

# CS2040C Data Structure and Algorithm PE

AY2024/25 Sem 1

Time Allowed: 2 hours

---

## Before PE Starts

1. Please **remember your coursemology login and password**. You need to log in to coursemology at the start of the PE with the password and login id. All submissions are through coursemology.
2. You will be logged into a special Windows account at the beginning of the PE. Do not use your own NUSNET account. **Do not log off** until the end of the PE. Do not close your coursemology in your browser.
- You have only **5** submissions for each part (instead of unlimited #submissions in assignments.) So please test out your code in your IDEs first before submitting them on coursemology. Do not use coursemology as a debugger to test your code. Please make sure your code can be compiled and runnable after you submitted to Coursesmology. Any crashing code will be zero mark.
3. Please remember that, once you are in a lab, you've already ENTERED the exam. And you should NOT communicate with any others anymore. Any form of communication with other students or the use of unauthorized materials is considered cheating and you are liable to disciplinary actions.
4. Throughout the PE, ALWAYS keep your Coursemology PE page opened. Closing it will cost you time to ask for a password to reopen it. It will also cause us to check on you. Do NOT open another Coursemology pages.
5. This is NOT an open-book assessment. You can only bring one piece of A4 size cheat sheet and one piece of blank paper, but **not** any electronic devices. You have to switch off/silence your mobile phone/smartwatch and **keep it out of view**. You cannot access your lecture slides and past assignment submissions on coursemology.

## During PE

1. You are advised to start your Visual Studio because it takes about 5 min or more to start VS for the first time.
2. If you are the first time using Visual Studio, click the .sln file after you unzip each part.
3. When you are told to stop, please do so **immediately**, or you will be penalized.
4. **You will be forced to log out at the time the PE ends SHARPLY**. Make sure you save and submit your work a few minutes (> 5 min) before it ends. The network will be jammed at the last few min of the assessment because everyone is submitting at the same time. **It's your own responsibility to submit the code to Coursemology on time**.
5. **Your program must be able to be compiled!** Or you will receive zero mark for that part.
6. All of the PCs will be running Windows and using **MSVS 2022**. You may have to choose the MSVS version because some of the machines may be installed with more than one version of MSVS. Some of the machines may be installed with MSVS Code also. Use it if you want.
7. The best browser is Firefox, because it is a lot faster than Chrome.
8. Any variables used must be declared within some functions. You are not allowed to use global variables
9. You should only modify the functions stated in the questions. You cannot include other extra libraries such as STL (except specified).
10. For all the classes provided, you can add more member functions if you need them (better private). However you cannot change the signatures (input arguments and output types) of the provided functions.

## Advice

- Read all the questions and plan before you start coding. It is not necessary that the first question is the easiest.
- Manage your time well! Do not spend excessive time on any task. If you are stuck, MOVE ON!
- Please save and backup your code regularly during the PE. Better to keep different versions.
- It is a bad idea to do major changes to your code at the last 15 minutes of your PE.

## Part 1 Linked List Interval Reversal (40 marks)

In this part, you are NOT allowed to include any library other than those provided. And you should not change `LinkedList.h`.

You are given the skeleton code similar to Assignment 1. You are given an integer `List` class with the following member functions implemented:

- `push_head(int element)` : To insert the element into the head of the list
- `print()`: To print the current list from the head to the tail
- `to_string()`: to return a string representation of the list

You should NOT modify any of the above functions in this part. You do not need to implement other functions but you can if you want. You only have to implement the function `reverse_interval()` mentioned below.

### Task

Your task is to implement a member function `reverse_interval(int a, int b)` that will reverse the elements from the  $(a+1)^{\text{th}}$  to  $(b+1)^{\text{th}}$  of the list. Here are some sample example of the test.

```
Original list
{1, 2, 3, 4, 8, 5, 3, 5, 6, 2, 9, 5, 1, 4, 1, 3}

reverse_interval(1, 4)
{1, 8, 4, 3, 2, 5, 3, 5, 6, 2, 9, 5, 1, 4, 1, 3}

reverse_interval(0, 3)
{3, 4, 8, 1, 2, 5, 3, 5, 6, 2, 9, 5, 1, 4, 1, 3}

reverse_interval(10, 6)
{3, 4, 8, 1, 2, 5, 9, 2, 6, 5, 3, 5, 1, 4, 1, 3}
```

You can assume  $0 \leq a, b < \text{size of the list}$ . However, you cannot assume  $a \leq b$ . Your function has to be efficient in space and time complexities. For example, you cannot create another array or list to copy and store the intermediate reversed list. And you have to use linked list data structure in this implementation.

### Submission

Please copy and paste your `LinkedList.cpp` file as your submission. You can exclude the line:

- `#include "LinkedList.h"`

## Part 2 MinHeap Heapify (30 marks)

In this part, you are NOT allowed to include any library other than those provided.

You are given the skeleton of our heap assignment. A bit different this time, we need to implement a **MinHeap**. There is a new constructor `Heap(int size, T* vec, bool printStep = true)` that will takes in an unsorted array `vec` with size `size`.

### Task

Your job is to implement the  $O(n)$  heapification mentioned in our lecture in the constructor:

```
Heap(int size, T* vec, bool printStep = true)
```

If the variable `printStep` is true, your constructor should print out every step of the heapification during the fly by using the given function `printHeapArray()` (and you should not modify `printHeapArray()`.) Here is an example with the given array for the constructor input (in the provided skeleton code).

```
vec[] = { 3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 4, 3, 2, 1 }.
```

The underlined numbers are the ones swapped before the printing.

```
3 1 4 1 5 9 2 6 5 3 5 8 4 3 2 1
3 1 4 1 5 9 2 6 5 3 5 8 4 3 2 1
3 1 4 1 5 9 2 6 5 3 5 8 4 3 2 1
3 1 4 1 5 9 2 6 5 3 5 8 4 3 2 1
3 1 4 1 5 9 2 6 5 3 5 8 4 3 2 1
3 1 4 1 5 9 2 6 5 3 5 8 4 3 2 1
3 1 4 1 5 9 2 6 5 3 5 8 4 3 2 1
3 1 4 1 5 9 2 6 5 3 5 8 4 3 2 1
3 1 4 1 5 9 2 6 5 3 5 8 4 3 2 1
3 1 4 1 5 9 2 6 5 3 5 8 4 3 2 1
3 1 4 1 5 9 2 6 5 3 5 8 4 3 2 1
3 1 4 1 5 9 2 6 5 3 5 8 4 3 2 1
3 1 4 1 5 9 2 1 5 3 5 8 4 3 2 6
3 1 4 1 5 9 2 1 5 3 5 8 4 3 2 6
3 1 4 1 5 4 2 1 5 3 5 8 9 3 2 6
3 1 4 1 3 4 2 1 5 5 5 8 9 3 2 6
3 1 4 1 3 4 2 1 5 5 5 8 9 3 2 6
3 1 2 1 3 4 2 1 5 5 5 8 9 3 4 6
1 1 2 1 3 4 2 3 5 5 5 8 9 3 4 6
```

If `printStep == false`. Your constructor will be silent and there is no need to print out anything.

### Submission

Please copy and paste your `heap.hpp` into Coursemology. You should not modify the two functions `printHeapArray()` and `printTree()`. The function `printTree()` is just to help you to debug and you should only use `printHeapArray()` for your output.

### Notes

In case you forgot about the heap array implementation, here are the index relationships of an item with index  $x$  in the heap array:

- The index of the parent :  $\text{floor}((x-1) / 2)$
- The index of the left child :  $2x+1$
- The index of the right child :  $2x+2$

### Part 3 Amoeba Kingdom (30 marks)

In this part, you are allowed to use the libraries `algorithm` and `vector`.

You are the king of the amoeba and you have a lot of amoeba warriors as your minions. However, because you have just defeated all of your enemies, now your own amoeba warriors are a threat to you. Let's think of a way to weaken their power!

Each of your amoeba warrior has an integer number that represents its power. For example, if you got four warriors and their powers are:

- Aryx: 2499
- Bert: 1899
- Carl: 2399
- Dave: 2099
- Eragorn: 1120

Their total power will be 8896. (It is so threatening!!!). The higher the power is, the amoeba is more powerful.

However, just right after you finished your conquests and defeat all of your opponents, you started to worry about your army. Will your own army turn against you? Time to think of a plan to weaken them! You do not want to let them know your evil plan, so you organize an "**Amoeba Gladiator Games**"! You let them to kill each other!

The match is as follows, you will pair them up. For each pair of amoeba warriors, they will fight and the one with more power wins. When it wins, it CONSUME its opponent! Lucky that the winner will not increase in power but just enjoy a very satisfying meal. If the number of warriors are odd, one unlucky amoeba will stay out of the match.

For example, if we match Aryx with Bert, and Carl with Dave (in which Eragorn will stay out of the match). Bert and Dave will be consumed and the final remaining total will be  $2499 + 2399 + 1120 = 6018$ . You successfully weakened your own army from 8896 to 6018 in total power!

However, you can only do one round of pairing, and your goal is to minimize the final total as much as possible. After some clever re-matching, you find out that you can make your army weakened to **5718** power in total if you match them differently.

#### Task

You are provided with a class `AmoebaGladiatorGame(int* powers, int size)` that will take in an integer array `powers` with size `size`. Write the function `minRemainingPower()` that returns the minimal total power remaining after an optimal matching of the amoeba warriors in an integer. The following should be the output for your test cases in the provided code. You can assume each power score for each amoeba is positive.

```
{2499,2399,1899,2099,1120} gives a minimal remaining total power of 5718  
{76,23,42,11,42,11} gives a minimal remaining total power of 129
```