**Programming Assignment 3**
*Due: February 26, 2018*

# 0 Instructions

Submit your work through Canvas. You should submit a tar file containing all source files and a README for running your project. Don't submit any other files (e.g., test case or pyc files).

More precisely, submit on Canvas a tar file named lastname.tar (where lastname is your last name) that contains:

- All source files. You can choose any language that builds and runs on ix-dev.

- A file named README that contains your name and the exact commands for building and running your project on ix-dev. If the commands you provide don't work on ix-dev, then your project can't be graded and there will be a significant penalty.

Here is an example of what to submit:

hampton.tar
  README
  my_class1.py
  my_class2.py
  my_class3.py
  problem.py
  another_problem.py
  ...

```
README
  Andrew Hampton

  Problem 1: python problem1.py <input_filename>
  Problem 1: python problem1.py <input_filename>
  ...
```

Note that Canvas might change the name of the file that you submit to something like lastname-N.tar. This is totally fine!

The grading for the project will be roughly as follows:

| **Task** | **Points** |
|---|---|
| Problem 1 | **15** |
| pass given sample test case | 5 |
| pass small grading test case | 5 |
| pass large grading test case | 5 |
| Problem 2 | **20** |
| pass given sample test case | 5 |
| pass small grading test case | 5 |
| pass large grading test case | 10 |
| Problem 3 | **15** |
| pass given sample test case | 5 |
| pass small grading test case | 5 |
| pass large grading test case | 5 |
| TOTAL | **50** |

# 1 Applications

**Problem 1. Prioritizing HTTP Requests**
Suppose you are working on a web application that receives HTTP requests over a LAN in batches. When you receive a batch of requests, they have already been preprocessed with an estimate of how long it will take your application to complete each of them.

You want to serve the requests in order of the estimated service time, with the shortest requests being served first.

Additionally, your application has two tiers of service: A and B. All of the requests in the A tier should be served before any of the requests in the B tier.

Write a program that will give the correct service order according to the above criteria. Your program **must use one or more priority queues** to accomplish this!

In the event of a tie in service time (and tier), the request that appears first in the input list should be served first (that is, your sort should be *stable*).

Your program should take a single command-line argument, which will be a filename. The input file will contain request strings. The first line of the input file will be an integer $0 \leq N \leq 10^6$ giving the number of requests. Following will be $N$ lines, each containing a string having the following format:

```
IP_ADDR TIER ESTIMATE
```

IP_ADDR is an IP address in decimal IPv4 format. TIER is either *A* or *B*. ESTIMATE is an integer $0 < X \leq 10^4$ representing a time estimate for processing the request. The separator is a single space.

Output the IP addresses in the order the requests will be served, separated by newlines. Again, in the event of a tie in service time (and tier), the request that appears first in the input list should be served first.

Example input file:

```
8
10.31.99.245 B 30
10.16.0.105 A 150
10.16.115.160 B 60
10.30.111.90 B 65
10.16.0.105 A 20
10.30.100.100 A 25
10.16.100.115 A 150
10.111.111.119 B 60
```

Example output:

```
10.16.0.105
10.30.100.100
10.16.0.105
10.16.100.115
10.31.99.245
10.16.115.160
10.111.111.119
10.30.111.90
```

**Problem 2. Rolling Median**
Suppose you have streaming (integer) data and want to compute some summary statistics. It's easy to calculate the cumulative rolling average: this is a constant time operation (look up the formula if you're interested!). What about the cumulative rolling median?

In this problem, you'll develop an algorithm to compute the cumulative rolling median and test it on simulated streaming data.

We will use the most common definition of *median*, as described on the Wikipedia page. We can simulate streaming data by giving a list of integers $L$ of length $n$ and calculating the median on the slice $L[1:i]$ for every $0 < i \leq n$.

Your program should take a single command-line argument, which will be a filename. The input file will contain integers, one per line. The first line of the input file will be an integer $2 \leq N \leq 10^5$ giving the number of integers in the list $L$. Following will be $N$ lines, each containing an integer $0 \leq x \leq 10^6$.

You should output the median of the slice $L[1:i]$ for every $0 < i \leq N$, with a newline between each result. If a median is not an integer, it should be printed to one decimal place. If a median is an integer, it should be printed as an integer (i.e., without a decimal point). See the sample output below.

Your solution should have runtime complexity $O(n \log n)$.

Hint: use two binary heaps, a maxheap to hold the smaller half of the data and a minheap to hold the larger half of the data. The median of the data is either at the top of one of the heaps or it's the average of those two values.

Example input file:

```
5
1
8
4
3
2
```

Example output:

```
1
4.5
4
3.5
3
```

The first line of the sample input says that the file contains 5 integers. So, we will read in 5 integers and with each new integer compute the median of those we have seen so far.

The first integer is 1 and the median of $\{1\}$ is 1. The next integer is 8 and the median of $\{1, 8\}$ is 4.5. The next integer is 4 and the median of $\{1, 8, 4\}$ is 4. The next integer is 3 and the median of $\{1, 8, 4, 3\}$ is 3.5. The final integer is 2 and the median of $\{1, 8, 4, 3, 2\}$ is 3.

———

# 2 Implementation

**Problem 3. Binary Search Tree**
For this problem, you will implement a binary search tree with integer keys. Do not use any builtin tree structures that your language might have. You must implement your own tree class that satisfies the binary search tree property as described in Chapter 12 (p. 287) of the textbook.

Your binary search tree data structure should implement (at least) the following methods with specified runtime, where $h$ refers to the height of the tree:

`insert(X)`: Inserts a node X into the tree. $O(h)$

`remove(X)`: Removes a node X from the tree. This method should be implemented as described in the textbook (pp. 295-298). In particular, use the in-order *successor* as the replacement node. Runtime: $O(h)$

`search(X, K)`: Returns a node in the subtree rooted at node X having key K, if present. Runtime: $O(h)$

`maximum(X)`: Returns the node in the subtree rooted at node X having the largest key. Runtime: $O(h)$

`minimum(X)`: Returns the node in the subtree rooted at node X having the smallest key. Runtime: $O(h)$

`to_list_preorder()`: Returns a list of the keys in the tree ordered by a pre-order traversal. Runtime: $O(n)$

`to_list_inorder()`: Returns a list of the keys in the tree ordered by an in-order traversal. Runtime: $O(n)$

`to_list_postorder()`: Returns a list of the keys in the tree ordered by a post-order traversal. Runtime: $O(n)$

(Depending on the programming language you use, replace *a node* with *a pointer to a node* as appropriate.)

Note: It's important that you implement the remove method as described. The pre- and post-order traversals will not match the reference output if implemented differently.

Note: The behavior of these methods in exceptional cases is unspecified. You should think about what these cases might be, and raise appropriate exceptions.

Note: No test case will insert duplicate keys into the tree.

Write a driver program that takes a single command-line argument, which will be a filename. The input file will contain instructions for tree operations. The first line of the input file will be an integer $0 \leq N \leq 10^6$ giving the number of instructions. Following will be $N$ lines, each containing an instruction. The possible instructions are:

`insert K`, where $-10^5 \leq K \leq 10^5$ is an integer: insert a node with key K into the tree. There is no output.

`remove K`, where $-10^5 \leq K \leq 10^5$ is an integer: remove a node with key K from the tree. If such a node exists, there is no output. If no such node exists, output *TreeError*.

`search K`, where $-10^5 \leq K \leq 10^5$ is an integer: output *Found* if a node exists with key K. If no such node exists, output *NotFound*.

`max`: output the maximum key in the tree. If the tree is empty, output *Empty*.

`min`: output the minimum key in the tree. If the tree is empty, output *Empty*.

`preprint`: print the keys of the tree according to a pre-order traversal, separated by a single space. If the tree is empty, output *Empty*.

`inprint`: print the keys of the tree according to an in-order traversal, separated by a single space. If the tree is empty, output *Empty*.

`postprint`: print the keys of the tree according to a post-order traversal, separated by a single space. If the tree is empty, output *Empty*.

Example input file:

```
20
inprint
remove 2
max
search 5
insert 1
insert 2
search 1
search 2
insert 3
inprint
insert 10
insert 5
inprint
preprint
postprint
search 5
remove 2
inprint
max
min
```

Example output:

```
Empty
TreeError
Empty
NotFound
Found
Found
1 2 3
1 2 3 5 10
1 2 3 10 5
5 10 3 2 1
Found
1 3 5 10
10
1
```