis and implementation	**Wrong** complexity analysis and implementation	• **Incomplete** complexity analysis **and** incomplete output from demo Or • Implementation sorts elements by **string**	• **Incomplete** complexity analysis **or** output from demo • Implementation sorts elements by **integer**	• **Complete** complexity analysis and demo with **minor issue** • Implementation sorts elements by **integer**	• **Complete** complexity analysis and demo **without issue** • Implementation sorts elements by **integer** • All instructions are followed

3.	Quick-sort (group)	**No** complexity analysis and implementation	**Wrong** complexity analysis and implementation	• **Incomplete** complexity analysis **and** incomplete output from demo Or • Implementation sorts elements by **string**	• **Incomplete** complexity analysis **or** output from demo • Implementation sorts elements by **integer**	• **Complete** complexity analysis and demo with **minor issue** • Implementation sorts elements by **integer**	• **Complete** complexity analysis and demo **without issue** • Implementation sorts elements by **integer** • All instructions are followed
4.	Binary-search (group)	**No** complexity analysis and implementation	**Wrong** complexity analysis and implementation	• **Incomplete** complexity analysis **and** incomplete output from demo Or • Implementation search element by **string**	• **Incomplete** complexity analysis **or** output from demo • Implementation search elements by **integer**	• **Complete** complexity analysis and demo with **minor issue** • Implementation search elements by **integer**	• **Complete** complexity analysis and demo **without issue** • Implementation search elements by **integer** • All instructions are followed
5.	Conclusion (group)	No conclusion and AVL comparison	• Poor conclusion, **not supported** by analysis and experiment • **No/Poor** AVL comparison.	• Moderate conclusion, **somewhat supported** by analysis and experiment • **No/Poor** AVL comparison.	• Moderate conclusion, **somewhat supported** by analysis and experiment • **Moderate** AVL comparison	• Good conclusion, **strongly supported** by analysis and experiment • **Good** AVL comparison	• Excellent conclusion, **strongly supported** by **comprehensive** analysis and experiment • **Excellent** AVL comparison
6.	Slide clarity and completeness (group)	No slide	• **< 50%** required contents	• **50% - 80%** required contents	• Include **>= 80%** contents **without references**	• Complete contents and references with **minor issue**	• **Clear and complete** contents and references **without issue**

7.	Experiments (individual)	**No** experiment result.	**≤ 5 sizes** **OR** **No chart** (table only)	**≤ 10 sizes**	• **≥ 10 sizes** • The running time of the algorithms in the largest dataset are separated by **less than 1 second**. • Cover all sorting and searching algorithms (-1 if an algorithm is not covered, -2 if two)	• **≥ 10 sizes** • The running time of the algorithms in the largest dataset are separated by **less than 60 seconds**. • Cover all sorting and searching algorithms (-1 if an algorithm is not covered, -2 if two)	• **≥ 10 sizes** • The running time of the algorithms in the largest dataset are separated by **at least 60 seconds**. • Cover all sorting and searching algorithms (-1 if an algorithm is not covered, -2 if two)
8.	Presentation and Q&A (individual)	0 mark for the whole assignment if absent or no presentation	Reading slide/note, unable to explain code or answer questions	Poor in presentation, barely able to explain code or answer questions	• Moderate in presentation and Q&A	• Good in both presentation and Q&A	• Excellent in presentation and Q&A

K. MARKSHEET

No	Component	Weight	Actual Mark
1.	Dataset generation (group)	5	
2.	Merge-sort (group)	5	
3.	Quick-sort (group)	5	
4.	Binary-search (group)	5	
5.	Conclusion (group)	5	
6.	Slide clarity and completeness (group)	5	
7.	Experiments (individual)	5	
8.	Presentation and Q&A (individual)	5	
	Total	40	