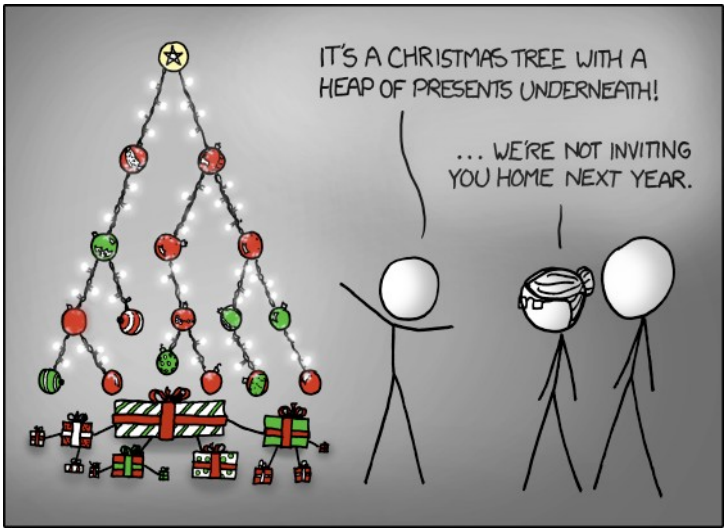
CS 302 – Assignment #8

Purpose: Learn about and implement priority queues.
Due: Monday (10/29) → Must be submitted on-line before class.
Points: 125 Part A → 75 pts, Part B → 50 pts

Assignment - Part A:

A fairly famous algorithm problem is the dynamic median problem. Specifically, you are given a set of integers coming from user input. Each number you get in a value in the data set. You have to find the dynamic median of the set of integers. The dynamic median is the midpoint value of the sorted set of number seen so far, or the average of the two mid points if the number of elements in the data set is even.



Source: www.xkcd.com/835

One of the most effective ways of solving this is heap data structure.[1] We will maintain two heaps, a min-heap and a max heap. When a number comes we will first compare it with the current median and put it to the appropriate heap.

Develop a polymorphic *minHeap* and *maxHeap* priority queues. *Note*, these priority queues may be used on a future assignment and should be written in a generic manner. When completed and tested, develop a polymorphic *dynamicMedians* class that uses the *minHeap* and *maxHeap* to solve the problem. A main is provided that call the dynamicMedians class for testing. In addition, the provided main tests the heap insert-one-at-a-time vs the heapify process.

Part B:

When completed, create and submit a write-up (PDF format) not too exceed ~750 words including the following:

- Name, Assignment, Section
- Summary of the implemented priority queue data structure.
- Big-Oh for the priority queue operations (insert, deleteMin, deleteMax, reheapUp, reheapDown, heapify, and printHeap).
- Big-Oh for the dynamic medians algorithm.
- Explain the difference between repeated inserting and using heapify.

Submission:

- Submit a compressed zip file of the program source files, header files, and makefile via the on-line submission by 4:00 PM.

All necessary files must be included in the ZIP file. The grader will download, uncompress, and type **make** (so you must have a valid, working *makefile*).

1 For more information, refer to: https://en.wikipedia.org/wiki/Heap_(data_structure)

Class Descriptions

- Minimum Priority Queue Class
  The minimum priority queue template class will implement functions specified below.

minHeap<myType>
-count: unsigned int
-heapSize: unsigned int
-*myHeap: myType
+minHeap(unsigned int=1000)
+~minHeap()
+getCount() const: unsigned int
+insert(const myType): void
+deleteMin(): myType
+peek() const: myType
+isEmpty() const: bool
+printHeap() const: void
+makeRandHeap1(const unsigned int, const unsigned int): void
+makeRandHeap2(const unsigned int, const unsigned int): void
-reheapUp(unsigned int): void
-reheapDown(unsigned int): void
-heapify(): void
-resize(): void

Function Descriptions

- The *minHeap()* constructor should initialize the binary heap to an empty state. The parameter must be checked to ensure it is at least 1000 and, if not, use the default value.
- The *~minHeap()* destructor should delete the heap.
- The *getCount()* function should return the total count of elements in the heap.
- The *insert()* function should insert an entry into the binary heap. If the count of heap entries exceeds the heap size, the heap must be expanded via the private *resize()* function. The heap properties must be maintained via the private *reheapUp()* function. The count should be updated.
- The private *heapify()* function should update the heap to apply the heap properties.
- The *isEmpty()* function should return true if there are no elements in the heap and false otherwise.
- The *printHeap()* function should print the current heap in level order with a blank line between each level. Refer to the sample output for an example of the formatting.
- The *deleteMin()* function should remove and return the minimum entry from the heap. The heap properties must be maintained via the private *reheapDown()* function. Additionally, the count should be updated. If the heap is already empty, the function should display an error message "**`Error, min heap empty.`**" and return {}.
- The *reheapUp()* function to recursively ensure the heap order property is maintained. Starts at tree leaf and works up to the root. Must be written recursively.
- The *reheapDown()* function to recursively ensure the heap order property is maintained. Starts at the passed node and works down to the applicable leaf. Must be written recursively.

- The *peek()* function should return the current to of the heap without making any changes to the heap.
- The *resize()* function should create a new heap array twice the size of the existing heap, copy all entries from the current heap into the new heap, and delete the old heap. The *heapSize* should be updated accordingly.
- The *makeRandHeap1()* function should initialize the random number generator with 73 (i.e., via the standard *srand(73)* call) and inset the passed number of random numbers into the heap by using the insert() function with the passed limit (rand()%limit).
- The *makeRandHeap2()* function should initialize the random number generator with 73 (i.e., via the standard *srand(73)* call) and inset the passed count of random numbers modulo the passed limit (i.e., *rand()%limit*) directly into the *myHeap[]* array and then call the *heapify()* function.

- Maximum Priority Queue Class
  The maximum priority queue template class will implement functions specified below.

maxHeap<myType>
-count: unsigned int
-heapSize: unsigned int
-*myHeap: myType
+maxHeap(unsigned int=1000)
+~maxHeap()
+getCount() const: unsigned int
+insert(const myType): void
+deleteMax(): myType
+peek() const: myType
+isEmpty() const: bool
+printHeap() const: void
+makeRandHeap1(const unsigned int, const unsigned int): void
+makeRandHeap2(const unsigned int, const unsigned int): void
-reheapUp(unsigned int): void
-reheapDown(unsigned int): void
-heapify(): void
-resize(): void

Function Descriptions

- The *maxHeap()* constructor should initialize the binary heap to an empty state. The parameter must be checked to ensure it is at least 1000 and, if not, use the default value.
- The *~maxHeap()* destructor should delete the heap.
- The *getCount()* function should return the total count of elements in the heap.
- The *insert()* function should insert an entry into the binary heap. If the count of heap entries exceeds the heap size, the heap must be expanded via the private *resize()* function. The heap properties must be maintained via the private *reheapUp()* function. The count should be updated.
- The private *heapify()* function should update the heap to apply the heap properties.
- The *isEmpty()* function should return true if there are no elements in the heap and false otherwise.

- The *printHeap()* function should print the current heap in level order with a blank line between each level. Refer to the sample output for an example of the formatting.
- The *deleteMax()* function should remove the maximum entry from the heap. The heap properties must be maintained via the private *reheapDown()* function. Additionally, the count should be updated. If the heap is already empty, the function should display an error message "**`Error, max heap empty.`**" and return {}.
- The *peek()* function should return the current to of the heap without making any changes to the heap.
- The *reheapUp()* function to recursively ensure the heap order property is maintained. Starts at tree leaf and works up to the root. Must be written recursively.
- The *reheapDown()* function to recursively ensure the heap order property is maintained. Starts at the passed node and works down to the applicable leaf. Must be written recursively.
- The *resize()* function should create a new heap array twice the size of the existing heap, copy all entries from the current heap into the new heap, and delete the old heap. The *heapSize* should be updated accordingly.
- The *makeRandHeap1()* function should initialize the random number generator with 73 (i.e., via the standard *srand(73)* call) and inset the passed count of random numbers into the heap by using the insert() function with modulo the passed limit (*rand()%limit*).
- The *makeRandHeap2()* function should initialize the random number generator with 73 (i.e., via the standard *srand(73)* call) and inset the passed count of random numbers modulo the passed limit (i.e., *rand()%limit*) directly into the *myHeap[]* array and then call the *heapify()* function.

- Dynamic Medians Class
  The dynamic medians template class will implement functions specified below.

dynamicMedians<myType>
-arrLength: int
-*myData: myType
-*mediansArr: myType
-MIN_LENGTH = 5: static const unsigned int
+dynamicMedians()
+~dynamicMedians()
+setData(myType [], const int): bool
+generateData(const int, const int): bool
+showData() const: void
+showMedians() const: void
+getDataItem(const unsigned int) const: myType
+getMediansItem(const unsigned int) const: myType
+findMedians(): void

Function Descriptions

- The *dynamicMedian()* constructor should initialize the data and medians to an empty state.
- The *~dynamicMedians()* destructor should delete the dynamically allocated arrays.
- The *getDataItem()* function should return the data value at the passed index.

- The *setData()* function should allocate the data array and set the values from the passed array. The function should ensure that passed size is ≥ to the MI_LENGTH. If not, display an error message "**`Error, invalid size.`**".
- The *generateData()* function should allocate the data array and populate the array with random numbers. The size (first argument) and the limit (second argument) are passed. The function should ensure that passed size is ≥ to the MIN_LENGTH. If not, display an error message "**`Error, invalid size.`**". If the size is valid, the random numbers should be generated (via *rand()%limit*).
- The *showData()* function show all items in the data array (with one space between each number). Refer to the sample output for formatting examples.
- The *showMedian()* function show all items in the median array (with one space between each number). Refer to the sample output for formatting examples.
- The *getMedianItem()* function should return the median value at the passed index.
- The *findMedians()* function should dynamically create the medians array and determine the dynamic median using the provided algorithm.

Refer to the example executions for output formatting. Make sure your program includes the appropriate documentation. See Program Evaluation Criteria for CS 302 for additional information. ***Note, points will be deducted for especially poor style or inefficient coding.***

Dynamic Medians Algorithm

We need two heaps, a maximum heap and a minimum heap, each containing roughly half the elements in the data set. The idea is to update maximum heap and minimum heap in such a way that we can always figure out the median value from the two heads.

- Create a minimum heap and a maximum heap and initialize current median to -1
- Iterate to obtain element (until no more elements).
  - If the count of the maximum heap and minimum heap heaps are the same
    - If the new element is less than the current median
      - Insert the new element into the maximum heap
      - Set the current median to the max heap top (without changing it)
    - Otherwise
      - Insert the new element into the minimum heap
      - Set the current median to the min heap top (without changing it)
  - If the count of the maximum heap is greater than the minimum heap
    - If the new element is less than the current median
      - Remove the top of the maximum heap and insert it into the minimum heap
      - Insert the new element into the maximum heap
    - Otherwise
      - Insert the new element into the minimum heap
    - Set the current median to the integer average of the two heap tops (without changing either heap)
  - If the count of the minimum heap is greater than the maximum heap
    - If the new element is less than the current median
      - Insert the new element into the maximum heap
    - Otherwise
      - Remove the top of the minimum heap and insert it into the maximum heap
      - Insert the new element into the minimum heap
    - Set the current median to the integer average of the two heap tops (without changing either heap)
  - Place the current median in the medians array.

Make File:
You will need to develop a make file. You should be able to type:

```
make
```

Which should create the executables. The makefile should be able to build both executables; one for the testing (i.e., pqTest) and the other for the main.

Test Script
A test script which will be used for scoring the final submission and is provided for reference.

Example Execution – Testing Program:
Below is an example program execution for the testing program.

```
ed-vm% ./pqTest
*****************************************************************
CS 302 - Assignment #8
Priority Queue Tester


=================================================================
Min Heap - Test Set #0
-----------------------------------
PQ Heap (level order):
amazon

apple
cisco

belkin
dell
oracle
newegg

jupiter
google
ebay

-----------------------------------
PQ Heap Size: 10
Priority Order:
amazon
apple
belkin
cisco
dell
ebay
google
jupiter
newegg
oracle

=================================================================
Min Heap - Test Set #1
-----------------------------------
PQ Heap (level order):
a

b
c

d
f
g
```

```
j

e
l
n
h
i
o
z
t

x
s
w
u
y
v
p
k
r
m
q

---------------------------------------
PQ Heap Size: 26
Priority Order:
a b c d e f g h i j k l m n o p q r s t u v w x y z

=====================================================================
Min Heap - Test Set #2
Min Heap - Large test successful.

=====================================================================
Max Heap - Test Set #0
---------------------------------------
PQ Heap (level order):
oracle

jupiter
newegg

dell
ebay
google
cisco

amazon
belkin
apple

---------------------------------------
PQ Heap Size: 10
Priority Order:
oracle
newegg
jupiter
google
ebay
dell
cisco
belkin
apple
amazon

=====================================================================
Max Heap - Test Set #1
---------------------------------------
PQ Heap (level order):
z
```

```
x
y

u
w
q
t

r
s
v
k
m
o
g
j

b
e
n
l
p
f
d
h
a
i
c

---------------------------------------
PQ Heap Size: 26
Priority Order:
z y x w v u t s r q p o n m l k j i h g f e d c b a

================================================================
Max Heap - Test Set #2
Max Heap - Large test successful.

================================================================
Should show heap empty errors.

Error, min heap empty.
Error, max heap empty.

****************************************************************
Game Over, thank you for playing.
ed-vm%
```

Example Execution:

Below is an example program execution for the main program.
Note, the **ed-vm%** is prompt on my machine. Your prompt will be different.

```
ed-vm%
ed-vm%
ed-vm% ./main
CS 302 - Assignment #8
Dynamic Medians Program.
************************************************************

------------------------------------------------
Data Set 0

Data:
-----
5 15 1 3 2 8 7 9 10 6 11 4
```

```
Dynamic Medians:
----------------
5 10 5 4 3 4 5 6 7 6 7 6


--------------------------------------------------
Data Set 1

Data:
-----
383 886 777 915 793 335 386 492 649 421

Dynamic Medians:
----------------
383 634 777 831 793 785 777 634 649 570


--------------------------------------------------
Data Set 2

Selected Data:
--------------
202362 546509 243507 745566 772925 759209 103727 960907 789737 313996

Selected Medians:
-----------------
202362 501595 501218 501290 501210 500881 500625 500150 499995 500185

--------------------------------------------------
Min Heap Tests

on-at-a-time inserts took: 1946 milliseconds
heapify inserts took: 1584 milliseconds


Verification in process, standby…

--------------------------------------------------
Max Heap Tests

on-at-a-time inserts took: 1909 milliseconds
heapify inserts took: 1459 milliseconds

Verification in process, standby...

**********************************************************
Game Over, thank you for playing.
ed-vm%
```