COMP 2402 AB- Fall 2023
Assignment #4

Due: Wednesday, November 22, 23:59

Submit early and often. Late submissions (up to 12 hours) will be accepted.

Academic Integrity

You may:

- Discuss general approaches with course staff and your classmates,
- Use code and/or ideas from the textbook,
- Use a search engine / the internet to look up basic Java syntax.

You may not:

- Send or otherwise share code or code snippets with classmates,
- Use code not written by you, unless it is code from the textbook (and you should cite it in comments),
- Use a search engine / the internet to look up approaches to the assignment,
- Use code from previous iterations of the course, unless it was solely written by you,
- Use the internet to find source code or videos that give solutions to the assignment.

If you ever have any questions about what is or is not allowable regarding academic integrity, please do not hesitate to reach out to course staff. We will be happy to answer. Sometimes it is difficult to determine the exact line, but if you cross it the punishment is severe and out of our hands. Any student caught violating academic integrity, whether intentionally or not, will be reported to the Dean and be penalized. Please see Carleton University's Academic Integrity page.

Grading

This assignment will be tested and graded by a computer program (and **you can submit as many times as you like; your highest grade is recorded**). For this to work, there are some important rules you must follow:

- Keep the directory structure of the provided **zip** file. If you find a file in the subdirectory `comp2402a4` leave it there.
- Keep the package structure of the provided **zip** file. If you find a package `comp2402a4;` directive at the top of a file, leave it there.

- Do not rename or change the visibility of any methods already present. If a method or class is public leave it that way.
- Submit early and often. The submission server compiles and runs your code and gives you a mark. You can submit as often as you like and only your best submission will count. There is no excuse for submitting code that does not compile or does not pass tests.
- Write efficient code. The submission server places a limit on how much time it will spend executing your code, even on inputs with a million lines. For some questions it also places a limit on how much memory your code can use. If you choose and use your data structures correctly, your code will easily execute within the time limit. Choose the wrong data structure, or use it the wrong way, and your code will be too slow for the submission server to grade (resulting in a grade of 0).

Submitting and Testing

The submission server is available here. If you have issues, please post to Discord to the teaching team (or the class) and we'll see if we can help.

Warning: Do not wait until the last minute to submit your assignment. There is a hard 5 second limit on the time each test has to complete. For the largest tests, even an optimal implementation takes 3 seconds, and may take longer if the server is heavily loaded.

Start by downloading and decompressing the Assignment 4 Zip File (comp2402a4.zip), which contains a skeleton of the code you need to write. The skeleton code in the **zip** file compiles fine. Here's what it looks like when you unzip and compile it from the command line:

```
alina@euclid:~$ unzip comp2402a4.zip
Archive:  comp2402a4.zip
  inflating: comp2402a4/UltraStack.java
  inflating: comp2402a4/UltraSlow.java
  inflating: comp2402a4/UltraFast.java
  inflating: comp2402a4/Tester.java
alina@euclid:~$ javac comp2402a4/*.java
```

For this assignment you need to implement `UltraFast`. The `Tester` class, included in the zip file gives a very basic demonstration of the code. Despite the name, `Tester` does not do thorough testing. The submission server will do thorough testing. To run the `Tester` from the command line:

```
alina@euclid:~$ java comp2402a4.Tester
```

The Assignment

This assignment contains only one part worth 100 marks.

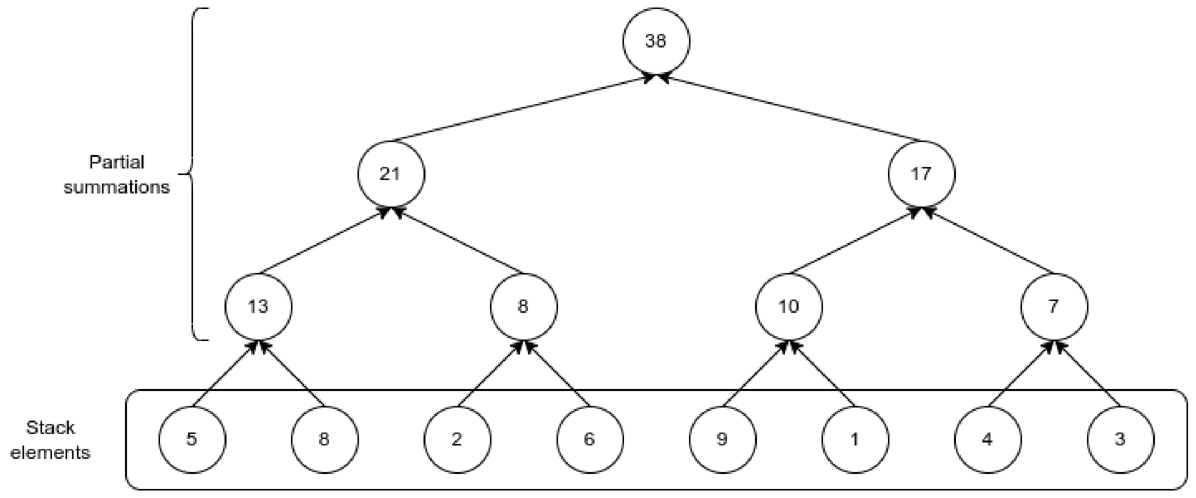
A `UltraStack` is an extended stack that supports six main operations: the standard Stack operations `push(x)` and `pop()` and the following non-standard operations:

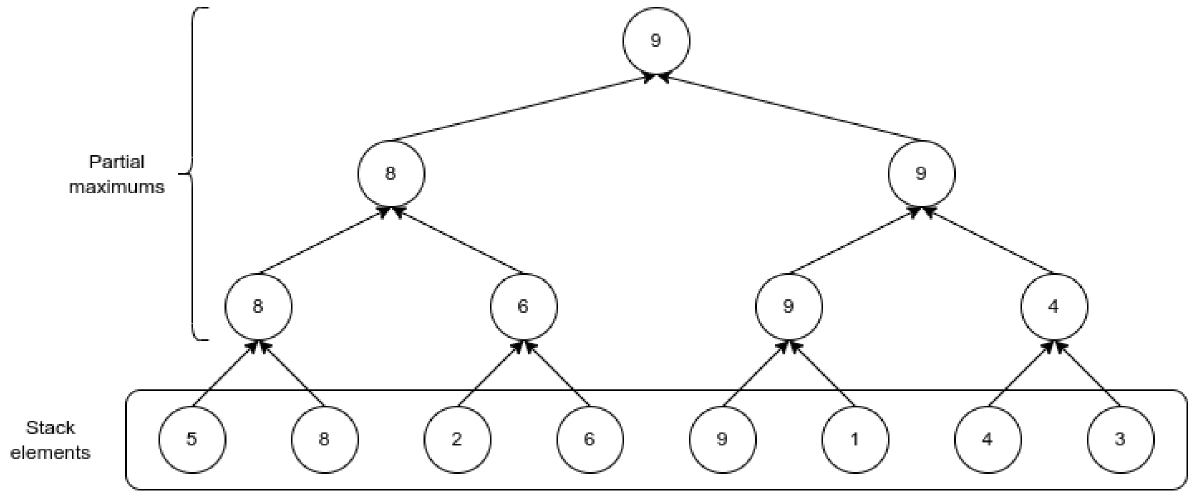
- `get(i)`: returns the `i` -th element (from the bottom) on the Stack.
- `set(i, x)`: sets the value of the `i` -th element (from the bottom) on the Stack to `x`.
- `max()`: returns the maximum value stored on the Stack.
- `ksum(k)`: returns the sum of the top `k` elements on the Stack.

The zip file gives an implementation of `UltraSlow` which implements these operations so that `push(x)`, `pop()`, `get(i)` and `set(i, x)` each run in $O(1)$ time, but `max()` and `ksum(k)` run in $O(n)$ time. You have to complete the implementation of `UltraFast` that implements all six operations. For `UltraFast`, the running time for `get(i)` and `max()` must be $\boldsymbol{O(1)}$, and for `push(x)`, `pop()`, `set(i, x)`, and `ksum(k)` running time must not exceed $\boldsymbol{O(\log n)}$. As part of your implementation, you may use any of the classes in the Java Collections Framework and you may use any of the source code provided with the Java version of the textbook. Don't forget to also implement the `size()` and `iterator()` methods. The `iterator()` method returns an `Iterator<Integer>` that iterates over the values in order, starting at the bottom and ending at the top of the Stack.

Think carefully about your solution before you start coding. Here are two hints:

1. Storing the partial summations and the partial maximums can save a lot of computation when you update an element in the Stack. The following figure (on the next page) might help you better understand the idea.
2. Complete binary trees are much simpler to store implicitly using arrays. Heaps use the same idea to store the elements.


Partial summations
38
21
17
13
8
10
7
Stack elements
5
8
2
6
9
1
4
3


Partial maximums
9
8
9
8
6
9
4
Stack elements
5
8
2
6
9
1
4
3