Durham University

Computer Science

Summative Assignment

Module code and title	COMP1081 Algorithms and Data Structures
Academic year	2024-25
Coursework title	ADS Coursework Assignment
Coursework credits	6.8 credits
% of module's final mark	34%
Lecturer	Eamonn Bell and Thomas Erlebach
Submission date*	Thursday, 23 January 2025 at 14:00
Estimated hours of work	13.6
Submission method	Gradescope

Additional coursework files	q1_supplementary.py q2_supplementary.py q3_supplementary.py q4_supplementary.py q7_supplementary.py
Required submission items and formats	q1.py – python file q2.py – python file q3.py – python file q4.py – python file q7.py – python file solution.pdf – pdf file with written answers to Q5, Q6, Q7(b), Q7(c)

* This is the deadline for all submissions except where an approved extension is in place. For benchtests taking place in practical sessions, the given date is the Monday of the week in which the benchtests will take place.

Late submissions received within 5 working days of the deadline will be capped at 40%.
Late submissions received later than 5 days after the deadline will receive a mark of 0.
It is your responsibility to check that your submission has uploaded successfully and obtain a submission receipt.

Your work must be done by yourself (or your group, if there is an assigned groupwork component) and comply with the university rules about plagiarism and collusion. Students suspected of plagiarism, either of published or unpublished sources, including the work of other students, or of collusion will be dealt with according to University guidelines (https://www.dur.ac.uk/learningandteaching.handbook/6/2/4/).

Algorithms & Data Structures 2024/25 Coursework

Eamonn Bell & Thomas Erlebach

Hand in by 23 January 2025 at 2pm

Attempt all questions. Partial credit for incomplete solutions may be given. This assignment counts for 34% of your module mark.

The following instructions on submission are important. You need to submit a number of files to Gradescope, and we automate some of the marking. If you do not make the submission correctly, some of your work might not be looked at and you could miss out on marks.

Python Files

Python programs are required for questions 1, 2, 3, 4, 7(a) and must be submitted to Gradescope as python files (i.e., as text files with extension .py containing python code). If you write your python code for a question in a jupyter notebook, be sure to export it as python file (using the File $\rightarrow$ Dowload as $\rightarrow$ Python (.py) operation) and submit that python file. The submitted python files MUST be named as follows:

- `q1.py` containing the functions `compute_winner`, `encode`, `decode`, and `compute_winner_compressed`.
- `q2.py` containing the functions `show`, `cat`, `smart_cat`, `make_queue`, `enqueue`, and `convert_to_array_queue`.
- `q3.py` containing the functions `hash_quadratic` and `hash_double`.
- `q4.py` containing the functions `ordering` and `all_orderings`.
- `q7.py` containing the three functions `SelectPivotPair`, `ThreePartition` and `ThreeWayQuickSort`.

We will use Python 3 to test your submissions. The supplementary python files that have been provided contain some simple tests so that you can check if your functions work correctly on basic inputs before submission. For questions 2 and 3, the supplementary python files contain code that you require to solve the question. When your work is assessed, a wider range of inputs will be tested and, in some cases, efficiency may be assessed as well.

Written Work

Written answers are required for questions 5, 6, 7(b) and 7(c). Your written answers can be typed or handwritten and scanned, but in the latter case it is your responsibility to make sure your handwriting is clear and easily readable. It is recommended that you prepare a single pdf-file (which you can name solution.pdf) with all your written answers. During the submission process on Gradescope, you can select for each question the pages of your pdf-file that contain the answer to the question.

Plagiarism

Your work must be done by yourself and comply with the university rules about plagiarism and collusion (https://durhamuniversity.sharepoint.com/teams/LTH/SitePages/6.2.4.aspx). Please remember that you must not share your work or make it available where others can find it as this can facilitate plagiarism and you can be penalised. This requirement applies until the assessment process is completed which does not happen until the exam board meets in June 2025. If you base any code you write or written answers on sources other than the lecture notes, you must clearly cite the source within the code or written answer.

---

1. This question tests your understanding of basic algorithmic thinking and elementary data structures.

   Player A flips a coin n times. Then, Player B independently flips the same coin another n times. Each player writes down a history representing the outcomes of each of their flips as a sequence of characters from the set $\{H, T\}$, where H represents "heads" and T represents "tails".

   The winner of the game is the player that produces the longest uninterrupted run of heads. If neither player produces any heads, or if both players produce an equally long uninterrupted run of heads, then the game is tied (i.e. a draw).

   For example, if $n = 6$, then one game can be represented as a pair of histories:

   $$\text{history}_A = \text{`}HTTHHT\text{`}$$
   $$\text{history}_B = \text{`}THHHHT\text{`}$$

   The game whose histories are history_A and history_B is won by player B, because the longest uninterrupted run of heads that player B has is 4, while the the longest uninterrupted run of heads that player A has is 2.

   Further notice that we can produce a compressed form of history_A, which is represented as a string $\text{compressed\_history}_A = \text{`}H1T2H2T1\text{'}$.

   The compressed representation of the history consists of successive pairs of characters from the set $\{H, T\}$ and single-digit, positive integers. Each pair stands for the outcome corresponding to the character repeated that integer number of times. For example, '$H3T1$' stands for the history '$HHHT$', which is a sequence of 3 heads and 1 tail.

   Compressed histories are represented as "densely" as possible; for example, some history that might be incorrectly represented as the string '$H2H2T1T1H2T1$' will be correctly compressed as '$H4T2H2T1$', ensuring that runs of different outcomes appear in strict alternation. For the rest of this question, you may assume that $n \leq 9$.

   (a) Write a function `compute_winner(history_A, history_B)` that computes the winner of the game, returning 'A' if A wins, 'B' if B wins, and 'D' if A and B are tied. **[2 marks]**

   (b) Write a function `encode(history)` that takes as input a string representing a game history and returns its compressed representation. **[2 marks]**

   (c) Write a function `decode(compressed_history)` that takes as input a compressed representation of a game history and returns its uncompressed representation. **[2 marks]**

   (d) Finally, write a function `compute_winner_compressed(compressed_history_A, compressed_history_B)` that computes the winner of a game given two *compressed* histories. Your implementation must not use your implementations of `compute_winner`, the encoder/decoder pair, or some other technique that is identical to decoding the compressed history before computing the winner as before. **[2 marks]**

You will be rewarded for correct, concise, and efficient implementations. Your implementations will be tested on a variety of inputs. A final two marks will be awarded if you can apply the use of recursion in your implementation of any one of the functions above. **[2 marks]**

You must not use any `import` statements in your solution. Include all of your functions in the same Python file `q1.py`.

2. This question tests your understanding of linked lists, arrays, and queues.

   For the rest of the question, you will be working with singly linked lists. To complete this question, you must use the implementations of `Node`, `LinkedList`, and `LinkedListWithTail` that are given in the supplementary material. You must not modify these implementations. It is not necessary to understand their implementation to solve this question but it is important to know how they can be used. Unless otherwise stated, you should use `LinkedList` in your solutions to this question.

   For example, if you want to create some nodes, you can do as follows:

   ```
   N1 = Node(data=7)
   N2 = Node(data=4)
   N3 = Node(data=3)
   ```

   Now you can introduce a link structure between them:

   ```
   N1.next = N2
   N2.next = N3
   ```

   Make sure that the node representing the last element in the logical ordering points to an appropriate value (`None` is a good approximation for the NULL special value that we use in pseudocode):

   ```
   N3.next = None
   ```

   And, finally, create a new linked list with `N1` at its head:

   ```
   linked_list = LinkedList(head=N1, size=3)
   ```

   If you mutate `linked_list`, you can change the size attribute (e.g. by decrementing it):

   ```
   linked_list.size = linked_list.size - 1
   ```

   Note that the `LinkedList` implementation does not store a reference to the tail of the list. A `LinkedListWithTail` is a `LinkedList` with an extra attribute, `tail`, that points to the tail of the linked list:

   ```
   linked_list_with_tail = LinkedListWithTail(head=N1, tail=N3, size=3)
   ```

   (a) Write a function `show(linked_list)` which prints the values from a given linked list, in the order that they appear in the linked list, each separated by a newline and no other character. **[2 marks]**

   (b) Write a function `cat(linked_list_a, linked_list_b)`, which returns a singly linked list that is the concatenation of the two inputted linked lists. The returned list contains all the nodes of `linked_list_a` followed by all the nodes of `linked_list_b`. You should not build up a new list from scratch. Your implementation may modify either one or both of the inputted lists. **[3 marks]**

(c) Now assume `linked_list_a` and `linked_list_b` are instances of `LinkedListWithTail`. Write a function `smart_cat(linked_list_a, linked_list_b)`, which returns a singly linked `LinkedListWithTail` that is the concatenation of the two inputted linked lists. The returned list contains all the nodes of `linked_list_a` followed by all the nodes of `linked_list_b`. You should not build up a new list from scratch. Your implementation may modify one or both of the inputted lists. **[3 marks]**

(d) Write a function `make_queue()` which creates and returns a `LinkedListWithTail` that contains the contents of the following queue $Q = \{4, 9, 18, 3, 21\}$, where 4 is at the front of the queue. You only need to return a `LinkedListWithTail` and do not need to implement any of the queue methods in your answer. **[2 marks]**

(e) Write a function `enqueue(ll_queue, value)` which returns a `LinkedListWithTail` that represents the contents of the queue after the value `value` has been enqueued into the queue represented by the linked list `ll_queue`. **[3 marks]**

(f) Finally, write a function `convert_to_array_queue(ll_queue)` that takes an instance of `LinkedList` and returns a tuple `(A, f, r)`, where:

- A is a Python `list` of length 10 that acts as an array of size 10 containing the data in `ll_queue` at appropriate locations. You can use the Python value `None` where there is no data relevant to the queue to be found at a given location of the array.
- f is an `int` with an appropriate value that facilitates access to the front of the queue.
- r is an `int` with an appropriate value that facilitates access to the rear of the queue.

[2 marks]

You must not use any `import` statements in your solution. Include all of your functions in the same Python file `q2.py`. It should not depend on anything other than the classes given in the supplementary material, which you must not modify or include in your submission and will be added to your code at marking time.

3. This question tests your understanding of hash tables.

Study the code in `q3_supplementary.py`, including the definition of the class `HashTable`, which provides some but not all of the functionality of a hash table.

(a) Write a function `hash_quadratic`, which takes as input a list D, which contains a sequence of positive integer keys to be inserted to an empty hash table in the order they appear in the list. You must use quadratic probing to resolve collisions in your implementation of `hash_quadratic`. **[7 marks]**

(b) Write a function `hash_double`, which takes as input a list D, which contains a sequence of positive integer keys to be inserted to an empty hash table in the order they appear in the list. You must use double hashing to resolve collisions in your implementation of `hash_double`. **[8 marks]**

For a key k, the hash function to be used is:

$$3k + 5 \pmod{17}.$$

The secondary hash function to be used is:

$$13 - (k \pmod{13}).$$

Your solution should not depend on anything other than the `HashTable` class given in the supplementary material, which you must not modify or include in your submission and will be added to your code at marking time.

You do not need to store the values associated with the keys inserted in the hash table. Your solution will be tested on a variety of inputs. You may also additionally assume any list D that these functions may receive as input will not contain any duplicates.

Note that although `HashTable` implements a method that adds data to a hash table at a given position, you will need to provide more code to determine *where* within the hash table you wish to store the data that you are inserting.

You must not use any `import` statements in your solution. Your submission for this question should be a single file `q3.py` containing two functions `hash_quadratic` and `hash_double`.

4. Let us call a permutation of the k integers $[0, 1, 2, \ldots, k-1]$ a Boldon House ordering of size k if every pairwise difference of successive integers mod k is monotone increasing. Concretely, if $R = [r_0, r_1, r_2, \ldots, r_{k-1}]$, and $D = [(r_1 - r_0) \mod k, (r_2 - r_1) \mod k, \ldots, (r_{k-1} - r_{k-2}) \mod k]$, then R is a Boldon House ordering if all elements of D are monotone increasing.

   A sequence is said to be monotone increasing if each element is greater than or equal to the previous element. Formally, a sequence of pairwise differences $D = [d_1, d_2, \ldots, d_{k-1}]$ is monotone increasing if for all i such that $1 \leq i < k-1$, $d_i \leq d_{i+1}$.

   For example, is $R^* = [1, 3, 5, 2, 0, 4]$ a Boldon House ordering of size 6? First, it is evidently a permutation of the integers $[0, 1, 2, 3, 4, 5]$. Second, consider the pairwise differences (mod 6) between successive elements:

   - $3 - 1 \mod 6 = 2$
   - $5 - 3 \mod 6 = 2$
   - $2 - 5 \mod 6 = 3$
   - $0 - 2 \mod 6 = 4$
   - $4 - 0 \mod 6 = 4$

   As above, $D^* = [2, 2, 3, 4, 4]$ and all of its elements are monotone increasing. So, yes - this is a Boldon House ordering. **Note**: You do not need to consider "looping around" i.e. the pairwise difference between the first and the last element.

   (a) Implement a function `ordering(k)` that returns one Boldon House ordering of size k. **[4 marks]**

   (b) Implement a function `all_orderings(k)` that returns all Boldon House orderings of size k. **[6 marks]**

   Your implementations **must** use backtracking. Your implementations will be tested on a variety of choices of k. The autograder will time out after 10 minutes when grading this question. Anything that exceeds this limit will still be marked. Marks will be awarded for the greatest values of k for which correct answers to `all_orderings(k)` can be produced within this time. (For reference: it is possible to get all orderings for $k = 12$ in well under a minute when backtracking is implemented correctly). Your submission for this question should be a single file `q4.py` containing at least two functions `ordering` and `all_orderings`. You may define other functions in this file, if required by your implementation of these two functions.

5. Prove or disprove each of the following statements. You will get 1 mark for correctly identifying whether the statement is True or False, and 1 mark for a correct argument.

   (a) $10x + 7x^2 + 3x^3$ is $O(x^3)$. **[2 marks]**

   (b) $x^{0.99}$ is $o(x)$. **[2 marks]**

(c) $4^{\log_2 x} - x$ is $\Theta(x^2)$. **[2 marks]**

(d) 3^x is $\Omega(4^x)$. **[2 marks]**

(e) $2^{(\log_2 x)^2}$ is $\omega(x)$. **[2 marks]**

6. The Master Theorem applies to recurrences of the form

$$T(n) = aT(n/b) + f(n)$$

for constants $a \geq 1$ and $b > 1$ and gives a solution in the following three cases:

Case 1: **If** $f(n) = O(n^{\log_b(a)-\epsilon})$ for some constant $\epsilon > 0$ **then** $T(n) = \Theta(n^{\log_b(a)})$.

Case 2: **If** $f(n) = \Theta(n^{\log_b(a)} \cdot \log^k n)$ with $k \geq 0$ **then** $T(n) = \Theta(n^{\log_b(a)} \log^{k+1} n)$. (Note that $\log^k n$ stands for $(\log n)^k$.)

Case 3: **If** $f(n) = \Omega(n^{\log_b(a)+\epsilon})$ for some constant $\epsilon > 0$ **and** if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all n large enough **then** $T(n) = \Theta(f(n))$.

For each of the following recurrences, give an expression for the runtime $T(n)$ if the recurrence can be solved with the Master Theorem. Otherwise, state why the Master Theorem cannot be applied. You should justify your answers. For each recurrence, you will get 2 marks for a correct answer and 1 mark for a correct justification.

(a) $T(n) = 3 \cdot T(n/3) + n(\log_2 n)^3$ **[3 marks]**

(b) $T(n) = 4 \cdot T(n/2) + 8n$ **[3 marks]**

(c) $T(n) = 8 \cdot T(n/2) + 5n^4$ **[3 marks]**

(d) $T(n) = n \cdot T(n/2) + n$ **[3 marks]**

(e) $T(n) = 8 \cdot T(n/4) + 4n^{1.5}$ **[3 marks]**

7. Consider the problem of sorting a list L of numbers into increasing order. You can assume that no two of the given numbers are equal. This question is about a variant of QuickSort that uses two pivots and makes three recursive calls. Furthermore, the selection of the two pivots is done by selecting 5 distinct random elements and choosing the second-smallest and the second-largest of those as pivots. Your submission for part (a) of this question must be a file `q7.py` that contains your python code for functions `SelectPivotPair`, `ThreePartition` and `ThreeWayQuickSort` and that is part of your submission of Python Files. Your submission for parts (b) and (c) must be part of your submission of Written Work.

   (a) The QuickSort algorithm we have seen in lectures chooses one pivot, partitions the given list into one list with numbers that are smaller than the pivot, the pivot element itself, and one list with numbers that are larger than the pivot. We want to change the algorithm as follows (and your task is to implement this modified algorithm in python):

   - If the input list L has at most 10 elements, the list must be sorted using the built-in sort() method of python lists and returned.
   - Otherwise, 5 elements from **distinct** random positions of L are put into a new list S, and that list S is sorted using the built-in sort() method of python lists. Then the second smallest and second largest element of S are chosen as pivots P_0 and P_1, respectively. We then partition L into three sublists: Sublist 1 contains all elements of L that are smaller or equal to P_0; sublist 2 contains all elements of L that are larger than P_0 but smaller or equal to P_1; sublist 3 contains all elements of L that are larger than P_1. The three sublists are sorted using recursive calls, and the results of the three recursive calls are concatenated and returned.

To implement this algorithm, you must complete the following three tasks:

i. Write a python function `SelectPivotPair` that takes a list L with at least 5 integers as input and returns a pair (P_0, P_1) of elements of L that are selected as described above. You can import the `random` module and use one or several suitable function(s) from that module to select 5 distinct random elements from L.

```
def SelectPivotPair(L):
    """Return a pair (P0,P1) of pivot elements

    Select 5 distinct random elements from L, sort them,
    and let P0 be the second smallest and P1 the
    second largest of those 5 elements.
    """
```

[5 marks]

ii. Write a python function `ThreePartition` that takes a list L and two pivot elements P_0 and P_1 with $P_0 < P_1$ as input and returns a tuple with three sublists of L.

```
def ThreePartition(L,P0,P1):
    """Return a triple (L0,L1,L2) of sublists of L

    L0 consists of all elements of L smaller or equal P0,
    L1 of all elements of L larger than P0 but smaller or
    equal P1, and L2 of all elements of L larger than P1
    """
```

[5 marks]

iii. Write a python function `ThreeWayQuickSort` that takes a list L of distinct integers as input and returns a sorted version of that list. If L has at most 10 elements, the function must sort L using the built-in sort() method and return L. Otherwise, the function must call `SelectPivotPair` to select (P_0, P_1), then call `ThreePartition` to partition L into the three sublists L_0, L_1 and L_2, make recursive calls on these three sublists to sort them, and then concatenate the results of those recursive calls:

```
def ThreeWayQuickSort(L):
    """Return a sorted version of L

    Use SelectPivotPair, ThreePartition and recursive
    calls to sort L.
    """
```

[5 marks]

The marking of your implementation of these functions will mainly consider correctness, but may also take into account efficiency.

(b) Assume that the 5 random elements that are selected in the `SelectPivotPair` function to form the list S are always the elements with rank $n/6$, $2n/6$, $3n/6$, $4n/6$ and $5n/6$ in L, where n denotes the length of L. Here, the *rank* of an element in L is its position (a number between 1 and n) in the sorted version of L. Analyze the worst-case running-time of `ThreeWayQuickSort` as a function of n under this assumption. (For simplicity, you can assume that n is a multiple of 6.) **[5 marks]**

(c) If we choose the smallest and the largest element of S as our pivots P_0 and P_1 in the `SelectPivotPair` function, and if the assumption that S always consists of the elements with rank $n/6$, $2n/6$, $3n/6$, $4n/6$ and $5n/6$ in L still holds, what will be the worst-case running-time of `ThreeWayQuickSort` as a function of n now? You can again assume that n is a multiple of 6. **[5 marks]**