University of Waterloo
CS240 Spring 2021
Assignment 1

Written Questions Due: Wednesday, May 26 at 5:00pm
Programming Question Due: Wednesday, June 2 at 5:00pm

The integrity of the grade you receive in this course is very important to you and the University of Waterloo. As part of every assessment in this course you must read and sign an Academic Integrity Declaration before you start working on the assessment and submit it **before the deadline of May 26th** along with your answers to the assignment; i.e. **read, sign and submit A01-AcInDe.txt now or as soon as possible**. The agreement will indicate what you must do to ensure the integrity of your grade. If you are having difficulties with the assignment, course staff are there to help (provided it isn't last minute).

The Academic Integrity Declaration must be signed and submitted on time or the assessment will not be marked.

Please read `http://www.student.cs.uwaterloo.ca/~cs240/s21/guidelines.pdf` for guidelines on submission. **Each question must be submitted individually to MarkUs as a PDF** with the corresponding file names: a1q1.pdf, a1q2.pdf, ... , a1q5.pdf, pq3.cpp .

It is a good idea to submit questions as you go so you aren't trying to create several PDF files at the last minute.
Remember, late assignments will not be marked.
Late assignments, however, can be reviewed for **feedback only** upon request to the ISAs at cs240@uwaterloo.ca.
Note: you may assume all logarithms are base 2 logarithms: $\log = \log_2$.

Problem 1 [4+4+4+4=16 marks]

Provide a complete proof of the following statements from first principles (i.e., using the original definitions of order notation).

a) $42n^4 - 13n^2 + 7 \in \Theta(n^4)$

b) $n^2(\log n)^{1.0001} \in \Omega(n^2)$

c) $\frac{n^2}{n+\log n} \in \Theta(n)$

d) n^n is $\omega(n^{13})$

Problem 2 [4+4+4=12 marks]

For each pair of the following functions, fill in the correct asymptotic notation among Θ, o, and ω in the statement $f(n) \in \sqcup(g(n))$. Prove the relationship using any relationship or technique that described in class.

a) $f(n) = n^4(7 + 3\cos 2n)$ versus $g(n) = 7n^4 + 5n^3 + 3n$

b) $f(n) = \sqrt{n}$ versus $g(n) = (\log n)^5$

c) $f(n) = \log\log n$ versus $g(n) = (\log\log\log n)^5$

Problem 3 [4+4+4=12 marks]

Prove or disprove each of the following statements. To prove a statement, you should provide a formal proof that is based on the definitions of the order notations. To disprove a statement, you can either provide a counter example and explain it or provide a formal proof.

a) For positive functions $f(n)$ and $g(n)$, $f(n) \notin o(g(n))$ and $f(n) \notin \omega(g(n)) \Rightarrow f(n) \in \Theta(g(n))$

b) $(\log n)^{\log n} \in O(n)$.

c) $\log n \times 2^{\sin(n^3)} \in O(n)$.

Problem 4 [4+4+4=12 marks]

Analyze the following piece of pseudocode and give a tight (Θ) bound on the running time as a function of n. Show your work. A formal proof is not required, but you should justify your answer (in all cases, n is assumed to be a positive integer).

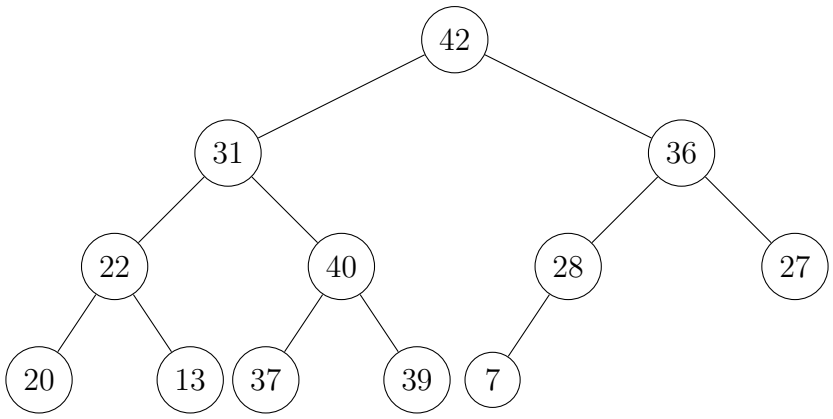
a)
```
s := 2021
for i := 1 to n + 42 do
   s := s * 4
   for j := 1984 to 2021 do
      for k := 5i to 7i - 1
         s := s * 42
```

b)
```
for i := 1 to n do
   for j := 1 to i * i do
      for k := 1 to log(n)
         s := s + 1
```

c)
```
x := -42
y := 1
while y < 5n do
   z := n * n * n
   while z > y do
      x := 2 - x
      z := z - y
   y := y + 5
```

Problem 5 [4 or 4(+2), graded out of 4 marks]

An *almost-heap* is a binary tree that satisfies all heap-properties except that at one item the order-property may be violated. Thus, it consists of an array A and one index i such that $A[j] < A[parent(j)]$ for all $j \neq i$. In the example below, the violated order-property occurs when $i = 4$, between 31 and 40.



Given i, where the violated order-property occurs, solve **one** of the following problems:
Note: If you give two solutions, we will only grade the first solution presented.

1) [4 marks] Show how to turn an almost-heap into a heap in time $O(\log^2 n)$.

2) [4(+2) marks] Let h be the height of the almost-heap and ℓ be the level that contains node i. Show how to turn an almost-heap into a heap in time $O((\ell + 1)(h - \ell + 1))$.

Use the following notation: the root of T is node $root(T)$, and the rightmost node in the last layer is $last(T)$. The key stored in a node i is denoted by $key(i)$; the parent, left and right child of a node i is denoted $parent(i)$, $left(i)$ and $right(i)$ respectively.

Problem 6 [14 marks]

In this programming question, you are asked to implement a priority queue in 3 ways, using C++. An item in the priority queue will have an integer priority and an integer value. The value is a timestamp recording the order in which items have been inserted; the first item inserted will have value 1, the second item value 2, and so on. Each implementation of the priority queue must support the insert and deleteMax operations, as well as, lookMax which returns the item with the highest priority (but does not remove it from the data structure).

Implementation 1: A singly linked list that is ordered by priority where the largest priority is in the front of the list. If two items have the same priority, then the tie is broken by ordering the item with the smallest value first.

Implementation 2: A max-heap implemented with a dynamic array (as described in the course notes). When the first item is inserted, an array of length 1 is dynamically allocated to accommodate the first item. You must then use the doubling strategy to reallocate the dynamic array when more space is required. Similarly, if after removing an item, only 25% or less of the dynamic array contains items, the size of the dynamic array should be reallocated to be twice the number of items in the dynamic array.
Note: for this implementation, you may not use a vector. You must implement the reallocation and doubling strategy yourself.

Implementation 3: A vector of queues. Each index i of the vector points to a queue of items where each item in the queue has priority i; if no items of priority i exist, this pointer may be left as NULL (i.e. you do not need to create an empty queue). At all times, the vector must be appropriately sized to be 1 + largest priority stored in the priority queue. You are free to choose the implementation of the queue at each vector index - a circular vector, linked list with back pointer, or std::queue is okay. The insert/delete (enqueue/dequeue, pushback/popfront) queue functions must run in O(1) time.

You may not use smart pointers or any built in data structures (unless specified) that may trivialize the implementations. You may use std::pair in any of the three implementations and std::vector may only be used in implementation 3. You may not use std:swap or anything from the algorithms library. If in doubt, please ask. You must also manage the dynamically allocated memory yourself (where applicable) and must free all memory before your program terminates.

In all implementations, lookMax must have runtime O(1).

At the top of your program, in comments, include a section for the algorithm analysis of each implementation. You may assume that n is the number of elements currently in the priority queue. For the algorithm analysis:

- define any additional variables that are required for the analysis

- state and briefly justify the running time of insert and deleteMax for each implementation
- state and briefly justify the amount of space required for each implementation

Implement your program in C++ and provide a main function that accepts the following commands from stdin (you may assume that all inputs are valid). You may assume the priority queues described above are numbered 1, 2 and 3, respectively.

- `i num priority` - inserts an item (`priority, timestamp`) into priority queue `num`.
- `d num` - removes the item with the highest priority from priority queue `num` and prints the priority and timestamp to stdout - the two integers are separated by a space and a newline follows the second integer. If the priority queue is empty, does nothing.
- `l num` - prints (but does not remove) the item with the highest priority from priority queue `num`. Prints the priority and timestamp to stdout - the two integers are separated by a space and a newline follows the second integer. If the priority queue is empty, does nothing.
- `r` - initializes or resets all priority queues to be empty.
- `x` - terminates the program.

Place your entire program in the file `pq3.cpp`