Project Number: 2

Project Title: CS241 Summer 2018 Project 2: Heaps

Class: CS 241

Diane Margo

[dmmargo@cpp.edu](mailto:dmmargo@cpp.edu)

July 23, 2018

Written description:

Section 1: Project Description

This project takes in a user’s input for how large the heap will be for an array to store a certain number of numbers. With this array, there is upheap for when a user adds numbers. Therefore, there needs to be a swap method. Along with that, there is a checker for if the user has already inputted the same number, which is the addCheck method in this program. Being able to check the numbers, users can add, remove, reheap, move, get the number of times swapped, and print values that are the first 10 numbers of the current heap at the time.

Section 2: Project Specification

The goal of this project is to work with max heap. Starting with the implementation of a heap, the heap requires the user to input a size for the array. The lastIndex and number of swaps is set to zero here.

Next, we have the Boolean to check if there has been a duplicate in the numbers. It’s good for checking optimal. This method checks to see if the user can or cannot add to the existing array. If the user can add, the program will place the new number into the heap at the lastIndex’s number and increment the number of indexes existing by one. It then returns true for being able to add successfully.

Furthermore, we have the boolean add method that does add and swapping. If the number cannot be added due to the number being the same in some part of the array or having the last index’s number greater than the heap’s length minus one, then it returns false. If not, the program will continue to make i the last Index and add the number to the array called heap. Afterwards, it gets the parent’s location in the array by following the formula (n-1)/2 so that while the lastIndex’s number is not equal to 0 and the number provided is greater than the parent, the program will continue to do swaps (upheap). At the end of this method, the program returns true and the lastIndex is incremented by one because it was successful in adding another number to the array.

The same method takes in a number and figures out if the number exits already or not. It goes through a for loop and if there is no number that equals to the heap of index i, then it returns false. If there is an indicator that something is the same, it returns true.

Additionally, there is the remove method that takes heap index 0, and sets it to the last index -1, moving the last node added to the beginning of the heap. The program then calls move method to move and proceeds to decrease the number of last index.

Next is the reheap method that finds the bottom of the levels from the number. It keeps track of the level(height) at which the user is at in the program. It then uses calculations to find the height using the function Math.ceil, taking the ceiling of the Math.log(double)( j + 1)) / Math.log(2.0)); j is the last index minus one. A new level is made and put into a while statement to check with height. If figures out how many children the one root with child(ren). With this, it compares and moves to the next height.

The move method helps to optimize move down swapped nodes. This method looks for bounds and checks before trying to access the heap. Afterwards, it does two types of switches depending on how big one index is to another.

The swap method just returns the integer that is swapped.

Last is the print method which prints the first n values of the heap. The ‘n’ value is given by the user.

Section 3: Testing Methodology

To ensure that I wouldn’t run into problems, I looked at the book/slides for reference and looked online to see how the methods were implemented into the code, what kind of return values they should have, and what kind of operations they did. To make sure there would be no errors in what would be returned, all variables were set to a default number or null. Another thing is to check that the signs are right.

First, checking to make sure that each method works one by one is the best way to ensure that one does not get errors. I did the addCheck first to get the basis on how to check for other numbers existence in the array. Since the function takes care of adding and checking through another method same, it makes sure that there wouldn’t be errors as far as duplicates go. The add method is something like addCheck, but it also provides the user with swapping function to upheap. We need to make sure that the index number is always updated so that the last Index can be up to date for other methods.

Later, I did the Search, remove and reheap. The remove method is simple because it follows the rule that the last thing added into the array will be placed in the front of the array and asked to be reheaped later on in the main program. Removing is just making sure that the index number is updated as add was. Reheap took some math calculations. In order to get the right number, I looked up the way to get height, which is taking the ceiling. The reheap method also comes with checking to see how many children a root has. Getting the parent is a simple formula (n-1)/2.

The move method needs to check for bounds to make sure that is within the range of the array. Depending on the number’s placements, the numbers would be moved or shifted down a certain way by swapping. This would count to how many swaps there are made.

There wasn’t anything to check for Swap. It simply returns a number that is updated from the fields. And print will print he last whatever number the user programmed into their code. It just needs to check to make sure that the user won’t go above the n number of values OR the last index number because the user could be putting a number over the number of last indexes.

The driver requires random numbers and user input. Therefore, it was a good idea to implement the keyboard for scanner and random first. Another thing is did was put an option for invalid input if the user didn’t put in a 1 or 2. A lot of the code for the main function was mostly for loops so that it could go through the heap. by initializing I and other variables, it helps prevent errors.

Section 4: Lesson Learned

From this project, I learned to make a max heap with new methods and checking. I learned a lot from the reheap method: finding the ceiling of a value, having to figure out how many children and how to swap according to how many children. Also, during the add method, I learned a lot about making parents a part the method for upheaping purposes.

Organization of the max-heap is interesting because the largest value will always be on the top, and the last node at the bottom will eventually be replaced at the beginning as it would at the top when we practice in class. The reheaping process is the same, but in code, there needs to be an account for the height, how many levels the program is at when figuring things out.

Knowing the last index count in this code is the most important thing to update. Without it, the bounds would be messed up, removing entries would not be accurate, and adding entries would be the same too. The process of upheaping and down heaping in program takes a lot of while-loop checks to stay in bounds.

Lesson 5: Analysis of Output

The output is the average swaps for a series of interactions along with swaps for optimal methods given that there are 20 sets of randomly generated integers that were computed and documented the average number of swaps for both methods. The second thing it outputs is the first 10 integers in the away and the number of swaps for both methods, later performing 10 removals on the heap and output the 10 integers that were just removed.

Program Output:

 