CS 201 Data Structures Library Phase 3 Due 11/19

Draft

Phase 3 of the CS201 programming project, we will be built around heaps. In particular, you should implement both a standard binary heap and Fibonacci heap. You will implement a class for each heap type.

You should create a class named Heap for the binary heap with public methods including the following (keytype is the type from the template). You should use your dynamic array class for the heap storage.

Function	Description	Runtime
Heap();	Default Constructor. The Heap should be empty	O(1)
Heap(keytype k[], int s);	For this constructor the heap should be built using the array K containing s items of keytype. The heap should be constructed using the O(n) bottom up heap building method.	O(s)
~Heap();	Destructor for the class.	O(1)
keytype peekKey();	Returns the minimum key in the heap without modifiying the heap.	O(1)
keytype extractMin();	Removes the minimum key in the heap and returns the key.	O(lg n)
void insert(keytype k);	Inserts the key k into the heap.	O(lg n)
void printKey()	Writes the keys stored in the array, starting at the root.	O(n)

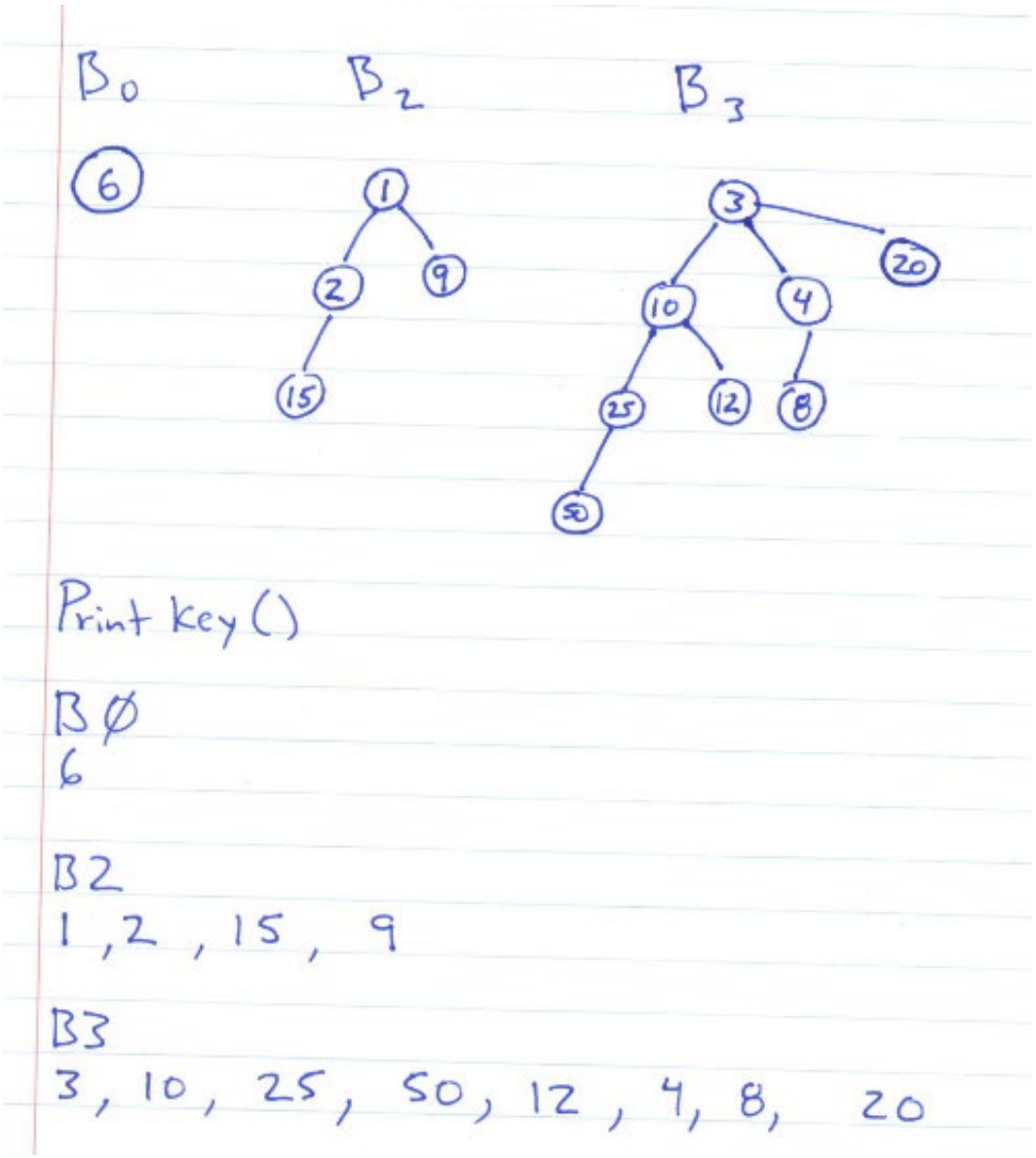
Your class should include proper memory management, including a destructor, a copy constructor, and a copy assignment operator.

You should create a class named FibHeap for the Fibonacci heap with public methods including the following (keytype is the type from the template). You should use dynamic allocation for the binomial trees. Note that in order to perform peekKey in O(1) time, you will need to maintain a pointer to the minimum value in the heap. We will not be implementing decrease-key, so there is no need to implement marked nodes, and the trees stored in the Fibonacci heap will always be binomial trees.

In order to make the structure of the heap unique, here are some rules to follow. The root list of the heap is a doubly linked list with head and tail pointers. All operations that add items to the root list should do so at the tail of the list, including merge and extract-min. The consolidate process should start at the head of the list working toward the tail, and at the end of consolidate the smaller binomial trees should appear before (closer to the head of the list) the larger binomial trees.

Function	Description	Runtime
FibHeap();	Default Constructor. The Heap should be empty	O(1)
FibHeap(keytype k[], int s);	The heap should be built using the array k containing s items of keytype. Once all the data is in the heap, a single call of consolidate should be used to form the binomial trees.	O(s)

~FibHeap();	Destructor for the class.	O(n)
keytype peekKey();	Returns the minimum key in the heap without modifying the heap.	O(1)
keytype extractMin();	Removes the minimum key in the heap and returns the key.	O(lg n) amortized
void insert(keytype k);	Inserts the key k into the heap.	O(1) amortized
void merge(FibHeap<keytype> &H2);	Merges the heap H2 into the current heap. Consumes H2.	O(1) amortized
void printKey()	Writes the keys stored in the heap, starting at the head of the list. When printing a binomial tree, print the size of tree first and then use a modified preorder traversal of the tree. Example below.	O(n)



For submission, all the class code should be in a files named Heap.cpp and FibHeap.cpp. You can either place your CDA code directly into the Heap.cpp and/or BHeap.cpp files or have an include statement in each file. If you use an include statement then please place the include in all files that need that code, and in that case add a #ifndef directive in your CDA file and also add the CDA.cpp file to your zip submission. Create a makefile for the project that compiles the file Phase3Main.cpp and creates an executable named Phase3. A sample makefile is available on Blackboard. Place Heap.cpp, BHeap.cpp and makefile into a zip file and upload the file to Blackboard.

- ☐ Create your Heap and FibHeap classes
- ☐ Modify the makefile to work for your code (changing compiler flags is all that is necessary)
- ☐ Test your Heap and FibHeap classes with the sample main provided on the cs-intro server
- ☐ Make sure your executable is named Phase3
- ☐ Develop additional test cases with different types, and larger arrays
- ☐ Create the zip file with Heap.cpp, FibHeap.cpp, CDA.cpp (if needed) and makefile
- ☐ Upload your zip file to Blackboard

No late submissions will be accepted.