Extended Topics on DSA

Requirements for completion

This project gets 10% course grade, including 5% for the programming part and 5% for the report.

Students are required to work in **groups, each of which has THREE members**. Any group of more or less than three members must be approved by teachers.

Each group **chooses ONE of the following topics** to complete this project.

#	Topics	Related algorithms or data structures
1	Abstract data structures 1	Self-organizing list and XOR linked list
2	Abstract data structures 2	Leftist heap
3	Abstract data structures 3	Priority queue using min/max binary heap
4	Abstract data structures 4	Reverse Polish Notation
5	Abstract data structures 5	Skew heap
6	Abstract data structures 6	Skip list
7	Data compression 1	Lempel–Ziv–Welch (LZW) compression
8	Data compression 2	Static Huffman compression
9	Search algorithms 1	Interpolation search and Jump search
10	Search algorithms 2	Multiplicative binary search and Meta binary search
11	Search algorithms 3	Ternary search and Exponential search
12	Sorting algorithms 1	Flash sort and Spread sort
13	Sorting algorithms 2	Merge sort variants
14	Sorting algorithms 3	Intro sort and pivot selection strategies
15	Sorting algorithms 4	Properties of sorting algorithms
16	Sorting algorithms 5	Tim sort and Block sort
17	String matching 1	Boyer-Moore and Commentz-Walter
18	String matching 2	Knuth–Morris–Pratt (KMP) and Aho–Corasick

A detailed description will be provided for each topic from Page 3.

Anything beyond the requirements of a topic (e.g., doing extra topics and/or algorithms) will not give you any extra credit; and the TAs have their rights to pick any topic for grading.

The submission will be in Week 10 without any extension.

The TA will interview all groups in Week 11. The date and location will be announced later.

Note that, in the interview, each student must

- Be present. Teachers must approve any absence.
- Understand your group work well: run the source code, explain what written in the report, and answer any question given by the TA

Requirements for submission

Create the folder ***StudentID01_ StudentID02_ StudentID03***. Replace *StudentIDxx* with the student number of each member, sorting the numbers in ascending order.

The above folder includes the following materials.

- SOURCE folder: the project's source code, only files of extensions **.cpp** and **.h** are required.
- Report.pdf: an analytical report of extension **.pdf**.

Compress the above folder into a file of extension **.zip** or **.rar** with the same filename.

Submission that violates any regulation will get no credit (zero).

You will get no credit (zero) for the programming part if your program commits any syntax/runtime error. Furthermore, the statistical analysis in the report will be ignored since it should be completed using results obtained from the program.

Academic ethics

Plagiarism and Cheating will result in a "0" (zero) for the entire course and will be subject to appropriate referral to the Management Board for further action.

The following behaviors are considered as committing plagiarism and cheating.

- Your work is at least 50% identical to other group's work or sources available on the Internet.
- Anything not made by your own (e.g., source code, analyses of algorithms, and demonstration examples) must be cited properly.
- Your output results and/or statistics in report are fabricated.

Topic 01: Abstract data structures 1

You are asked to study the following data structures: **Self-organizing list** and **XOR linked list**

How to prepare your source code

- Implement the basic operations for the given data structures, including
  - Test whether a linked list is empty
  - Get the number of items in a linked list
  - Insert a new item into the list: consider different locations for insertion, if possible
  - Remove an item from the list: consider different locations for removal, if possible
  - Build a linked list from given items
  - Remove all items from the linked list
- Implement the specific operations for the given data structures
- *There is no restriction on how to organize the code. In the main function, provide several simple examples to demonstrate how to use your code.*

How to prepare your report

- For each data structure:
  - Introduce its background: history and applications
  - Briefly mention its variants and improvements (if there are any)
  - Consider a linked list that already has some elements. Show, step by step, how the linked list changes while experiencing a series of item insertion and item removal.
  - Report the time complexity for each basic operation
- Point out the similarities and dissimilarities of the given data structures in comparison to a regular linked list.

Reference to start your research

- Self-organizing list
  https://en.wikipedia.org/wiki/Self-organizing_list
- XOR linked list
  https://en.wikipedia.org/wiki/XOR_linked_list

Topic 02: Abstract data structures 2

You are asked to study the following data structure: **Leftist heap.**

How to prepare your source code

- Implement the basic operations for the given data structure, including
  - Test whether a heap is empty
  - Get the number of items in the heap
  - Return the item at the root of the heap
  - Insert a new item into the heap
  - Remove an item from the heap
  - Build a heap from given items
  - Remove all items from the heap
- Implement the specific operations for the given data structure
- *There is no restriction on how to organize the code. In the main function, provide several simple examples to demonstrate how to use your code.*

How to prepare your report

- Introduce the data structure's background: history and applications. Briefly mention its variants and improvements (if there are any)
- Consider a leftist heap that already has some elements. Show, step by step, how the heap changes while experiencing a series of item insertion and item removal.
- Report the time complexity for each basic operation
- Point out the similarities and dissimilarities of the given data structure in comparison to a regular heap

Reference to start your research

- Leftist heap or Leftist tree
  https://en.wikipedia.org/wiki/Leftist_tree

Topic 03: Abstract data structures 3

You are asked to study the following data structure: **Priority queue using min/max binary heap**.

How to prepare your source code

- Implement the basic operations for the given data structure, including
  - Test whether a priority queue is empty
  - Get the number of items in the priority queue
  - Insert a new item into the priority queue
  - Remove an item from the priority queue
  - Build a priority queue from given items
  - Remove all items from the priority queue
- Implement the specific operations for the given data structure
- *You can choose to implement either max heap or min heap.*
- *There is no restriction on how to organize the code. In the main function, provide a demonstration of inserting items into and removing items from a heap.*

How to prepare your report

- Introduce the data structure's background: history and applications.
- Consider a min/max heap that already has some elements. Show, step by step, how the heap changes while experiencing a series of item insertion and item removal.
- Report the time complexity for each basic operation
- It is obvious that a binary heap, in this case, is a binary tree. Is it a binary search tree? Is it a complete / full / perfect binary tree? Justify your answers.
- Briefly describe the key ideas of at least two binary heap variants

Reference to start your research

- Binary heap
  https://en.wikipedia.org/wiki/Binary_heap
- Priority queue
  https://en.wikipedia.org/wiki/Priority_queue

Topic 04: Abstract data structures 4

You are asked to study the following techniques:

- Represent an arithmetic expression using Reverse Polish Notation.
- Use a stack to convert an expression from infix notation to postfix notation.
- Use a stack to evaluate an arithmetic expression in postfix notation.

How to prepare your source code

- The program inputs an INFIX arithmetic expression, e.g., 3+4. Arithmetic operations include addition (+), subtraction (-), multiplication (*) and division (/). No space between the operand and the operation. For simplicity, the operands are positive integers only.
- The program output a numerical value, which is the result of evaluating the input expression.
- Implement the arithmetic expression evaluation, which has two main steps: convert an expression from infix notation to postfix notation and evaluate an arithmetic expression in postfix notation.
- *There is no restriction on how to organize the code. In the main function, provide several simple examples to demonstrate how to use your code.*

How to prepare your report

- Make a differentiation among infix, postfix, and prefix notations.
- Trace the notation conversion (i.e., from infix to postfix) and the expression valuation, step by step, using several examples. Note that the examples should not be too trivial.
- Beside arithmetic expression evaluation, identify more practical applications of stack

Reference to start your research

- Reverse Polish Notation
  https://en.wikipedia.org/wiki/Reverse_Polish_notation
- Arithmetic expression evaluation using stacks
  https://www.geeksforgeeks.org/expression-evaluation/

Topic 05: Abstract data structures 5

You are asked to study the following heap data structure: **Skew heap**.

How to prepare your source code

- Implement the basic operations for the given data structure, including
  - Test whether a heap is empty
  - Get the number of items in the heap
  - Return the item at the root of the heap
  - Insert a new item into the heap
  - Remove an item from the heap
  - Build a heap from given items
  - Remove all items from the heap
- Implement the specific operations for the given data structure
- *There is no restriction on how to organize the code. In the main function, provide several simple examples to demonstrate how to use your code.*

How to prepare your report

- Introduce the data structure's background: history and applications. Briefly mention its variants and improvements (if there are any)
- Consider a skew heap that already has some elements. Show, step by step, how the heap changes while experiencing a series of item insertion and item removal.
- Report the time complexity for each basic operation
- Point out the similarities and dissimilarities of the given data structure in comparison to a regular heap

Reference to start your research

- Skew heap
  https://en.wikipedia.org/wiki/Skew_heap

Topic 06: Advanced linked lists 6

You are asked to study the following data structure: **Skip list**.

How to prepare your source code

- Implement the skip list, include the following basic operations:
  - Test whether a skip list is empty
  - Get the number of elements in a skip list
  - Insert a new item into the skip list
  - Remove an item from the skip list
  - Build a skip list from given items
  - Remove all elements from the skip list
- Implement the specific operations for the given data structure
- *There is no restriction on how to organize the code. In the main function, provide several simple examples to demonstrate how to use your code.*

How to prepare your report

- Introduce the data structure's background: history and applications. Briefly mention its variants and improvements (if there are any)
- Consider a skip list that already has some elements. Show, step by step, how the skip list changes while experiencing a series of item insertion and deletion.
- Report the time complexity for each basic operation
- Point out the similarities and dissimilarities of the given data structure in comparison to a regular linked list

Reference to start your research

- Skip list
  https://en.wikipedia.org/wiki/Skip_list

Topic 07: Data compression 1

You are asked to study the **Static Huffman coding**. During the study, you may need to refer to the Adaptive Huffman coding.

How to prepare your source code

- Implement the Static Huffman coding for compression and decompression.
- For the compression phase, the program receives a text file and produces a binary file containing the compressed content. For the decompression phase, the program reversely converts the compressed content in the binary file to the original text content.
- *There is no restriction on how to organize the code. In the main function, provide several simple examples to demonstrate how to use your code.*

How to prepare your report

- Introduce its background: history and applications
- Briefly mention the variants and improvements (if there are any) from the original algorithm
- Trace the algorithm, step by step, for both compression and decompression phases, using simple examples
- Analyze the algorithm's time complexity in the best case and worst case
- Point out the similarities and dissimilarities of the two algorithms: Static Huffman coding and Adaptive Huffman coding

Reference to start your research

- Huffman coding
  https://en.wikipedia.org/wiki/Huffman_coding

Topic 08: Data compression 2

You are asked to study the **Lempel–Ziv–Welch (LZW) algorithm** for data compression.

How to prepare your source code

- Implement the LZW algorithm for compression and decompression.
- For the compression phase, the program receives a text file and produces a binary file containing the compressed content. For the decompression phase, the program reversely converts the compressed content in the binary file to the original text content.
- *There is no restriction on how to organize the code. In the main function, provide several simple examples to demonstrate how to use your code.*

How to prepare your report

- Introduce its background: history and applications
- Trace the algorithm, step by step, for both compression and decompression phases, using simple examples.
- Analyze the algorithm's time complexity in the best case and worst case.
- Present a comprehensive overview of LZW-related algorithms, such as LZ77 and LZ78. For each algorithm, just give the overall idea while ignoring the details.

Reference to start your research

- Lempel–Ziv–Welch (LZW) https://en.wikipedia.org/wiki/Lempel%E2%80%93Ziv%E2%80%93Welch

Topic 09: Search algorithms 1

You are asked to study the following two searching algorithms: **Interpolation search** and **Jump search**.

How to prepare your source code

- Implement the given algorithms. Each algorithm accepts three arguments, including an array of positive integers, the array's size, and the number to be sought.
- *There is no restriction on how to organize the code. In the main function, provide several simple examples to demonstrate how to use your code.*

How to prepare your report

- For each algorithm:
  - Introduce its background: history and applications
  - Briefly mention the variants and improvements (if there are any) from the original algorithm
  - Trace the algorithm, step by step, using a simple example.
  - Analyze the algorithm's time complexity in the best case and worst case.
- Complete the following table. Each entry presents the average running time of finding five random numbers on a dataset of 10000, 100000, and 1000000 elements, respectively.

Algorithms \ Data sizes	10,000	100,000	1,000,000
Search algorithm 1			
Search algorithm 2			

  Similarly, make another table for sorted data.
- State your own comments from observations on the above statistics.

Reference to start your research

- Interpolation search
  https://en.wikipedia.org/wiki/Interpolation_search
- Jump search
  https://en.wikipedia.org/wiki/Jump_search

Topic 10: Search algorithms 2

You are asked to study the following two searching algorithms: **Multiplicative binary search** and **Meta binary search**.

How to prepare your source code

- Implement the given algorithms. Each algorithm accepts three arguments, including an array of positive integers, the array's size, and the number to be sought.
- *There is no restriction on how to organize the code. In the main function, provide several simple examples to demonstrate how to use your code.*

How to prepare your report

- For each algorithm:
  - Introduce its background: history and applications
  - Briefly mention the variants and improvements (if there are any) from the original algorithm
  - Trace the algorithm, step by step, using a simple example.
  - Analyze the algorithm's time complexity in the best case and worst case.
- Complete the following table. Each entry presents the average running time of finding five random numbers on a dataset of 10000, 100000, and 1000000 elements, respectively.

Algorithms \ Data sizes	10,000	100,000	1,000,000
Search algorithm 1			
Search algorithm 2			

  Similarly, make another table for sorted data.
- State your own comments from observations on the above statistics.

Reference to start your research

- Multiplicative binary search
  https://en.wikipedia.org/wiki/Multiplicative_binary_search
- Meta binary search
  https://www.geeksforgeeks.org/meta-binary-search-one-sided-binary-search

Topic 11: Search algorithms 2

You are asked to study the following two searching algorithms: **Ternary search** and **Exponential search**.

How to prepare your source code

- Implement the given algorithms. Each algorithm accepts three arguments, including an array of positive integers, the array's size, and the number to be sought.
- *There is no restriction on how to organize the code. In the main function, provide several simple examples to demonstrate how to use your code.*

How to prepare your report

- For each algorithm:
  - Introduce its background: history and applications
  - Briefly mention the variants and improvements (if there are any) from the original algorithm
  - Trace the algorithm, step by step, using a simple example.
  - Analyze the algorithm's time complexity in the best case and worst case.
- Complete the following table. Each entry presents the average running time of finding five random numbers on a dataset of 10000, 100000, and 1000000 elements, respectively.

Algorithms \ Data sizes	10,000	100,000	1,000,000
Search algorithm 1			
Search algorithm 2			

  Similarly, make another table for sorted data.
- State your own comments from observations on the above statistics.

Topic 12: Sorting algorithms 1

You are asked to study the following two sorting algorithms: **Flash sort** and **Spread sort**.

How to prepare your source code

- Implement the given algorithms. Each algorithm accepts an array of positive integers and the array's size as arguments.
- *There is no restriction on how to organize the code. In the main function, provide several simple examples to demonstrate how to use your code.*

How to prepare your report

- For each algorithm:
  - Introduce its background: history and applications
  - Briefly mention the variants and improvements (if there are any) from the original algorithm
  - Trace the algorithm, step by step, using a simple example.
  - Analyze the algorithm's time complexity in the best case and worst case.
- Complete the following table. Each entry presents the average running time of sorting a dataset of 10000, 100000, and 1000000 elements in random order, respectively.

Algorithms \ Data sizes	10,000	100,000	1,000,000
Sorting algorithm1			
Sorting algorithm2			

  Similarly, make two other tables for sorted data and reverse sorted data.
- State your own comments from observations on the above statistics.

Topic 13: Sorting algorithms 2

You are asked to study the **merge sort variants**: in-place merge sort, out-of-place merge sort, and natural merge sort.

How to prepare your source code

- Implement the given algorithms. Each algorithm accepts an array of positive integers and the array's size as arguments.
- *There is no restriction on how to organize the code. In the main function, provide several simple examples to demonstrate how to use your code.*
- *You may reuse your own code of original Merge sort from the lab assignment.*

How to prepare your report

- For each algorithm (out-of-place Merge sort and in-place Merge sort):
  - Trace the algorithm step by step using a simple example.
  - Analyze the algorithm's time complexity in the best case and worst case.
- Complete the following table. Each entry presents the average running time of finding five random numbers on a dataset of 10000, 100000, and 1000000 elements, respectively.

Algorithms \ Data sizes	10,000	100,000	1,000,000
Sorting algorithm 1			
Sorting algorithm 2			
Sorting algorithm 3			

  Similarly, make another table for sorted data.
- State your own comments from observations on the above statistics.
- Discuss the pros and cons of natural merge sort over regular merge sort.

Reference to start your research

- Merge sort
  https://en.wikipedia.org/wiki/Merge_sort

Topic 14: Sorting algorithms 3

You are asked to study the **intro sort** and **pivot selection strategies**.

How to prepare your source code

- Implement the given sorting algorithm and at least three different pivot selection strategies.
- *There is no restriction on how to organize the code. In the main function, provide several simple examples to demonstrate how to use your code.*
- *You may reuse your own code of sorting algorithms from the lab assignments.*

How to prepare your report

- Present at least three different pivot selection strategies. For each strategy, discuss its pros and cons.
- For the sorting algorithm
  - Introduce its background: history and applications
  - Briefly mention the variants and improvements (if there are any) from the original algorithm
  - Trace the algorithm, step by step, using a simple example.
  - Analyze the algorithm's time complexity in the best case and worst case.
- Complete the following table. Each entry presents the average running time of finding five random numbers on a dataset of 10000, 100000, and 1000000 elements, respectively.

Algorithms \ Data sizes	10,000	100,000	1,000,000
Intro sort			
Quick sort using median-of-three pivot			

  Similarly, make another table for sorted data.
- State your own comments from observations on the above statistics.

Reference to start your research

- Intro sort
  https://en.wikipedia.org/wiki/Introsort
- Choice of pivot
  https://en.wikipedia.org/wiki/Quicksort#Choice_of_pivot

Topic 15: Sorting algorithms 4

Consider the following algorithms

- Bubble sort
- Selection sort
- Insertion sort
- Heap sort
- Quick sort
- Merge sort
- Radix sort
- Counting sort

Study their properties of stability, in-place, and parsimony.

Study their Big-O time complexities, for the best, worst, and average cases

How to prepare your source code

- Provide the code segments to check whether a sorting algorithm (any of the above) is stable
- *There is no restriction on how to organize the code. In the main function, provide several simple examples to demonstrate how to use your code.*
- *You may reuse your own code from the lab assignments.*

How to prepare your report

Fill in the following table with corresponding information about stability, in-place, and parsimony

Algorithms	Stability	In-place	Parsimony	Note
Bubble sort				
… …				

* Put a check mark to a cell if the algorithm has the corresponding property. Add notes when necessary.

* Do not consider the parsimony property for non-comparison sorting algorithms.

Fill in the following table with corresponding information about Big-O time complexities.

Algorithms	Best case	Average case	Worst case	Note
Bubble sort				
… …				

* Add notes if the algorithm has many variants with different time complexities

Topic 06: Advanced sort 2

You are asked to study the following two sorting algorithms: **Tim sort** and **Block sort**

How to prepare your source code

- Implement the given algorithms. Each algorithm accepts an array of positive integers and the array's size as arguments.
- *There is no restriction on how to organize the code. In the main function, provide several simple examples to demonstrate how to use your code.*

How to prepare your report

- For each algorithm:
  - Introduce its background: history and applications
  - Briefly mention the variants and improvements (if there are any) from the original algorithm
  - Trace the algorithm, step by step, using a simple example.
  - Analyze the algorithm's time complexity in the best case and worst case.
- Complete the following table. Each entry presents the average running time of sorting a dataset of 10000, 100000, and 1000000 elements in random order, respectively.

Algorithms \ Data sizes	10,000	100,000	1,000,000
Sorting algorithm 1			
Sorting algorithm 2			

  Similarly, make two other tables for sorted data and reverse sorted data.
- State your own comments from observations on the above statistics.

Reference to start your research

- Tim sort
  https://en.wikipedia.org/wiki/Timsort
- Block sort
  https://en.wikipedia.org/wiki/Block_sort

Topic 17: String matching 1

You are asked to study the following two string matching algorithms:

- Boyer-Moore
- Commentz-Walter, an extension of Boyer-Moore

How to prepare your source code

- Implement the two given algorithms
- *There is no restriction on how to organize the code. In the main function, provide several simple examples to demonstrate how to use your code.*

How to prepare your report

- For each algorithm:
  - Introduce its background: history and applications
  - Briefly mention the variants and improvements (if there are any) from the original algorithm
  - Trace the algorithm, step by step, using a simple example
  - Analyze its time complexity in the best case and worst case
- Point out the similarities and dissimilarities of the two algorithms

Reference to start your research

- Boyer-Moore
  https://en.wikipedia.org/wiki/Boyer%E2%80%93Moore_string-search_algorithm
- Commentz-Walter
  https://en.wikipedia.org/wiki/Commentz-Walter_algorithm

Topic 18: String matching 2

You are asked to study the following two string matching algorithms:

- Knuth–Morris–Pratt (KMP)
- Aho–Corasick, an extension of KMP

How to prepare your source code

- Implement the two given algorithms
- *There is no restriction on how to organize the code. In the main function, provide several simple examples to demonstrate how to use your code.*

How to prepare your report

- For each algorithm:
  - Introduce its background: history and applications
  - Briefly mention the variants and improvements (if there are any) from the original algorithm
  - Trace the algorithm, step by step, using a simple example
  - Analyze its time complexity in the best case and worst case
- Point out the similarities and dissimilarities of the two algorithms

Reference to start your research

- Knuth-Morris-Pratt https://en.wikipedia.org/wiki/Knuth%E2%80%93Morris%E2%80%93Pratt_algorithm
- Aho-Corasick https://en.wikipedia.org/wiki/Aho%E2%80%93Corasick_algorithm