CSCD 320: Algorithms
By: Bin Mei
May 15th, 2014

Picking N Largest Numbers Using Minimum Heap from a Large Data Set

Abstract:

In this project, a minimum heap is used to select 10,000 largest numbers from the given data set. A minimum heap is used because it utilizes the binary tree structure, and has a time complexity of O(log n) for min-heapify functions, as well as a time complexity of O(n log n) for building. Such time complexity is considered fair for a large data set.

Introduction:

Large data manipulation has long been a important field of study in Computer Science. As the number of data rises, the computing time required to manipulate these data also increases. Therefore, if the data is sufficiently large, may result in the cases where the normal iterative algorithm never finish computing due to the excessively large time complexity and subsequent long computing time. Although some algorithms such as B-Tree would trade a computer's memory space for computing time, it is nevertheless limited by the memory space and the type of problems it can solve.

In light of these limitations, heap presents itself be an useful alternative solution. This is because heap utilizes both an array to store information, as well as a binary tree like structure to keep the information organized. In this project, a minimum heap is used because it would not only fit under the memory requirement constraint, it would also have a time complexity of O(log n) for insertion and deletion operations, which is very fast with the extreme large data set given by the premise of this project.

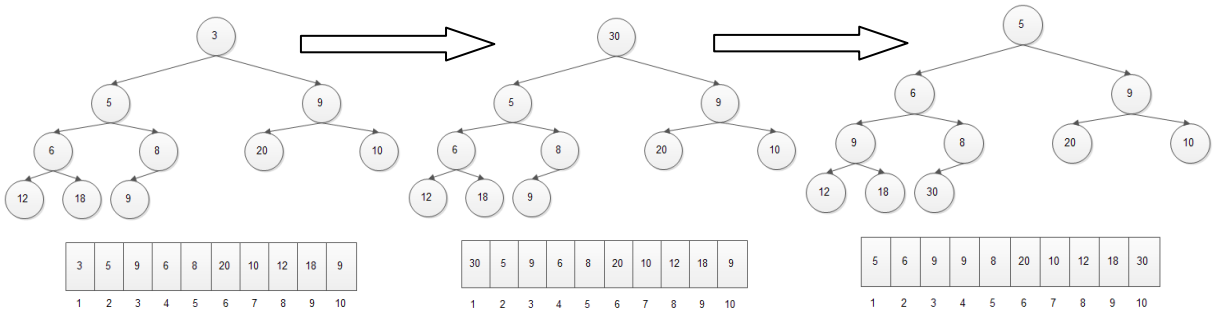
Algorithmic Idea and Detailed Algorithms:

The premise of the problem of this problem is to find the largest 10,000 numbers within a data set. Due to the strict memory constraint, a minimum heap of 10,000 numbers is built using the first 10,000 numbers in the given set. The building time of this initial should be O(n log n) because it would take O(n) time to traverse the first 10,000 numbers, and O(log n) time to insert it into the initial minimum heap.

After the initial heap is built, the algorithm would keep traversing the rest of the numbers in the set. For the 10001st number. The algorithm would compare it to the root of the min-heap. Since we are looking for the largest 10,000 number, if the 10001st number is smaller than the root of the min heap. We can just

keep going to the 10002nd number. However, if the 10001st number is larger than the root of our min-heap. We replace the current root of our min-heap with the 10001st number, and then use the heapify function to put it to its correct location in our heap. We will repeat this process until we have finished traversing the number set. Since it takes O(n) time to traverse the entire set, and O(log n) to insert the new numbers, the time complexity for this algorithm would therefore be O(n log n).

Using the 10 digit min-heap below as an example, we can see that 3 is the root of the array, and therefore the smallest number of that heap. If the incoming number(30) is larger than 3, then logically 3 can not be one of the largest number of the set. Therefore, 30 should become the new root of the min heap, and the heap would undergo maintenance to retain its heap structure such that after the shifting down process, 5 is now the new root of the min-heap.



It should also be noted that this algorithm works in the given memory constraint where the memory can not hold more than 10,000 numbers at a time. Using a minimum heap where only the root is compared with newly traversed number, the memory used by our min-heap should always be exactly of 10,000 numbers.

Strategy for Implementation:

The implementation strictly follows the algorithm idea above. Initially a heap of size 10,000 is created. After that, the numbers are parsed in from the data.txt file, and are inserted one by one into the heap. A counter is implemented to keep track of the inserted data size.

```
if (count < 10000){
   heap.insert(num);
}
```

When the inserted data size is larger than the heap size of 10,000. An else statement is used to insert the newly traversed number from the data set into the heap using the *insertAtRoot(num)* method.

```
else{
```

```
            heap.insertAtRoot(num);
        }
```

In the *insertAtRoot* method, the following algorithm is used insert the newly traversed number, ONLY if that number is larger than the root of the min-heap. Since the heap's data is stored in an array, that number therefore is data[0]. If that number is larger than the root, then it will become the new root, and subsequent *shiftDown(0)* method will be used to move the new number into its correct place in the heap. If the new number is not larger than the root, however, the program will simply move on to the next number in the data set.

```
        else if (data[0] < item){
            data[0] = item;
            shiftDown(0);
        }
```

Results:

With the included *data.txt* file, the program was able to pick 10,000 largest number numbers from a set of randomly generated 100,000 numbers. The 10,000 largest numbers should be in the range of the following set: `[9001461, 9001504, 9002124, …, 9428080, 9649400, 9962413]`

Conclusion:

Minimum heap is an effective way of solving this problem with a fair time complexity of O(n log n).