CS 3310 Data and File Structures Instructor

Dr. Ajay K. Gupta

Western Michigan University

Grad TA: Rajani Pingili

Anfaal Faisal

SOFTWARE LIFE CYCLE REPORT – FOR ASSIGNMENT #4

PHASE 1: SPECIFICATION

1. The main goal of this assignment was:
    - Understanding the mechanisms of priority queues in depth
    - Creating and manipulating heaps based on linked lists and arrays
    - Using priority queues to solve practical problems
    - Get experience with implementing tree-based data structures and their traversals
2. Practice good coding conventions

The application had to do the following:

- Read and convert data from text file
- Implement required function
    a. add(v): Adds value v to set S.
    b. getMedian(): Returns the current median value of the set.
    c. deleteMedian(): deletes upper median of the set for even cardinality, otherwise deletes median if it's an odd cardinality.
    d. sortUsingMedians() which returns a sorted array of length n, where n is the current cardinality of the set S as maintained
    e.

PHASE 2: DESIGN

1. Java Project
    - App
        - **public static void** main(**String[]** args) Entry point of program. Catch files with operations list
    - ArrayHeap
        - ArrayHeap() – Simple constructor
        - **public void** add(**int** v) – add data to heap, with array rebuilding, if needed

        - **public int** removeMax() – remove and return biggest value
        - **public int** getMax() – simple return biggest value
        - **public int** size() – return size of heap
        - **public String** toString() – return string view of heap
    - MedianQueue
        - **public** MedianQueue() – Simple constructor
        - **public void** add(**int** v) – add data to queue, with array rebuilding, if needed
        - **public int** deleteMedian() – remove and return median value
        - **public int** getMedian() – simple return medain value
        - **public String** toString() – return string view of heap
        - **public static int**[] sortUsingMedians(**int**[] data) – sort data, using specific mechanism
    - SortedLinkedList
        - Array Heap() – Simple constructor
        - **public void** add(**int** v) – add data to list, with array rebuilding, if needed
        - **public int** removeMin() – remove and return smallest value
        - **public int** getMin() – simple return smallest value
        - **public int** size() – return size of heap
        - **public String** toString() – return string view of heap
        -
    - Node – internal SortedLinkedList class
        - Node (int data) – Complicated constructor

This app provide testing and comparison of three different types of specific data string – heap, array and list.

.

Space complexity analysis

Analysis of space complexity for the code is summarized in the table below:

Method	Input space	Private fields space
App	1	1 private field
main	1	8 in-function variables
ArrayHeap	0	2
add	1	2-4
removeMax	0	0

getMax	2	0
size	0	0
toString	0	1
MedianQueue	0	4
sortUsingMedians	1	3
add	1	1
getMedian	0	1
deleteMedian	0	1
toString	0	1
SortedLinkedList	0	2
add	1	1-2
removeMin	0	1
getMin	0	0
size	0	0
toString	0	1
Node	1	1

Time complexity

Table of time complexity of methods

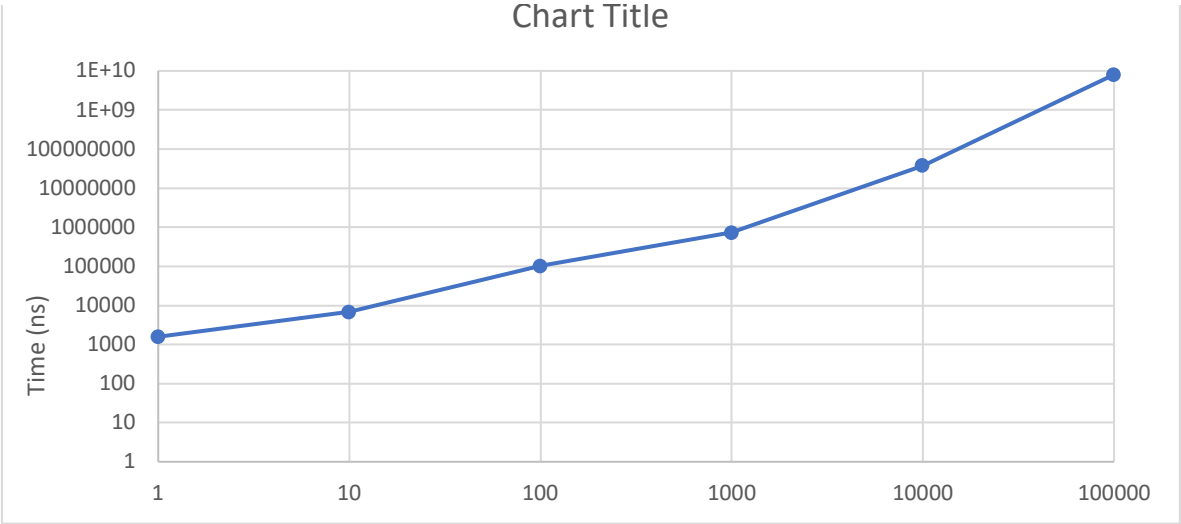
	Worst-Case	Average-Case	Best-Case	Space Complexity
getMedian	O(n log n)	O(n log n)	O(1)	O(n log n)
add	O(n log n)	O(log n)	O(n)	O(n)
deleteMedian	O(n log n)	O(log n)	O(n)	O(n)
sortUsingMedians	O(n log n)	O(log n)	O(n)	O(n)

Results

The empirical results agreed closely with the hand analysis as shown below. Each data point is the average of a 5 different instantiations of sorting the random array. The random numbers were from 0 to m included, where m is a maximal item array length..
System.nanotime() documentations shows that there may be some inaccuracies in measured time, so one non-logged circle of search was providing, during to some warming of Java Virtual Machine.

Each graph below show average time of sorting data. Y-axis represented in $\log_{10}$, for better readability.

First, simple hashing algorithm, which sums and multiply values of bytes of hashed string. Most slow algorithm



Second algorithm ,which turn string into char type, and sum/multiply their values. Average time

System.nanotime() was used to measure code execution time because System.militime() always returned a zero, since run time was less than a millisecond for small numbers of n.

Direct answer on questions

a. Explain how to perform the operation getMedian() in O(1) time given such a representation.
Only type of getMedian() complexity as O(1) is storing of median as separate value, and returning direct link of it in request
b. Explain how to perform the operation S.add(v) in O(log n) time, where n is the current cardinality of the set, while maintaining such a representation.
If S is stored using an array instead of two priority-queues, operation as S.ad () in worst-case in array is O(n). But, in addition to requested operations, in many cases it can call array rebuilding, which can increase complexity to O(n log n)
c. Explain how to perform the operation S.deleteMedian() in O(log n) time, where n is the current cardinality of the set, while maintaining such a representation.
If S is stored using an array instead of two priority-queues, operation S.deleteMedian() in worst-case in array is O(n). But, in addition to requested operations, in many cases it can call array rebuilding, which can increase complexity to O(n log n)
d. Design and analyze efficient algorithms to support these three operations when S is stored using an array instead of two priority-queues. Compare the theoretical time complexities of the two solutions. You do not need to implement this solution to find median of a set.
Theoretical time complexies for both solutions is similar, O(n log n)
h. A large sequence of add()/getMedian()/deleteMedian() is given in an input file hw4input.txt, measure the average time for add(), getMedian() and deleteMedian() operations from this sequence. Do these timings concur with the theoretical time complexities? If not, can you explain

why there is discrepancy? For this and the remaining, turn-off the interactive prompts from the console and execute the operations listed in the hw4input.txt file.

Data from hw4input.txt was processed and measured (data attached in file output.txt). In general time complexity similar wit those data, which obtained on theoretical calculations

(10 points extra credit): Using the same data values as in hw4input.txt file, use merge-sort to return a sorted array of length n. Compare this timing with sortUsingMedians()? What do you observe? Compare these two timings with merge-sort's theoretical time complexity O(n log n). What do you observe?

Result of timing added in file output2.txt

PHASE 3: RISK ANALYSIS

There are no risks associated with this application. There is a possible risk of a file not found exception.

PHASE 4: VERIFICATION

The algorithms were tested multiple times, and the searched items was printed out multiple times to verify that each step of the process was computed correctly.

PHASE 5: CODING

The code of this program is include in the zip file. The code is explained by in line comments or in Javadoc.

PHASE 6: TESTING

Console text, which was used to testing behavior of program, showed in file output.txt. Due to very big amount of text data, all screen messages are turned into txt file.

PHASE 7: REFINING THE PROGRAM

No refinements are needed for this program.

PHASE 8: PRODUCTION

A zip file including the source files from eclipse and the output of the program have been included.

PHASE 9: MAINTENANCE

This application can be further improved once feedback from the grader is obtained.